AN ALTERNATIVE PERIPHERAL EXECUTIVE FOR THE DATA GENERAL AOS/VS OPERATING SYSTEM

submitted in fulfillment of the requirements for the degree of

Master of Science

by

Robert Satchwell Tennant
Department of Computer Science
Rhodes University

ACKNOWLEDGEMENTS

My thanks to James Trew for his assistance with the unravelling of the interface between EXEC and PMGR. Without this information MCP would not have been possible.

My thanks also to all those people in Middelburg Steel and Alloys who gave so freely of their time and ideas in the creation and testing of MCP, particularly Charles Nell, Kassie Kasselman, Mark Tarlton and Carn Iverson.

Thanks also to my supervisor, Peter Clayton, for his assistance and encouragement during this year.

TABLE OF CONTENTS

TABLE OF CONTENTS continued

GLOSSARY OF TECHNICAL TERMS

The following is some of the more important Data General terminology used in this thesis.

TERM	MEANING
AOS/VS	Advanced Operating System / Virtual Storage. The standard operating system for 32 bit Data General computers.
AGENT	The section of AOS/VS that processes system requests.
AG	Address Generator. Part of the Data General CPU.
ALU	Arithmetic Logic Unit. Part of the Data General CPU.
ATU	Address Translation Unit. Part of the Data General CPU.
BMC	Burst Multiplexor Channel. The high speed bus.
CB	Control Block. Used by the scheduler to schedule system tasks.
CLI	Command Line Interpreter. The Data General version of COMMAND.COM.
DG	Data General.
DCH	Data CHannel. The slow bus.
DCT	Device Control Table. Used by AOS/VS for device interrupts.
DMA	Direct Memory Access.
EXEC	The section of the operating system that provides the real time environment.
FPU	Floating Point Unit. Part of the CPU.
GSMGR	Global Synchronous ManaGeR. Used for synchronous communications.
IAC	Intelligent Asyncronous Controller. The hardware used for asyncronous communications.
ILC	Intelligent Lan Controller. The hardware used for Local Area Networking.
IP	Instruction Processor. Part of the CPU.
IPC	Inter Process Communications port. Used for information transfer between processes.
MCP	Multi-user Control Program. The subject of this thesis.
MCU	Memory Control Unit. Part of the CPU.
NADGUG	North American Data General User Group.
PMGR	Peripheral ManaGeR. The section of the operating system that controls I/O to asyncronous peripherals.

RMA Remote Management Agent. A section of the XODIAC networking software that controls file transfer between computers.

SVTA Server Virtual Terminal Agent. A section of the XODIAC networking software that controls terminal access across the network.

TCB Task Control Block. Used to schedule tasks in a process.

XODIAC The Data General local area networking software.

X25 The section of XODIAC that handles the communication protocol.

1. Introduction

The last decade has seen major advances in computer technology. This has created a problem for computer vendors, namely how to keep the operating system in line with the hardware. The problem is compounded by the fact that "patching" compromises system integrity. The smaller computer companies such as Data General (DG) are the worst affected due to their limited manpower resources.

The action most commonly taken is to limit operating system changes to the bare minimum consistent with the new hardware, and to write add-ons instead. An example of this in the Data General environment is the XODIAC[1] suite of programs to support Local Area Networking. The major problems with add-on software are losses in efficiency and a fragmented work environment for the user.

An example of the lack of change in the Data General operating system is EXEC[1]. EXEC is that part of the system that creates the on-line environment for the users and supports the peripherals connected to the computer. The system has changed little in functionality or operation since it was introduced for the 16 bit computers of the late 1970s, even though Data General computers and peripherals have advanced technologically by orders of magnitude since then. Consequently the intelligence of modern peripherals, such as laser printers and PCs[2], is not exploited. Furthermore the on-line environment, while adequate for the low user counts of the late 1970s, is limited and difficult to manage on the large machines of today.

This thesis describes MCP[3], a replacement system for EXEC, written to address these problems. MCP was developed specifically for the Steel Division of Middelburg Steel & Alloys (Pty) Ltd. (MS&A), who are installing a network of 14 Data General processors including an MV20000/2, 3 x MV15000/10, 2 x MV15000/8, 1 x MV4000 and

---

[1] Product of Data General Corporation

[2] Personal Computers compatible with IBM PC, XT or AT

[3] MCP is an acronym for Multi-user Control Program

7 x MV2000[4]. Peripherals to be connected to these machines include; 80 printers, 120 terminals and 240 PCs. Due to the size of this installation, a uniform user environment and comprehensive system management capability was required. MCP has replaced EXEC for all applications within the Steel Division of MS&A, and has subsequently been adopted by the other divisions of the company.

MCP endeavours to meet two areas of need, those of the user and those of the system manager. Much of the motivation behind these needs is contained in a paper [TEN87] presented by the author at the Las Vegas NADGUG[5] conference. In summary, the user's needs are met via the following :

A terminal environment allowing on-line usage of the machine with transparent connection to a number of networked machines.

A PC environment allowing record or file transfers between any two machines (DG or PC) in the network, transparent printer output re-direction from PC packages to Data General printers, control of DG print queues from the PC, print pass through from a Data General machine to PC printers and DG terminal emulation.

A printer environment allowing output to be queued to any printer on the network with the programmable features of that printer selectable via switches.

A batch processing environment allowing automatic queuing and prioritisation of large reports, selection of job classes and control over sequencing of jobs.

The system manager's needs are met via the following :

A security system allowing many levels of terminal access, with hooks into application programs allowing access control to be extended to the program menu and sub-menu level.

---

[4] The MV series are 32 bit processors varying in size from 11 MIP (MV20000/2) to 1 MIP (MV2000).

[5] North American Data General User Group

A powerful operator command syntax to allow for easy control of the system.

An accounting system based on performance criteria, integrated with the communications and systems programming documentation records. This ensures accuracy and allows control of system usage.

A systems management toolbox, including; a user security profile editor, a real time system monitoring package, a menu driven systems management interface, event logging and reporting, fail safe traps and emergency shutdown procedures.

In Chapters 3 to 9 these needs are discussed in more detail. Security and operator command syntax are interwoven with the other needs in the chapters. A brief description of each chapter follows.

Chapter 2 describes the standard Data General operating system environment. This includes a brief look at Data General hardware, a more detailed description of the operating system and a detailed study of EXEC's functionality, strengths and weaknesses. It should be noted that EXEC is proprietary software and as such is not supplied with any internal technical documentation. Much of the information on it had to be obtained by "hacking", this has meant that references to it are few and inadequate.

Chapter 3 discusses the internal architecture of MCP. Special attention is paid to memory management, the multi-tasking philosophy, code paths and interfacing between MCP and the rest of the operating system.

Chapter 4 addresses the terminal environment created by MCP. The main issues covered are security, operator control, networking and the basic program logic used.

Chapter 5 describes the PC environment. This includes a communications protocol, information transfer facilities, print-out transfer from PC-DOS[6] to queues on the Data General, terminal emulation, security, and operator control.

---

[6] PC-DOS refers to the Disk Operating System for Personal Computers written by Microsoft Corporation

Chapter 6 discusses the printer environment. The main issues are flexible and automatic printer programming, print queues and re-direction over the network, user and operator control of printers.

Chapter 7 concerns the batch processing environment. The philosophy and structure of the batch environment are discussed along with capacity, user and operator control.

Chapter 8 looks at the accounting system with special attention paid to the philosophy behind the generation of variable and peripheral costs.

Chapter 9 covers the system management toolbox needed to run MCP effectively. This includes a real time monitor, user security profile editor, system manager's interface to manipulate system files, log files, fail safe traps and emergency shutdown procedures.

The conclusion takes a retrospective look at MCP, quotes some performance characteristics, and proposes some improvements that could be made to the system.

Appendix A[7] contains the revision history of MCP, a summary of MCP commands and a table of the PMGR vectors.

Appendix B[8] contains program listings of the main MCP system.

Appendix C contains program listings for MCP's subordinate processes.

Appendix D contains program listings for the PC routines.

---

[7] Due to their bulk, the appendices are bound separately.

[8] MCP is the property of M S & A and the source listings are therefore restricted.

2. The standard DG operating system environment

In order to bring MCP into context, this chapter briefly describes the Data General eclipse MV series architecture and operating system. Particular attention is paid to EXEC which is the program that MCP replaces. The information presented in this chapter is based on AOS/VS[9] revision 7.5.

2.1 The MV system architecture

The following discussion is based on the Architecture Manual [ECL86/1] normally issued to Data General computer engineers, and to a lesser extent the Principles of Operation Manual [ECL86/2]. Data General offers a range of 32 bit processors from the MV 1400 (1 MIPS) to the MV 20000 model 2 (11 MIPS). While the architecture differs over the range, in general the CPU can be defined as a group of microcode controlled pipelined subsystems. These subsystems are:

The instruction processor (IP) which fetches and decodes instructions thus providing the start address and data for the relevant microcode routines.

The microsequencer determines the address of the next microinstruction and passes it on to the control store thus building a pipeline of microinstructions.

The address translation unit (ATU) translates a logical program address (consisting of segment, first level page table offset, second level page table offset and offset in page) into a physical address (consisting of a page number and offset in page). This address will now be sent to the data cache or the instruction processor in the case of a program branch.

The address generator (AG) converts displacement addresses generated by the instruction processor into logical addresses which are then passed to the ATU for conversion into physical addresses. The physical address is then used when the instruction moves into the execute pipeline stage.

---

[9] AOS/VS stands for Advanced Operating System / Virtual Storage and is the standard operating system supplied by Data General for the Eclipse MV series computers.

The arithmetic logic unit (ALU) performs arithmetical, logical and rotational functions.

The floating point unit (FPU), found on the faster machines only, is a separate processor that provides floating point instructions along with integer multiply and divide and intrinsic functions (trig, log, exponent etc). It receives the microcode routine start address and data.

In addition to the above, subsystems exist for memory and I/O management. The memory system can be summarised as follows:

The ATU in addition to converting logical to physical addresses also provides hardware based memory security. The 4 Gb of logical memory made possible by a 32 bit system is divided into 8 segments of 512 Mb each. Each segment has an associated ring numbered from 0 to 7. Programs in inner rings can access data in outer rings while programs in outer rings can call routines in inner rings through predefined ring gates. The protection mechanism is illustrated by the following table :

OPERATION	INWARD	OUTWARD
LOAD/STORE	NO	YES
CALL	YES	NO
RETURN	NO	YES

The memory control unit (MCU) receives the address from the ATU and, in the case of a write, produces 7 check bits that are stored with the data. In the case of a read, checking is performed and error correction applied for single bit errors or an error trap generated for multibit errors. The MCU also maintains a 64K x 1 bit page modification RAM. Each bit corresponds to a page of memory and is set when the corresponding page is modified. This table is logically part of segment 0 and is used by the operating system for memory paging.

The I/O system is driven by means of one or more I/O controllers (IOC). The IOC can be accessed by program driven I/O (PIO) or direct memory access (DMA). Two busses connect the IOC to the device controllers, namely the data channel (DCH) for slow and medium speed devices, and the burst multiplexer channel (BMC) for

fast devices. Some of the more common device controllers are:

Disk and tape controllers make use of the BMC and incorporate a 32 bit MicroEclipse processor. The exact architecture is dependant on the type of device being supported.

The intelligent LAN[10] controller (ILC) uses the data channel bus, incorporates a 16 bit Eclipse processor and allows connection to an Ethernet 802.3 LAN. The entire X25 connection layer can be down-loaded to this board.

The intelligent asyncronous controller (IAC) uses the data channel bus, incorporates a 16 bit Eclipse processor and provides 8 modem control ports or 16 normal asyncronous ports for connection to peripherals. This implies that, provided standard I/O is used to peripherals, the overhead is taken off the main CPU.

2.2 The AOS/VS operating system

Further information on the AOS/VS architecture can be found in the AOS/VS Internals Manual [AOS86/1] given to students (Software Engineers) on the S140 internals course.

In the AOS/VS context, tasks are scheduled by means of task control blocks (TCB) which consist of a state save area, links to establish a queue, and a status area. The system scheduler decides which task receives the CPU (system tasks always have priority), that task's state is restored, and execution of the task continues.

[10] Local Area Network

Figure 2.1 - A typical operating system

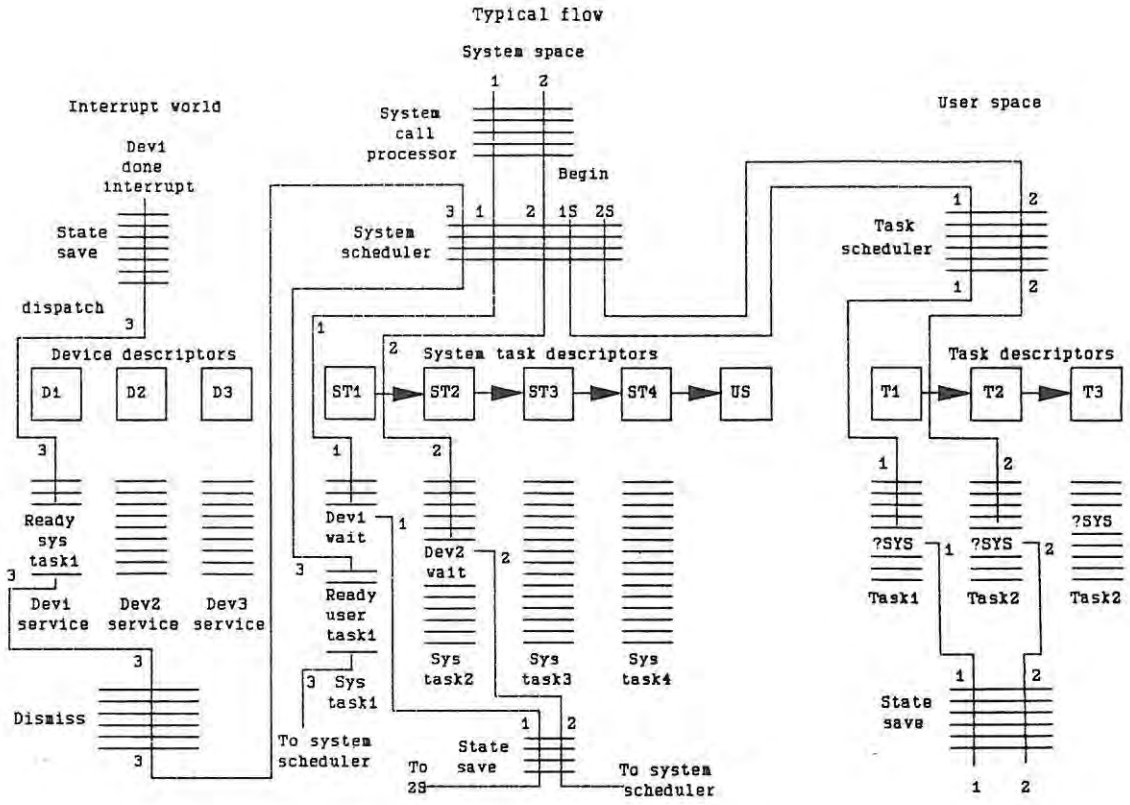


Figure 2.1 can be described as follows. A user task will be given the CPU by the system scheduler (see 1S and 2S in the diagram). When the task requires system resources it will issue a "?SYS" system request. The task will be suspended and control passed to the system call processor (1 and 2 in diagram) which will create a system task control block (CB) for the system code path to be executed. When the scheduler allocates the CPU to this new system task, execution of the code path begins. Many of the code paths will result in a request to an I/O device in which case an entry will be made in the device control table (DCT) and control passed back to the scheduler. When the "done" interrupt is received (3 in diagram) from the device the appropriate device driver is executed which, on completion, re-enables the system task which, in turn, re-enables the user task.

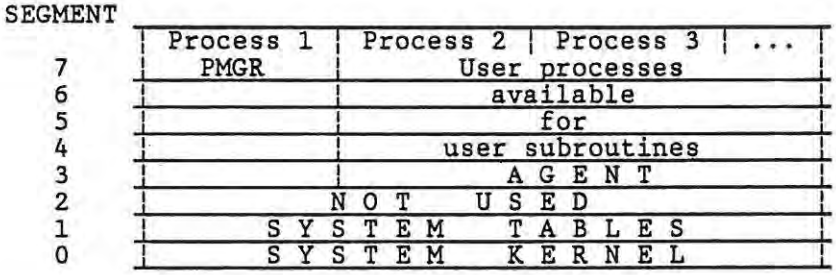
AOS/VS differs in a few ways from the picture painted above.

Firstly, each user process has a process table, and the system scheduler only schedules system task CBs and processes. User TCBs are scheduled by a task scheduler within the process. The system scheduling algorithm is heuristic and prioritises processes based on their behavioral patterns. I/O bound processes are given a higher priority than CPU intensive processes.

Secondly, system requests are not passed directly to the system call processor but to the AGENT which attempts to guarantee system integrity by extensive checking of the call parameters. The AGENT can divert the call to an independent process such as add-on software like XODIAC[11], the peripheral manager (PMGR) for asyncronous I/O or EXEC for printer management.

Thirdly, I/O to asyncronous devices such as terminals and printers is not done by the operating system kernel but by an additional process called the peripheral manager (PMGR). This program will be discussed in more detail in the next chapter as it forms the major interface between MCP and the operating system.

AOS/VS is spread over the first 4 rings of memory in the following manner:



From this table it can be seen that each process contains the operating system in its logical address space but that the operating system is protected by the ring security policy. This arrangement allows for efficient communication between the operating system and the user process.

A brief description of the contents of the system segments follow:

[11] as described in the introduction

2.2.1 Segment 0 - System Kernel

The kernel contains the interrupt world, system tasks and the most important data tables (see figure 2.1). The interrupt world includes the I/O drivers, interrupt handlers, device control tables (DCTs) and machine state save area.

The more important system tasks include memory management, the file system, the system call processor and the system scheduler.

The data tables held in segment 0 are the system task control blocks and the user process tables. Both of these are used by the system scheduler.

2.2.2 Segment 1 - System data tables

This segment holds all the remaining tables used by the kernel. They include the channel control blocks, logical memory page tables and the process table extenders.

2.2.3 Segment 3 - The Agent

PMGR code is also contained in this segment but will be handled separately in 2.5 below.

The agent performs 8 major functions, namely;

a) Dispatching of all system calls (all system requests from user programs come to the agent).

b) Provides an environment for application oriented system calls within a top-down structured operating system.

c) Validation of user supplied parameters.

d) Extended operating system support.

e) System call deflection for the resource manager agent (part of the XODIAC network system discussed in the introduction).

f) System call deflection for EXEC.

g) System call deflection for GSMGR (this is the global synchronous manager which supports synchronous devices just as PMGR supports asyncronous devices).

h) System call deflection for PMGR.

Note that the last 4 functions support add-ons to the operating system. Because all calls pass through the agent an integrated view of the system can be given to the user program. In other words, the same system call can be used to write data to a disk file (kernel), a console (PMGR), or a file on a remote machine (XODIAC-RMA).

2.3 Structure of add-on processes to AOS/VS

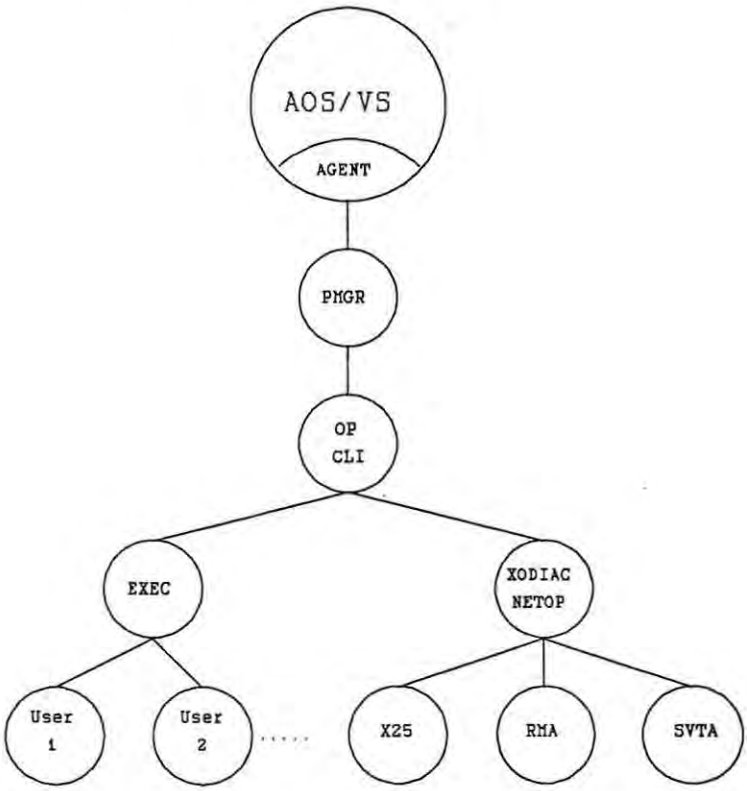
So far, the kernel and agent have been briefly introduced. The remaining parts of the operating system, namely; PMGR, GSMGR, EXEC and XODIAC have been briefly mentioned as add-ons. They exist on the system as user processes and, with the exception of PMGR (which is automatically started by AOS/VS), are started by the operator as part of the system startup procedure. There is an additional process called the command line interpreter (CLI) that provides a similar function to COMMAND.COM in the PC-DOS environment [DOS84] in that it allows a user to access the system functions via a command syntax.

When AOS/VS is loaded via the system bootstrap loader, it automatically starts PMGR and a CLI process for the master console. This is all that is needed for the system to function as a single user machine. In order to allow multiple users to use the machine the operator will start EXEC and instruct it to enable consoles. A user may then log on at one of these consoles by entering a username and password. EXEC will then start a process (normally the CLI) for the user. EXEC also performs other functions such as printer spooling and batch processing which will be discussed later.

In addition to EXEC, if local area networking is to be used, the operator will start XODIAC. XODIAC consists of a number of processes, namely; NETOP (the network supervisor), X25 (the X25 protocol controller), RMA (the resource manager agent) for file transfer over the network, SVTA (the server virtual terminal agent) for terminal facilities over the network and various data base servers such as RQL (the remote SQL agent) [MAN86]. The process structure is shown in figure 2.2

These processes interact with each other and with the user processes via three communications methods provided by AOS/VS. These are described in the next chapter.

Figure 2.2 - The AOS/VS process structure



2.4 AOS/VS inter process communications support

In order to insert a new program into the operating system suite it is necessary to look at the methods of passing data between processes supported by AOS/VS. These are covered in more detail in the AOS/VS system concepts manual [AOS86/2]. Three methods exist:

2.4.1 Shared memory

As AOS/VS is a virtual memory system, memory pages must be swapped in and out from disk. For a normal process, code is loaded into memory from the program file and unshared memory is swapped in and out from a unique temporary file created in a special directory called :PAGE by AOS/VS when the process is started.

A process can share part of its memory with another process by instructing AOS/VS to swap that area of memory to a different file. When another process instructs AOS/VS to swap part of its memory to the same file then AOS/VS modifies the logical page tables of both processes to point to the same memory.

Note that while this method is very efficient it does not cater for synchronization between processes. Update conflict is also a problem that has to be catered for programmatically, via page locking or some other mechanism.

2.4.2 Connection management

Under this method of communication a process declares itself to AOS/VS as a server. Another process declares itself as a customer. Once this connection is established, the server can move data from a customer data area to its data area via the ?MBFC (move bytes from customer) system call. The server can move data from its data area to the customers data area via the ?MBTC (move bytes to customer system call.

Note that this method is less efficient as AOS/VS has to move the data from one area of memory to another. As with the shared memory method there is no synchronization between processes. The server has to, via some other means (normally IPC), find out the memory address of the customer's data area. The customer has no control over the communications process.

2.4.3 Inter Process Communications (IPC) ports

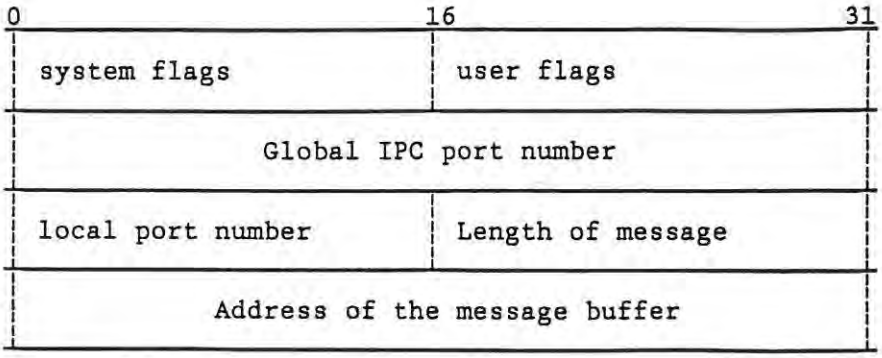
This is the primary communications method used to communicate with PMGR and is therefore discussed in more detail.

In a paper entitled "Message Passing Between Sequential Processes: the reply primitive and the Administrator Concept" [GEN81], W Gentleman discusses the Thoth communication primitives. This is the communications philosophy used in IPC communications. There are four main issues: Do processes block during communications or not, what addressing mechanism is used, what is a message and in what format is it provided to or by the system, and finally how are communications failures handled or avoided. The functions for the three Thoth message passing primitives are as follows:

```
destination_id = .send(msg,id)
requestor_id = .receive(msg,id)
sender_id = .reply(msg,id)
```

If "id" is omitted from the receive then messages can be received from any process. The DG implementation of these principles is as follows.

Firstly, one process creates a file with a file type of IPC. A second process issues a ?ILKUP system call to look up the port number of this IPC and hence derive the address (or in DG terms the global port number) of the receiving process. Hereafter the sending process can send messages to the port number via a ?ISEND system call which corresponds to the .send function. The first process receives the messages via a ?IREC system call which corresponds to the .receive function. A message can be sent and a reply obtained in one call via the ?IS.R system call which corresponds to the .reply function. Parameters are passed via a header which contains the following information:



The system flags are used to control process synchronization. A sending process can be suspended pending receipt of the message or continue processing when the message is sent. The receiving process can be suspended pending an incoming message or continue processing if no messages are available. If the sender does not suspend and the receiver is not suspended on an ?IREC at the time the message is sent then the message is spooled for the receiver. Not only is synchronization possible but the processes need no information about each other apart from the global port number to communicate.

User flags are passed unchanged to the user process and can be used to pass additional information to the receiving process.

The local port number allows only specific messages from those spooled to the receiver to be received. If a local port number of 0 is specified then all messages from all processes are received. This corresponds to omitting the id from the .receive function.

The message length indicates the size of the message. If this is set to 0 then only the header is sent. This is a fast method of communication as the header is kept in memory and thus no disk I/O is involved.

IPC communications posess one major disadvantage; all spooled messages are written to the IPC file on disk. This means that IPCs are a slow and inefficient method of communication. Communication to PMGR is done via IPC and this means that all data read or written to a console results in disk I/O. This is one of the most serious efficiency problems in the Data General

operating system.

2.5 The peripheral manager (PMGR)

As PMGR forms the major interface to the system for both MCP and EXEC, more detail is included. Further information on PMGR internals can be found in the AOS/VS Internals manual [AOS86/1].

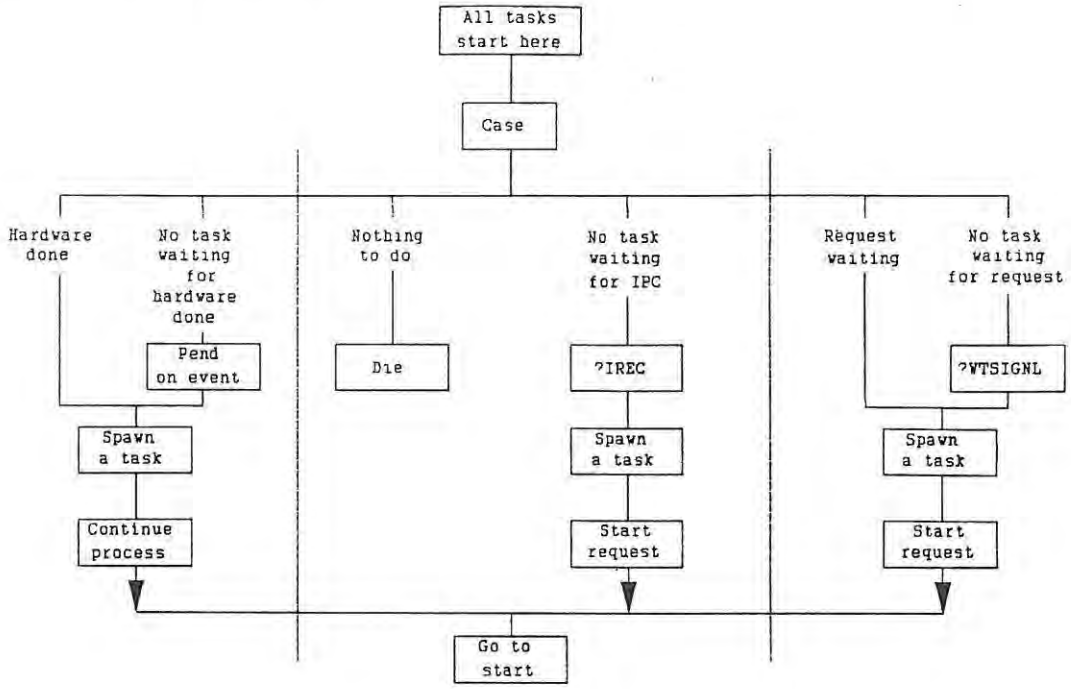
During initialization PMGR determines the number of peripherals on the system. In a special directory (:PER) an IPC file is created for each peripheral. The port numbers of these IPC files form part of a shadow port table (a more detailed discussion on the IPC ports created by PMGR is presented in section 3.1). The IACs are then initialized and the initial PMGR task is created.

In the idle state, three suspended tasks will exist inside PMGR, one to handle the hardware interface to the IACs, one to receive messages from the IPC ports and one to detect signals (used for synchronization) from user processes (see figure 2.3). As soon as an event occurs to activate one of the tasks, the task will spawn another task to service the next event and then cater for the current event itself. This means that the number of tasks active at a point in time will be determined by the number of hardware interrupts, IPC messages and signals being processed.

Note that PMGR has its own multi tasking environment and is thus not constrained as to the number of tasks active.

A user process interfaces with PMGR via the AGENT which transfers the data to/from PMGR via the console IPC ports. Synchronization is achieved via signals so that the user process continues processing only once the I/O is complete. EXEC interfaces direct to PMGR via the console IPC ports only. The PMGR interface is discussed in more detail in chapter 3.

Figure 2.3 - The global PMGR flow chart



2.6 AOS/VS security environment

As security is a key area in MCP, a brief discussion on the security base provided by AOS/VS is required. Further information can be found in the AOS/VS system concepts manual [AOS86/2].

2.6.1 Disk security

Each directory and file on disk has associated with it an access control list (ACL). This list consists of a series of usernames and access specifications. The username can be explicit or a template. The user can be granted one or more of the following privileges: Owner, Append, Write, Read and Execute. When a new file is created it is given the default ACL of the creating process.

2.6.2 Process security

As discussed in section 2.3 processes are organized in a tree structure. A father process can pass on any privileges that it possesses to its son. It can also terminate its sons. If a father process terminates for any reason then all the sons will be terminated by AOS/VS. For each process AOS/VS maintains a 16 bit privilege word in the process table. Each bit corresponds to a privilege. A process may not have a privilege unless its father has that privilege. The privileges are as follows:

Unlimited sons	This process can create an unlimited number of sons.
Change program type	Allows a 16 bit program to start a 32 bit program and visa versa.
Proc without block	The process may remain unblocked while its sons execute.
Set working set limit	The process may define its sons working set limit.
Change priority	The process may change its priority or create a son with a higher priority than itself.
Change type	The process may change its type or create a son with a different type. Process types include Resident, Pre-emptable and Swapable.
Change user name	This process can create sons with a different user name than its own.
Access devices	This process can perform direct I/O to user devices.

Use IPCs	This process can issue IPC communication system calls.
Superuser	This process is a superuser. This means that it can bypass the disk security discussed in 2.6.1.
Superprocess	This process is a superprocess. It can bypass the process security discussed in this section.
Peripheral process	This process can access peripherals via PMGR.

2.7 EXEC

As stated in the introduction EXEC provides the multi user environment. Since it is a proprietry product, no technical details on EXEC have been published, and so this discussion will be limited to its functionality.

Further information on EXEC commands and functionality is contained in the "How to Generate and Run AOS/VS" manual [HOW86].

EXEC meets three basic needs, namely the identification and introduction of users to the system, the control of printers, and batch processing. For each of these environments there is a user view and an operator/system manager view.

2.7.1 The user/terminal environment

2.7.1.1 User's view :

When a console is enabled by the operator a message is displayed on the console asking the user to enter his username and password. Once this information is entered EXEC fetches his user profile (explained in the system manager's view below). The contents of a special file called LOGON.MESSAGE is then displayed on his console. This allows the

operator to inform users of important news. After this the program specified in the user's profile is started with the security privileges specified by the profile.

2.7.1.2 System manager's view:

When introducing a new user to the system the system manager creates a profile and a disk directory for him. The profile contains information on privileges, priority and environment to be granted to this user along with the program to be run when the user logs onto the system.

The system manager can instruct EXEC to treat an async port as a console with the ENABLE command. DISABLE will release the port and TERMINATE will force the user currently using that port off the system. These commands take either a console number or "/ALL" as an argument signifying either one or all async ports. This is very limiting as, in most cases, a group of consoles needs to be referenced. This problem is solved in part by using "macros" (similar to PC-DOS batch files [DOS84]) containing all the consoles in a group. On a large system one console may often be part of more than one group (eg a system, a network node, a database, a department), making maintenance difficult when console numbers change.

When a user logs off the terminal, an entry is made in the system log file of his connect time.

2.7.1.3 Deficiencies in EXEC user/terminal environment:

a) Insufficient levels of security (discussed in chapter 3).

b) All consoles are treated as dumb devices. No support for PCs.

c) No facility to create groups of terminals.

d) No accounting facilities (except connect time entry in the system log).

e) No support for networked machines.

f) Only one profile per user. Thus it is not possible to create two passwords for one user name with different privileges.

2.7.2 The printer environment

2.7.2.1 User's view :

A user outputs data by sending it to a print queue. Each printer has at least one queue associated to it. Data can be placed in the queue in two ways:

Firstly files can be placed in the queue by means of the QPRINT command. Various options can be set such as priority, number of copies and "delete after printing".

Secondly the print queue can be opened as a file allowing data to be written to it record by record from a program. The data is printed once the I/O channel to the queue is closed. No documentation could be found to confirm the mechanism whereby this is achieved but from observation and experimentation it would appear that AGENT intercepts I/O against a queue and re-directs it to EXEC via an IPC port (discussed in section 2.4.3).

If the printer is started in TEXT mode then EXEC strips all control characters from the data, formats it into pages and inserts form feeds to skip the perforation. Page formatting information is obtained from a "forms" file created by the system manager. The printer is assumed to be a dumb device and no control characters can be sent to exploit features such as condensed print.

If the printer is started in BINARY mode then all the data will be sent to the printer unchanged, but no print formatting is done and programs must use binary I/O to the queue.

Entries in a queue can be removed by means of QCANCEL or held for later printing via QHOLD.

2.7.2.2 System manager's view:

A new queue is introduced to EXEC by means of the CREATE and OPEN commands and removed by CLOSE and DELETE. The entries in a queue can be removed by means of the PURGE command.

Once a queue has been created one or more printer ports can be assigned to service it via the START command. A separate command must be given for each printer which is laborious in machines with a large number of print devices. EXEC starts a new process for each printer port called XLPT. Due to the very low resource requirement to drive a printer a separate process for each printer seems to be a waste of memory and scheduling overhead.

Attributes can now be assigned for the printer. These include print headers, trailers, default forms, lines per page, characters per line, binary mode and "even pagination".

Printing is started by the CONTINUE command. Printing can be interrupted at any stage to alter the attributes via the PAUSE command. A printer port is released via the STOP command, resulting in termination of the associated XLPT process.

2.7.2.3 Deficiencies in EXEC printer environment:

a) No support is provided for printer features such as letter quality, italics, variable character and line spacing. Thus the user must generate these codes. Furthermore, control codes can only be output to a printer enabled in binary mode. Binary mode

printing inhibits page formatting. This paradox puts printer features beyond the reach of the average user.

b) Page formatting is achieved via software rather than using the printer's features.

c) An additional program is started by EXEC for each printer thus increasing system overhead.

d) Forms files are used to specify page formats for printing. Changing of forms on a printer requires operator intervention even though printers are situated locally for the user and remotely for the operator. On systems with high printer counts this represents an unnecessary burden on both operator and user.

e) No facility exists for automatic re-direction of printing across the network. Both the remote machine network name and queue name have to be supplied by the user.

2.7.3 The batch processing environment

2.7.3.1 User's view :

A special queue called BATCH_INPUT is used for batch jobs. Jobs are submitted to this queue via the QBATCH command. Batch jobs usually consist of macros which specify the work to be done. The command accepts switches to specify the time the job is to run, its priority, the queue to which output is to be sent etc. The contents of the queue can be viewed with the QDISPLAY command. Jobs can be removed by QCANCEL and held with QHOLD and later released for processing with QUNHOLD.

2.7.3.2 System manager's view:

The BATCH_INPUT queue is created in the same way as the print queues. Entries in the queue are serviced by means of streams. Each stream can

run one job at a time thus the number of jobs that can run concurrently from the batch queue are determined by the number of streams that are created. Streams are created by the CREATE/STREAMS=n command.

As with printers, the batch queue can be paused and continued and jobs can be held for later processing. Individual jobs can be cancelled or the entire queue purged. A currently running job can be flushed or restarted. Streams can be set to process jobs of a certain priority via the QPRIORITY command. This is of limited value however because the user can set his own queue priority when submitting the job.

2.7.3.3 Deficiencies in the EXEC batch processing environment:

a) On a real time oriented machine like the DG the batch environment is normally used to run CPU intensive jobs in back ground and thus free the terminal for interactive use. To guarantee good response for the real time environment, batch programs must be tightly controlled. In the EXEC environment queue priority is the only way of differentiating between jobs and as this is set by the user, control is compromised.

b) Some batch programs run by a user are more important than others. It is not possible to grant different programs run by one user different scheduling priorities.

c) Only one batch queue exists. Access to a queue can be controlled via the ACCESS command but as there is only one queue this can only be used to permit or prevent use of batch.

d) Security privileges are granted as per the user's profile thus it is not possible to differentiate between real time and batch privileges.

2.7.4 The Tape control environment

EXEC allows control to be effected over the tape environment by allowing users to request the tape resource. The operator is then prompted to mount a tape and confirmation given to the user when the tape is ready.

The tape environment will not be discussed in more detail for the following reasons :

a) Due to the implementation of tape I/O in AOS/VS, data transfer to and from tape drives is too slow to be used for any function other than backups.

b) A better tape I/O utility (DBRDUMP) for backups already exists.

c) The user can obtain exclusive use of a tape drive by using the CLI command ASSIGN which makes the EXEC support redundant.

d) EXEC notifies the operator of tape requests via the master console. This assumes that the operator is monitoring the console which in the AOS/VS real time environment is unlikely. A simple phone call to the operator is more effective.

2.8 The Michigan terminal system

In order to compare the functionality of EXEC with other environments outside the DG architecture, The Michigan terminal system (MTS) running on IBM 370 architecture is evaluated. The information has been taken from two papers, namely "The Michigan Terminal System" by D Boettner and M Alexander [BOE75] and "A penetration analysis of the michigan terminal system" by B Hebbard, P Grosso, T Baldridge, C Chan, D Fishman, P Goshgarian, T Hilton, J Hoshen, K Hoult, G Huntley, M Stolarchuk and L Warner [HEB80].

The functionality offered by the IBM operating systems is not as extensive as that of AOS/VS. This means that MTS has to perform many of the functions provided by the operating system in the DG environment. Some of the functions that are

provided by AOS/VS that MTS has to provide on the IBM include; resource allocation and scheduling, task initiation and termination, interrupt handling, inter-task communication and timer services. Because of the very different environment and the need for more low level support on IBM equipment, MTS varies considerably from EXEC. The functionality offered to users is discussed in the following sections.

2.8.1 Terminal support

Terminal users log onto the system by entering their signon ID and password. File security is enforced by "attaching" files to signon IDs. This is very limiting and onerous on the system manager when compared to the DG system of ACLs. Once logged on the user may enter commands. This environment is similar to that provided by EXEC and carries the same deficiencies in that security depends entirely on password and no support is provided for PCs.

2.8.2 Printer support

Although very little data is given in the papers on printers under MTS it would appear that printer output is sent to output streams which correspond to EXEC's queues via the $LIST command. As with the EXEC environment no use is made of the printer's intelligence and thus the same deficiencies apply.

2.8.3 Batch support

Batch command files are sent for batch processing in one batch queue. This is similar to EXEC's batch environment.

2.8.4 System accounting

MTS provides for the collection of information on system usage. This functionality is provided by AOS/VS in the DG environment but like EXEC no use is made of the information.

In conclusion, the environment offered under MTS is similar to EXEC but not as easy to use. Most of the deficiencies mentioned under the discussion on EXEC apply equally to MTS. Due to the similarity between MTS and EXEC, MTS will not be

used in the comparison with MCP in chapters 4 to 9.

3. The internal architecture of MCP

In section 2.7 the functionality of EXEC and some of its deficiencies were discussed. MCP has been designed to address these deficiencies and to offer increased functionality for peripheral support. In addition to the terminal, printer and batch environments offered by EXEC, MCP also supports PC workstation, accounting and system management support. These six environments are discussed in detail in chapters 4 through 9.

This chapter details the architecture of MCP and the manner in which a multi-user environment is achieved. A number of philosophies were attempted before arriving at the structure detailed in this chapter. Appendix A contains a summary of the revisions made to MCP in arriving at the current structure. Two main factors affected the design of the MCP system.

Firstly a bug in the FORTRAN 77 task control environment causes the entire process to be terminated when a task is terminated via the TQKILL function. As the whole of MCP is written in FORTRAN 77 with the exception of the "panic trap" and ring 6 gate declarations which are written in assembler, a structure similar to that described in section 2.5 on PMGR is not possible.

Secondly AOS/VS only allows 32 tasks per process. When standard I/O is performed to a terminal via AOS/VS the calling task is suspended. This implies that a maximum of 32 consoles can be controlled by a process using standard I/O. The obvious solution to this problem is to interface direct to PMGR in a similar manner to the AGENT and EXEC. The following discussion details the PMGR interface upon which MCP is based.

3.1 The PMGR interface

As PMGR is an integral part of the operating system, no documentation on the interface is available. The following information was gleaned by writing programs that created IPC ports that looked like console IPC ports and then monitoring the messages that were passed to them when they were enabled as terminals under EXEC.

3.1.1 Console IPC global ports created by PMGR

When the operating system is loaded, PMGR references the system configuration tables to obtain information on all the asyncronous ports that are present. An IPC file is created in a special directory called PER for each port. Three entries are created in the AOS/VS IPC table for each port, the first is used for device control, the second for read and the third for write. The first table entry for a port corresponds to the global port number of the IPC file in PER. For unknown reasons the entries in the IPC table do not correspond to the asyncronous port number but to the order in which the ports were specified in the system configuration tables. It is thus necessary for MCP to maintain a conversion table to convert between asyncronous port numbers and IPC global port numbers. The layout of the IPC table entries defined by PMGR is represented in figure 3.1.

Figure 3.1 - IPC ports created by PMGR

Async port	IPC file in PER	IPC global port numbers	
12	CON12	X+0	Control IPC port
		X+1	Read IPC port
		X+2	Write IPC port
13	CON13	X+3	Control IPC port
		X+4	Read IPC port
		X+5	Write IPC port
14	CON14	X+6	Control IPC port
		X+7	Read IPC port
		X+8	Write IPC port
		.	
		.	
		.	
		.	
		.	

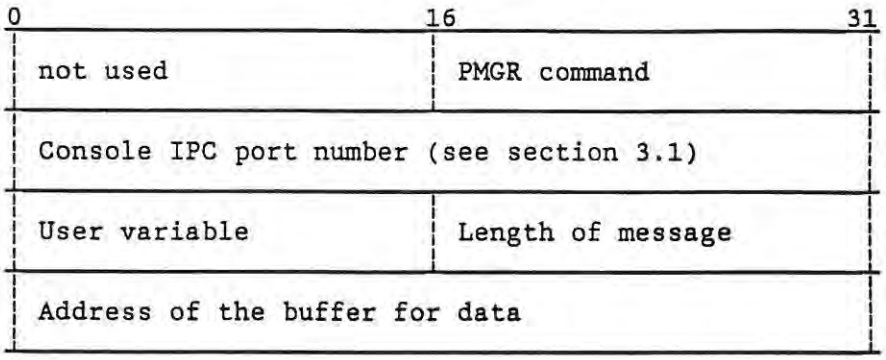
The value of X is found by performing a lookup of the IPC file in PER.

3.1.2 Flow of control between MCP and PMGR

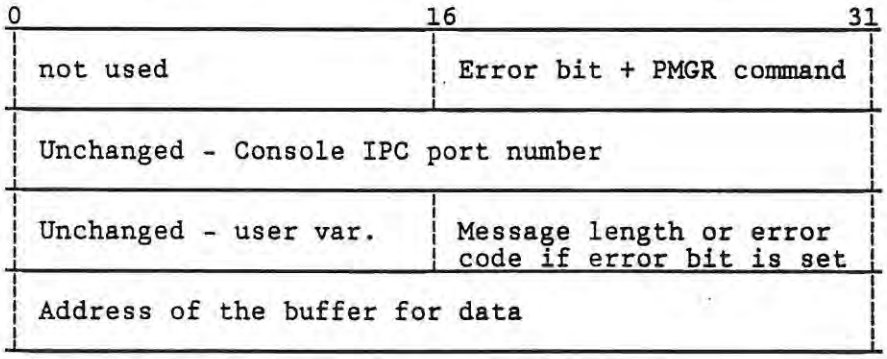
A program wishing to communicate with PMGR must first create an IPC file of its own to receive return messages from PMGR. It can then send an IPC header (and message for commands such as write) to the IPC port described in 3.1.1. Part of the header will be the command that PMGR is to execute. The program can then continue with some other task. When PMGR has completed the command it returns an IPC header to the program's IPC port. The header will contain the IPC port of the console, the command that was executed and an error code if applicable. For commands such as read, a message containing the data read will also be returned. The format of the IPC headers sent and received to PMGR by a program are shown in figure 3.2.

Figure 3.2 - IPC message headers for communicating with PMGR

IPC message header sent to PMGR



IPC message header received from PMGR on completion

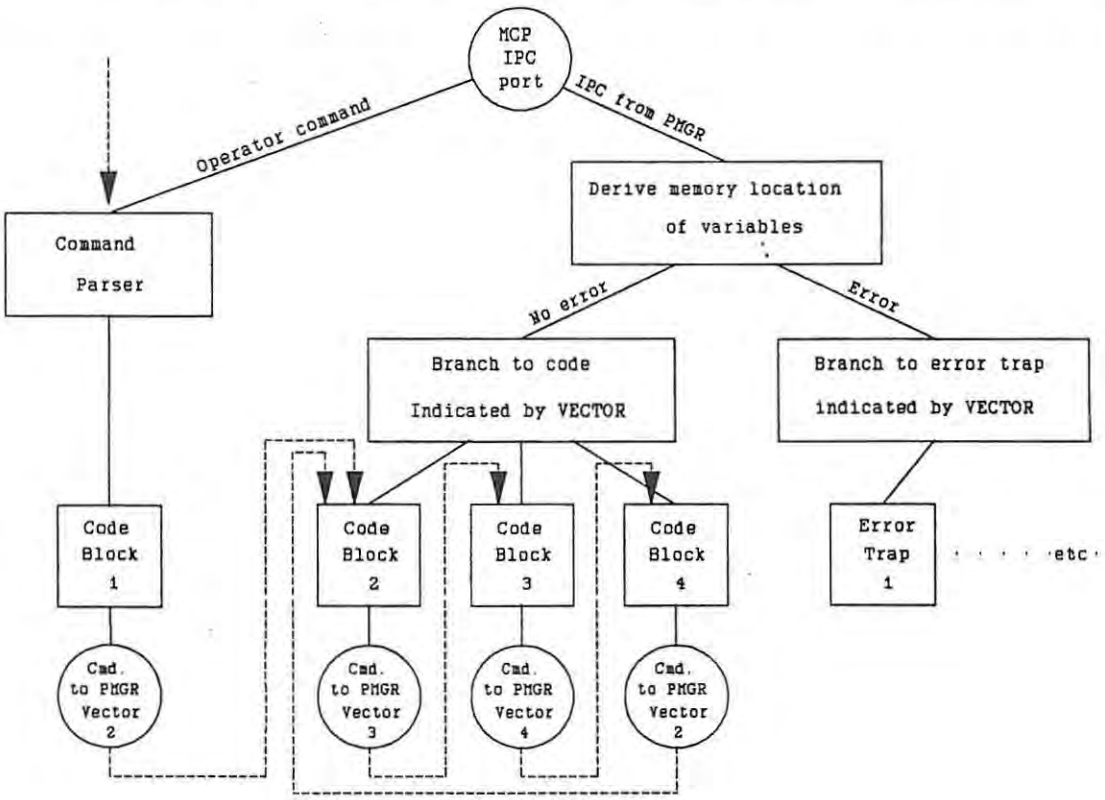


Note that the console IPC and the user variable are always returned unaltered to the program.

In MCP the console IPC port number is used as an index to an array row containing all the variables for that console. On sending an IPC header to PMGR, the user variable is used as a vector to indicate the section of code to be executed on return from PMGR. If an error occurred in PMGR then the vector is used to indicate which error trap to execute. Thus on receipt of an IPC from PMGR, MCP always knows what variables to use and what code to execute. As PMGR is working with slow asyncronous devices, MCP spends most of its time waiting for IPCs from PMGR.

An example of the flow of control as seen by MCP is graphically depicted in figure 3.3. In this example, the flow is initiated by the receipt of an operator command from the IPC. The dotted lines depict the logical program flow. The solid lines show the actual program flow.

Figure 3.3 - An example of the flow of control between MCP and PMGR



3.1.3 PMGR commands

The following is a brief list of the PMGR commands used by MCP.

Assign	Assign the console to this process.
Release	Release the console for other processes.
Open	Open the console for I/O.
Close	Opposite of open.
Get delimiters	Get the table of data sensitive delimiters.
Set delimiters	Set a new table of data sensitive delimiters.
Get characteristic	Get the current characteristics of this console. Characteristics include device type, baud rate, time outs, echo, lower case control, modem control etc.
Set characteristic	Change the characteristics of this console.
Read	Read data from the console.
Screen edit	Special form of read, allows standard DG screen editing to be performed while data is being input.
Write	Write data to the console.
Set time out	Set the time interval for time outs. Actual time out is switched on via the set characteristics command above.

3.2 The internal structure of MCP

MCP achieves a multi-user environment by using virtual tasks. For speed twelve AOS/VS tasks are used. MCP virtual tasks are made possible by memory management, IPCs and the PMGR interface described in 3.1.2. The following discussion introduces the basic principles used in MCP.

3.2.1 Memory organization

MCP has been developed in FORTRAN 77 because of the tasking features and operating system interface offered in the Data General implementation. Fortran however imposes certain limitations on the organisation of memory.

Since MCP needs to handle a number of consoles "simultaneously" separate variables need to be maintained for each console. In FORTRAN 77, the easiest way of achieving this is via arrays with each row corresponding to a console. The variables used for console management are grouped into a number of tables, namely; a terminal/PC table, a printer table and a profile table for security control. Because PMGR identifies consoles via the console IPC port number a mapping function is used in MCP to switch from row number (console number) to IPC port number and visa versa.

These tables are further grouped into a contiguous memory area via a named common block. This allows the data to be shared among all the tasks. It also allows the common block to be located in shared memory.

Use of shared memory has a number of advantages, namely;

a) Memory paging is done to a permanent file (MCPCONDAT) rather than the temporary area in the :PAGE directory. Whenever the program terminates, AOS/VS flushes the contents of shared memory back to disk. This means that if MCP terminates abnormally, an image of its variable contents is stored in the shared memory file which greatly simplifies debugging.

b) Authorized processes can use the shared memory to gain access to MCP's variables. Access to MCP's memory is used in the system management toolbox by programs such as the monitor.

c) When the program is started and the shared memory set up, the memory is initialized with the contents of the MCPCONDAT file. This means that the variables contain the same information as when MCP was last stopped. This cuts down the amount of memory initialisation required.

MCP print and batch queues also use shared memory for similar reasons to those discussed for terminals. Advantages include:

a) The variables assume the same value on startup as they had on termination thus preserving the queues.

b) There is no degradation in the response time when compared to a normal program as memory paging is just re-located to another area of disk.

c) No disk I/O is required to maintain the queue apart from normal paging done by AOS/VS.

d) Memory resident queues can be scanned and modified quickly.

As with the consoles, queue shared memory is divided into a number of tables, namely a print queue table, a batch queue table and a batch class table.

Apart from the tables in shared memory MCP makes use of a few work variables. These are not associated to particular consoles and are thus transitory. To avoid memory contention two conditions must be satisfied. A work variable may only be used by one task and a code path's use of a variable must be complete before that code path surrenders control of the task.

3.2.2 IPC usage

As discussed in section 2.4 incoming messages are spooled in an IPC file until they are fetched via the IREC system call. The main program loop of MCP is to receive a message, decode it, branch to the relevant code to process it and then loop back to receive the next message. There are three categories of IPC message handled by MCP, namely:

3.2.2.1 Process termination messages.

These are sent by AOS/VS when a son process of MCP terminates. Four types of process are of concern to MCP.

Firstly user processes started by MCP on receipt of a valid username and password from a terminal. Receipt of this termination message would result in the start of a logon sequence at this terminal.

Secondly batch processes started by the batch system. Receipt of this termination message would result in a reschedule of the batch queues to determine the next job to run.

Thirdly the XODIAC network SVTA[12] process linked to MCP via a server/customer relationship (see section 2.4). SVTA functions in a similar manner to PMGR for virtual consoles connected over the network. Notification of termination of this process would result in MCP dropping all enabled virtual terminals.

Finally the accounting system runs as a sub-process to MCP. This is discussed further in chapter 8.

3.2.2.2 Operator messages

These consist of commands from the operator, commands from a user, commands from MCP to itself and resource requests from an MCP process running on a remote machine in a network. All of these messages result in creation of a new code sequence or virtual task (discussed later in this chapter).

Operator commands concern the control of devices such as enabling consoles or starting printers. All the commands are discussed in detail in chapters 4 to 9.

---

[12] Server Virtual Terminal Agent

User commands are strictly controlled for security reasons and are limited to queue manipulation. They are also discussed in chapters 4 to 9.

Commands from MCP to itself or internal operator commands are used by secondary tasks to pass control back to the main task and also to create new virtual tasks.

Resource requests from remote MCPs are used to allow users of MCP on a remote machine to use resources on the local machine transparently.

3.2.2.3 PMGR/SVTA command completion messages

The PMGR interface has been discussed in section 3.1. Once the completion message is received, MCP branches to the relevant code block to continue processing for that console port.

3.2.3 Multi-tasking

MCP uses a limited number of AOS/VS tasks and a multitude of virtual tasks created in the MCP environment. Each of these are discussed in the following sections.

3.2.3.1 Virtual tasking environment

Each device under MCP is serviced by at least one virtual task. A virtual task is a logical code block that consists of a number of physical code blocks within various physical tasks. Dynamic linkage between physical code blocks is achieved via vectors.

The illusion of multi-tasking between virtual tasks is not achieved by task control blocks or separate CPU allocation but is the result of MCP frequently passing control of a console port to another process such as PMGR or a son process. Thus typically MCP will receive an IPC message related to a particular console port, it will branch to a small block of code which will result in another command to PMGR or

the starting of a son process. Once this code block is complete MCP will fetch an IPC message for another console port.

In summary, because these code blocks are small, the waiting time for an incoming IPC message for a particular console port is short thus creating the appearance of a separate task for each console port. Furthermore the vectors discussed in section 3.1 link the code blocks together into a logical program flow. The result of these two factors is a virtual task.

A virtual task is created by an operator command such as enable terminal or start printer, and is killed by an irrecoverable error, completion of the command or another operator command such as disable terminal or stop printer. It may be dormant for long periods of time such as when a process is started for a user at a terminal, but as soon as the process terminates and the termination IPC message is received it will be reactivated by the main physical task (see section 3.2.3.2).

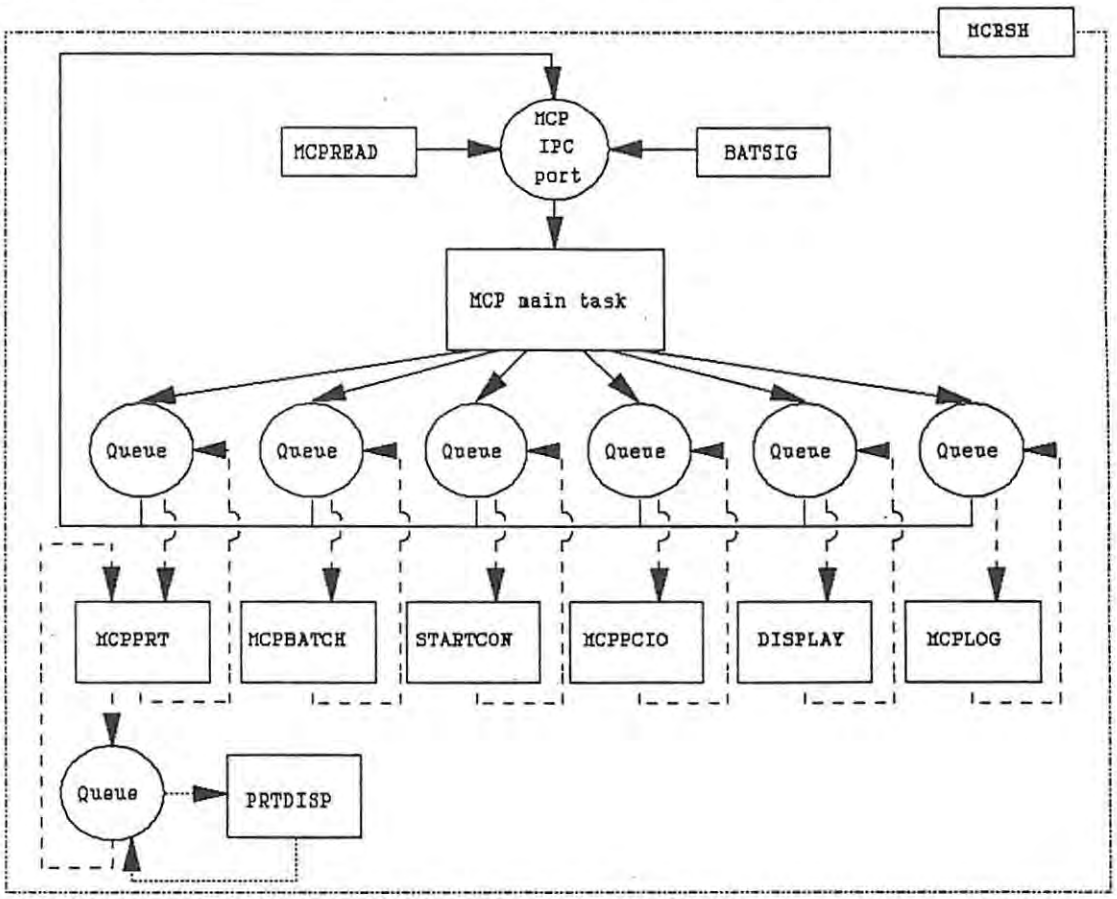
Generally, one virtual task exists for each console port serviced by MCP. This rule is broken in some instances such as when a PC is transferring a file to the DG. In this example one virtual task exists to handle the communications protocol to the PC and one to store the data on disk. In these cases the second virtual task is created by MCP sending itself an internal operator command. The second virtual task is killed when its function is complete.

3.2.3.2 AOS/VS tasking environment

From the above discussion on virtual tasks it is apparent that if a code block is too long or involves slow operations such as disk I/O then the waiting console IPC messages will not be serviced promptly and poor response will result. To solve this problem all slow code paths have been moved into separate tasks thus freeing the main task to process the next IPC message.

Figure 3.4 demonstrates the flow of control through the tasks making up the MCP system. The solid lines show the main task flow, the broken lines show the sub-task flow and the box surrounding the picture depicts the general role of MCRSH.

Figure 3.4 - The flow of control between MCP and its tasks



Control of a virtual task is passed from the main task to another task via a memory resident queue. The queue will contain the console number for access to the console tables, a vector to indicate the code block to be executed in the task and the relevant work variables not held in the console tables.

The task will then process the contents of its queue in a similar way to that in which the main task processes the IPC messages. In the

majority of cases, execution of a code block results in a PMGR command or the starting of a son process for a terminal user. In these cases control is passed back to MCP via the IPC port and hence the main task. If control does not leave MCP after a code block is executed then control is either passed to another task by means of its queue or back to the main task by means of an internal operator command.

Once a task has emptied its queue it suspends on an inter-task TQREC message receive. Thus in their idle state all MCPs subtasks are suspended on a TQREC and the main task is suspended on an IREC of the IPC port.

There are six sub-tasks of the type described above and five dedicated tasks to perform specialized functions. The tasks and a brief description of their functions follow. Further detail on the system is supplied in chapters 4 to 9.

i) Main task

This task has been discussed in detail in this chapter. It is the kernel of the system and delegates work to the tasks described below. A full discussion of its functionality is contained in chapters 4 to 9.

ii) STARTCON

This task handles logon processing for terminal users. It reads the user's profile from disk, performs security checking, displays the logon message on the terminal, performs connection to remote machines if applicable and starts the user's process. It also displays the disable message during terminal disable.

iii) MCPPRT

This handles the printer environment. It manages the print queues, programs printers, displays headers, outputs files to

the printers and executes user instructions such as number of copies. It passes all operator requests for printer information to the PRTDISP task.

iv) PRTDISP

This task displays information to operators on printer status and to users on print queue contents.

v) MCPPCIO

This task controls the PC environment. It transfers files between the PC and the DG file system, controls security, manages the PC to PC transfer queues, directs PC printing to DG print queues and provides networking facilities for PCs to other DG machines.

vi) MCPBATCH

This task manages the batch environment. It manages the batch queues, parses user and operator commands for the batch system, controls security, schedules batch jobs in the various classes and starts processes to execute the batch jobs.

vii) DISPLAY

This task displays MCP information to the operator. This includes terminal status, PC status and technical information to assist with debugging of problems with communications to devices.

The following tasks are the five special tasks mentioned in 3.2.3 b) above.

viii) MCPLOG

This task writes messages to the MCP log files on operator commands and possible security breaches.

ix) BATSIG

This task sends wake up messages to the batch system to initiate re-scheduling. This is required to start batch jobs that are submitted for running at a specific date and time.

x) MCPREAD

This task intercepts the EXEC print requests and re-directs them to MCP. This is required for handling print output from certain DG packages such as CEO[13].

xi) MCPNET

This task communicates with MCP on a network of remote machines specified by the system manager to obtain active print queues on each machine. These queues along with information on the host machine is stored in memory arrays for use by MCP during re-direction of printer commands.

xii) MCRSH

This task is used to control abnormal termination of MCP. If the MCP process panics[14] then if it were to terminate, all the son processes would also be terminated by AOS/VS. This is very

---

[13] Data Generals comprehensive electronic office package

[14] A process panic is a term used by Data General to denote an irrecoverable error. This type of error normally occurs when AOS/VS memory protection is violated and is imposed by AOS/VS forcing the program to branch to a page 0 memory error trap address. This is discussed in more detail in section 9.

inconvenient from the user's point of view. For this reason an assembler routine is used to trap the MCP process before it terminates. This task is used to free this trap once all the users have been notified and their jobs stopped.

4. The MCP terminal environment

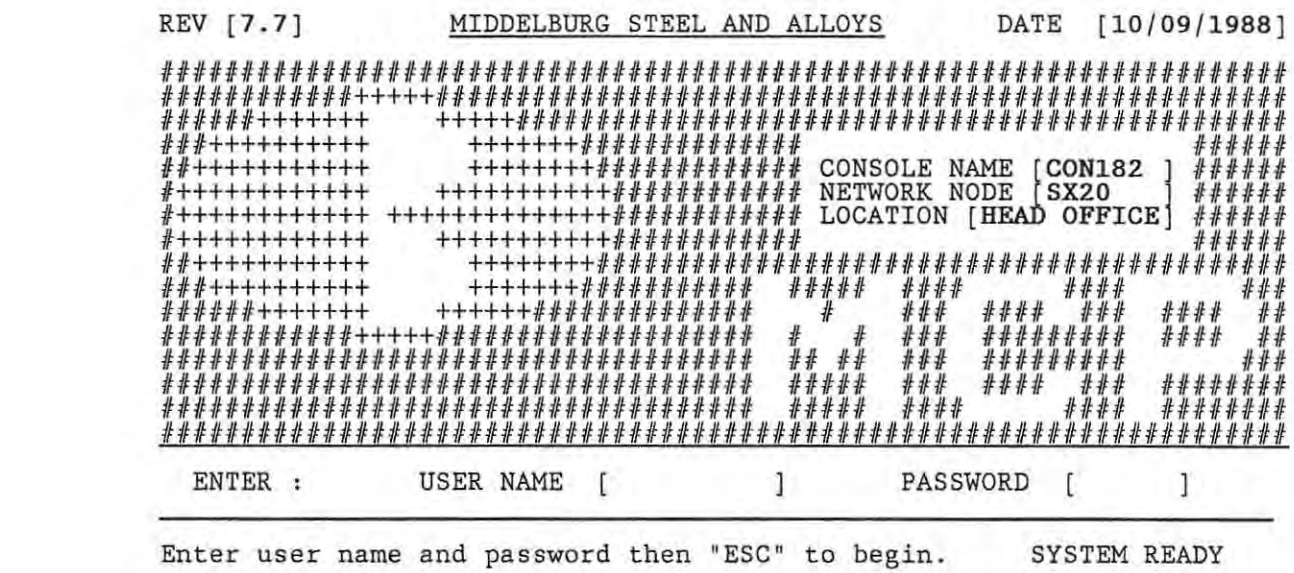
In section 2.7.1 EXEC terminal support was discussed. This chapter describes the support of terminals under MCP. The discussion is divided into three areas, namely; functionality, security and program structure.

4.1 Functionality of the MCP terminal environment

MCP attempts to address the deficiencies highlighted in the discussion on EXEC (see section 2.7). Unlike the EXEC environment where one profile file is created for each user name, MCP maintains an encrypted file of user names and passwords which refers to the profile file. This allows one user name with different passwords to reference different programs, have different security privileges and even access different machines. The security measures are discussed further in section 4.2.

4.1.1 User's view:

Figure 4.1 - The MCP logon screen



As with EXEC a logon screen is displayed with fields provided for user name and password. The logon screen is kept in a file. Modification of this file allows the system manager to tailor the screen to display the company's logo

or symbol. If no user name is entered within ten minutes then the screen is blanked to prevent unnecessary damage to the phosphor. Once the screen is blanked, pressing any key will result in it being re-displayed. The screen used by MS&A appears in figure 4.1.

When the user enters his user name and password and no security violations are found, the LOGON.MESSAGE file is displayed on the console. This allows the system manager to communicate important information to all the users. After this the program from the user's profile is started with the security access specified in the profile. Under the EXEC environment, no facilities exist to set a search list and default ACL for the user. The search list is used to specify directories other than the working directory to be searched for files and programs. The default ACL is used to assign ACLs to files created by the user. Both of these facilities are essential for most systems. Under EXEC a CLI program is used to set both the search list and default ACL and to start the user's program. This is inefficient as more processes are started than are needed. Under MCP, the search list and default ACL are contained in the profile file and are set up when the program is called. This obviates the necessity of starting a redundant CLI process.

Once the user's program is running, MCP ignores the console until a program termination message is received from AOS/VS. When this message is received the logon screen is displayed and the process repeated.

In their paper on distributed operating systems [TAN86], Tanenbaum and van Renesse discuss the differences between network and distributed operating systems. In summary, they define a networked system as one where the users are aware of which machine they are using whereas in a distributed system they are not. The AOS/VS-XODIAC-EXEC networking system supplied by Data General falls squarely in the networking definition. A user wishing to connect to a remote machine must first log onto his local machine and issue a connection call to the remote machine and then log onto the remote machine. MCP attempts to provide an environment as close as possible to a distributed system. In the user's profile a network node is stored. If this is not the node of the local machine then MCP will connect the user to the

remote machine automatically. This means that no matter what machine the console is connected to the user will be connected to the correct machine to run his program.

This is achieved as follows. Firstly the local MCP sends an IPC message to MCP on the remote machine specified in the user's profile. The remote MCP then locates and releases a virtual console for use by the user. The remote MCP then starts the program on this console with a unique process name and notifies the local MCP. The local MCP then starts a process on the local machine to connect the user to the process on the remote machine.

4.1.2 System managers view :

A profile is created for each user name password combination rather than for user name only as in the case of EXEC. As has been mentioned this gives far more flexibility and reduces the number of different user names that have to be created. The MCP profile offers considerably more security options than the EXEC equivalent which will be covered in more detail in the section 4.2.

An additional file called MCPLOCATIONS is maintained by the system manager. This file contains information by console on terminal location, network node, queue, name and headers. This information is used for terminals, PCs, and printers, and facilitates a more powerful command syntax than EXEC.

The maintenance of all MCP's files is performed via the system managers interface utility discussed in chapter 9.

As with EXEC, asyncronous ports are enabled as terminals with the ENABLE command, released with the DISABLE command and users forced off the system with the TERMINATE command. Additional syntax has however been added to the MCP commands. Possible variations of the commands are as follows:

ENABLE @CON128 - Enable port 128 as a terminal

ENABLE ALL - Enable all unassigned ports as terminals

ENABLE USERS ROB - Enable all ports with a location in the MCPLOCATIONS file starting with the letters ROB

ENABLE USERS - Enable all ports with a valid location in the MCPLOCATIONS file.

The addition of the USERS phrase to the command allows groups of consoles to be manipulated. This circumvents the problems with macros discussed under EXEC in section 2.7.1. During machine startup the ENABLE USERS command will enable all terminals on the system with one command thus greatly speeding up the process. In addition the ENABLE USERS command recognizes PCs and printers from the MCPLOCATIONS file and starts them as well. This means that with one command all the peripherals on the system can be activated. This has reduced the time to activate peripherals from 15 minutes under EXEC to less than one minute.

In addition to the commands described above MCP offers a number of extra commands, namely;

CLEAR @CONnn - The Data General machines use soft flow control (control+S to stop transmission, control+Q to resume). If a user types control+S or a printer gets out of sync then transmission stops. Under EXEC the console has to be disabled or the printer stopped and re-activated to clear a control+S. This command has the same effect.

RELEASE @CONnn - This command forces MCP to drop the console port via PMGR. It is used mainly with ports using modem control signals when the modem is not active and "clear to send" is disabled.

LOCATION @CONnn - This command displays the location of a console.

DEBUG @CONnn - This command provides information on MCP flags and status for diagnosing the problem with a console port.

4.2 Terminal security offered by MCP

Apart from the AOS/VS security discussed in section 2.6 EXEC only offers username and password. The password is stored in the user's profile file. This means that a person who gains access to the master console or superuser privileges can display the contents of the file and obtain the password. To overcome this EXEC allows the option of password encryption. Once a password is encrypted it cannot be reversed. This means that if a user forgets his password (a fairly regular occurrence) there is no way of determining what it was.

MCP maintains an encrypted file of user names and passwords. The encryption algorithm is dynamic and changes based on time of day. It employs bit scrambling over the entire user name and password string. As both the user name and password are encrypted and can reside anywhere in the file the potential security risk is reduced. MCP can decrypt the file entries and thus user's passwords can be interrogated by the system manager. This is done by the system manager's interface which can only be run by the system manager. As an added precaution the system manager's password cannot be displayed.

MCP does not echo entry of a password during logon to prevent accidental disclosure. The user can change his password at any time if he has been granted the privilege to do so. This is done by entering the old password and pressing the erase page key. MCP then asks for the new password and stores it.

Passwords in isolation are not sufficient to provide a watertight system. MCP provides a number of additional features to enhance security. These include internal features and application program features supported by MCP.

4.2.1 Additional internal features

a) Fields are provided in the user profile for inclusion or exclusion of locations. This implies that the user can be constrained to log on at specific locations only or prevented from logging on at specific locations. The location in the profile is matched to the terminal location from the MCPLOCATIONS file. This allows, for example, payroll systems to be executed

in the pay office only.

b) Fields have been provided to specify time intervals between which users may logon to the system. This, for example, prevents a pay office employee from working alone after hours.

c) Change password monthly. If this option is set then the user is forced to change his password monthly. When he logs on for the first time in a new month he is asked to supply a new password. If he does not logon for two months the user name and password become invalid and must be cleared by the system manager. The system manager is seldom informed when employees leave the company. This feature prevents old employees from gaining access to the computer.

d) User limit. This feature allows the number of consoles logged on with the same username and password to be controlled. For high security profiles such as the system manager, this is normally set to 1.

e) Modem access. If this privilege is not set then a user may not use a dial up modem to access the system.

f) Virtual console access. If this privilege is not set then a user may not use this profile from a remote machine.

g) If ten invalid user name / password pairs are entered in succession on a console then that console is automatically disabled and has to be re-enabled by the system manager or operator. This discourages users from experimenting with passwords.

4.2.2 Application program features

In his paper on operating system structures to support security and reliable software [LIN76], TA Linden states that for security to be effective it should reside not only in the operating system but should permeate from the operating system kernel right through the applications software. Pursuing this philosophy MCP provides a facility to control

security through all menus in application programs. This means that access to individual menu options in application programs can be controlled from MCP without program re-compilation.

When a program is started, MCP creates a unique scratch file in the :PER directory containing the user's ADDITIONAL ID obtained from the profile. On startup, the program issues a call to the MCP utility routine GETUS. This routine fetches the additional ID from this file and also checks to ensure that the program is a direct son of MCP. This means that only that user name and password can execute the program and provides protection from a superuser who cannot execute it from the CLI. The additional ID is stored for later use.

Whenever the program receives a menu option number from the user it issues a call to the CHECKMEN utility routine specifying the menu name and option number. CHECKMEN then uses the additional ID, menu name and menu number to access a bit map file and validate if the user is permitted to access this option. Access to the bit map files is restricted to the systems manager and project leaders.

Each user name / password pair can have a unique additional ID and thus appropriate access to applications menus.

4.3 Program structure

This section covers the program logic behind the terminal environment. The intention is not to cover the program in detail but to rather give an overview of the logic used.

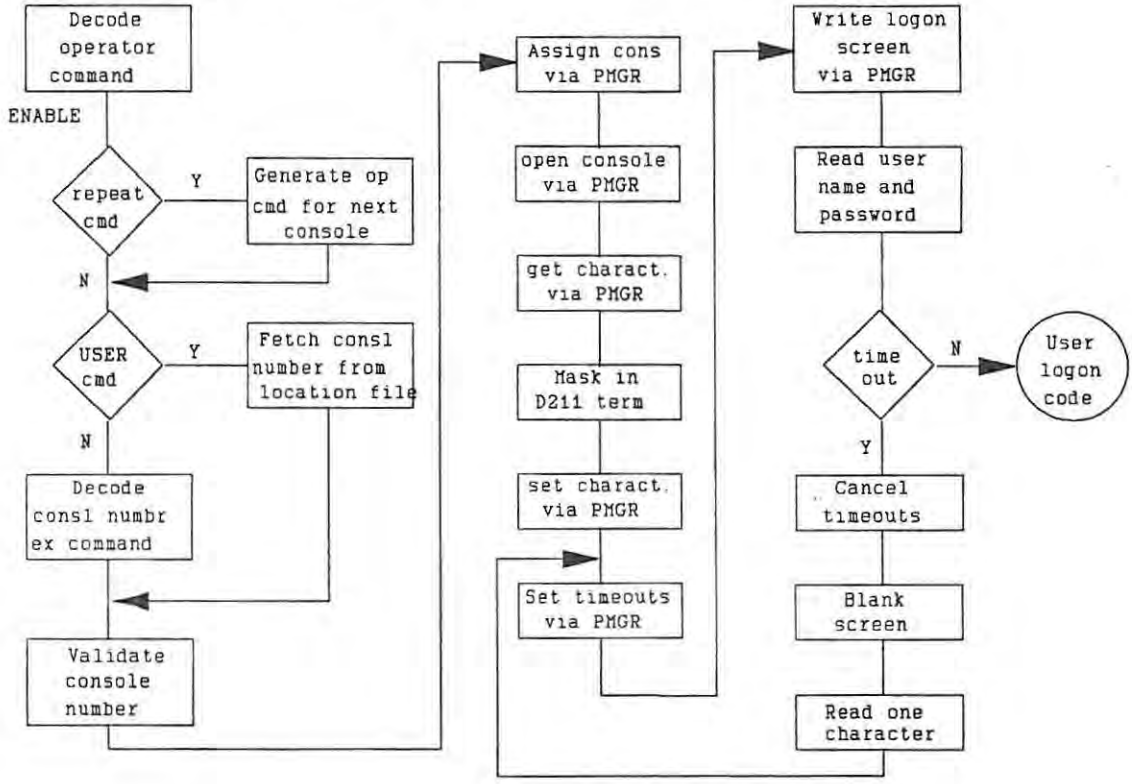
As discussed in chapter 3, at least one virtual task is created for each active async port. The virtual tasks for terminal devices run partly in the main task and partly in the STARTCON task. The interface between the tasks will be covered at the end of this chapter. The discussion on the virtual tasks will ignore the physical tasks for the time being.

There are five main code blocks for terminals, namely console enable, console disable, user terminate, user logon and user program termination.

4.3.1 Console enable

This code block is entered upon the receipt of an ENABLE command from the operator. If the command is for an explicit console, the console number is decoded from the command line. If the command is ENABLE USER then the console number is obtained from the MCPLOCATIONS file. If the command is repetitive (ie ENABLE ALL or ENABLE USERS) then a dummy operator command is sent to the IPC to enable the next console.

Figure 4.2 - Flow chart of the terminal enable code block



Once the console number has been determined and validated to ensure that it is valid and not in use, a series of commands are sent to PMGR (or SVTA in the case of virtual consoles). The sequence is as follows; assign console,

open console port, set timeout to 10 minutes, get characteristics, mask D211[15] into characteristics, set new characteristics, write logon screen, read user name and password. Once the user name and password have been entered control passes to the user logon code block described in 4.3.4.

If no input is received within 10 minutes then a timeout will occur. On receipt of a timeout the screen is blanked, timeouts cancelled and a single character read requested. Once a character is typed the program loops to the write logon screen code. This process is depicted by the flowchart in figure 4.2.

4.3.2 Console disable

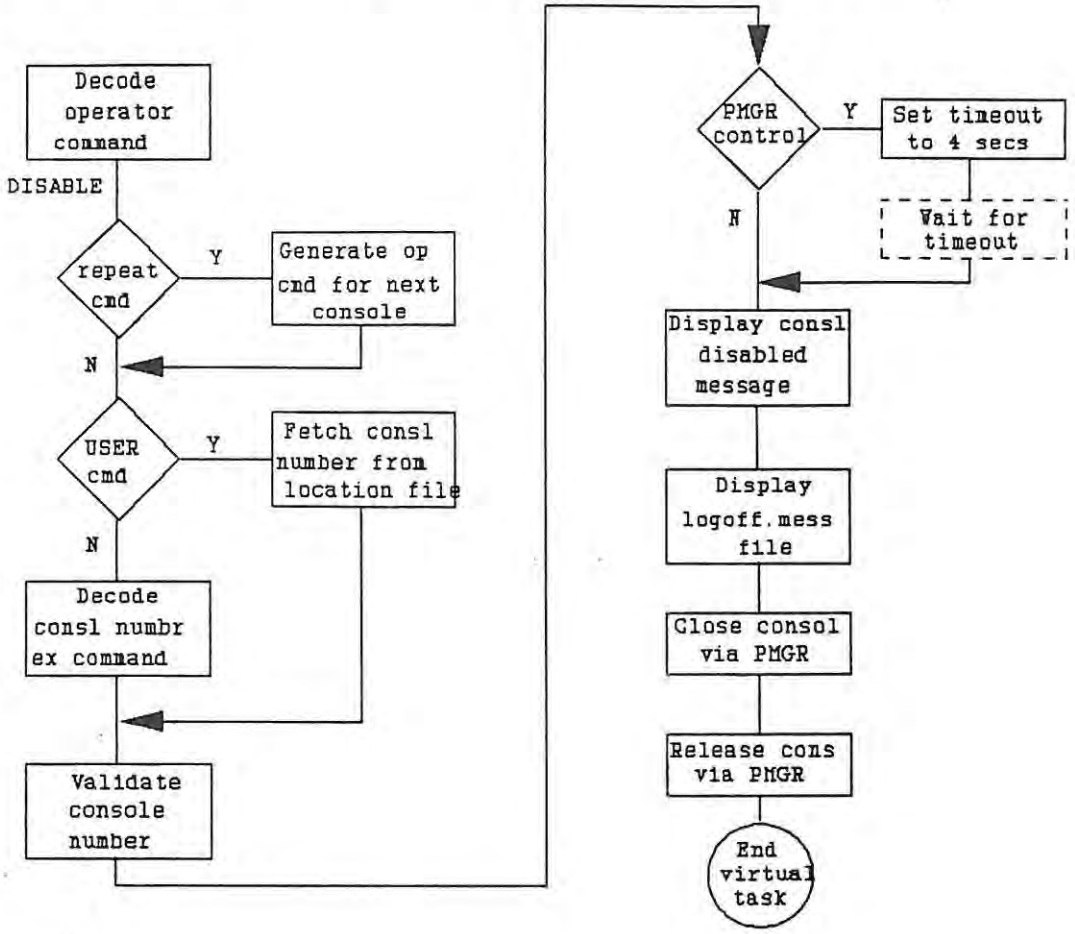
This code block is entered upon the receipt of a DISABLE command from the operator. As with the enable command checks are performed for DISABLE USER and the console number is obtained from either the command line or the MCPLOCATIONS file. The console port number is then checked to ensure that it is enabled as a terminal.

Disabling a console presents a problem because in 99% of cases control of the console at the time that the disable request is received does not lie with MCP but with PMGR. Thus it is necessary to force PMGR to relinquish control of the console. This is done by setting the console time out to four seconds and thus forcing PMGR to pass control of the device back to MCP with a timeout error message. Once the timeout message is received a disable message is displayed on the console followed by the contents of a special file called LOGOFF.MESSAGE. This file can be used by the system programmer to inform the user as to the reason for being disabled. Once this is complete the console is closed and released from MCP control. This process is depicted by the flowchart in figure 4.3.

---

[15] The D211 is the most basic terminal supplied by Data General

Figure 4.3 - Flow chart of the terminal disable code block



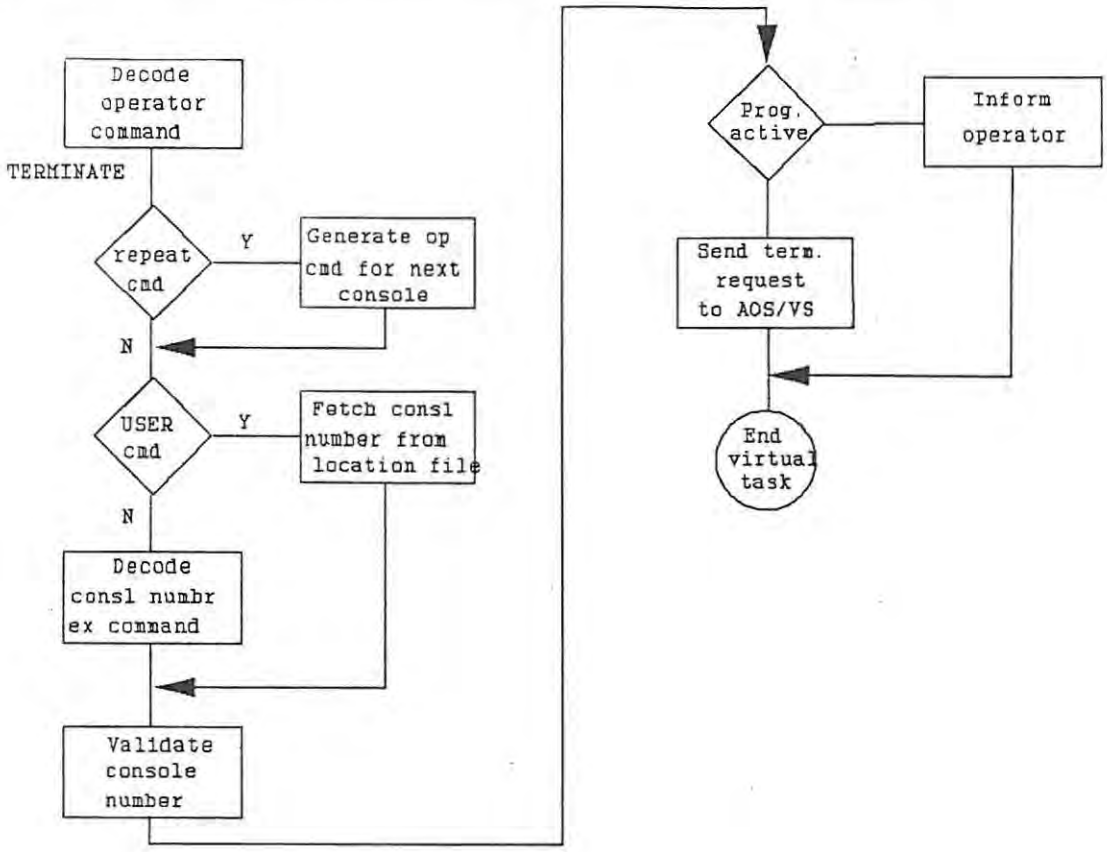
4.3.3 User terminate

This block of code is entered upon receipt of a TERMINATE command from the operator. As with the enable and disable commands, checks are performed for TERMINATE USER and the console number is obtained from either the command line or the MCPLOCATIONS file. The console number is checked to ensure that a user program is currently running on it.

Once the console number has been determined, the process ID (PID) of the program started on that console is looked up in the console table and a request issued to AOS/VS to terminate the process. When the process is terminated, AOS/VS sends MCP a termination message which results in the

execution of the code to re-enable the terminal (see 4.3.5). This process is depicted by the flowchart in figure 4.4.

Figure 4.4 - Flow chart of the user termination code block



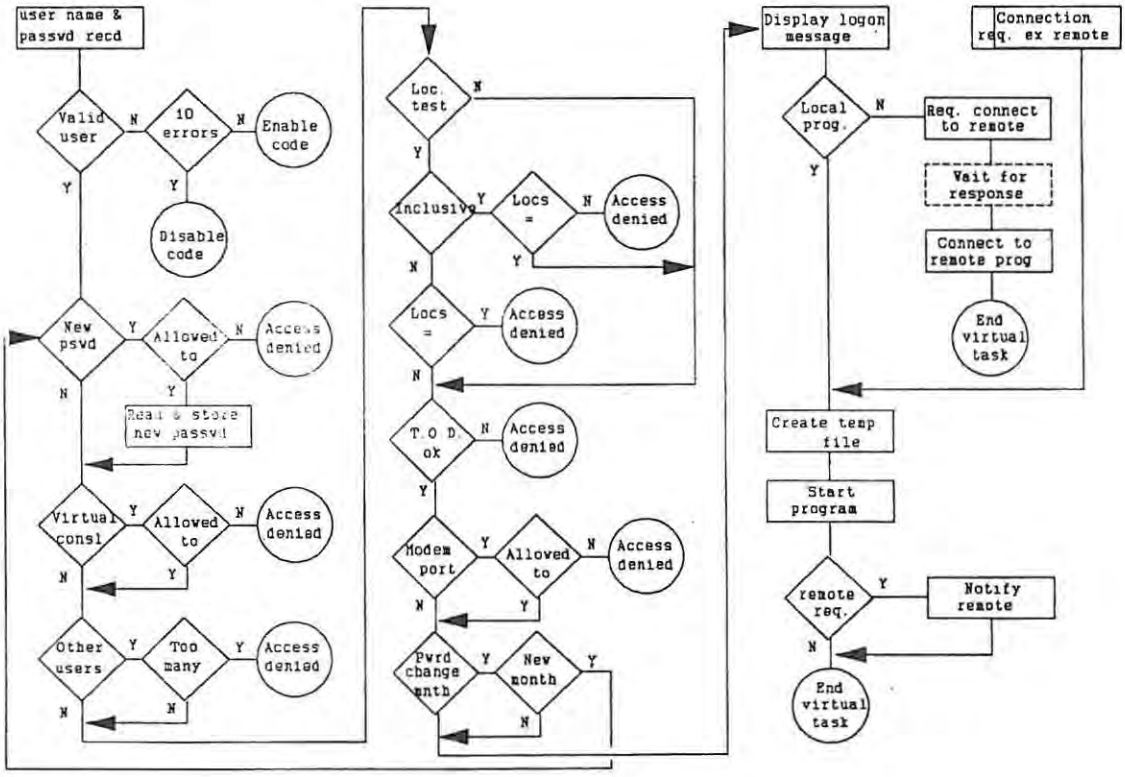
4.4.4 The Logon

The Logon block of code performs all the security checks described in section 4.2 and, provided that no security breaches are detected, starts the program specified in the profile for the user. First the username and password are checked and, if valid, the related profile file name obtained. If 10 successive invalid user name / password pairs are entered then a branch is made to the disable console code block. The termination character of the password is then checked. If it is an ERASE PAGE (octal 14) then if the user's privileges permit password change, the new password is input from the console and the user name / password file modified.

Once a valid user name / password in entered the profile file is read into memory. A number of security checks are now performed, namely if this is a virtual console then does the user have virtual console privileges, if there are other users logged on with the same user name then has the maximum user count been exceeded, is the user at a valid location, can the user log on at this time of day, if this is a modem port then does the user have modem privileges, if monthly password change has been specified then has this user changed his password this month.

Provided that all the security checks pass the LOGON.MESSAGE file is displayed on the console. If the profile's network node differs from the local machine's then a request for connection is passed to MCP on that machine. When notification is received that the remote process is running then a connection is established for the user,

Figure 4.5 - Flow chart of the user logon code block

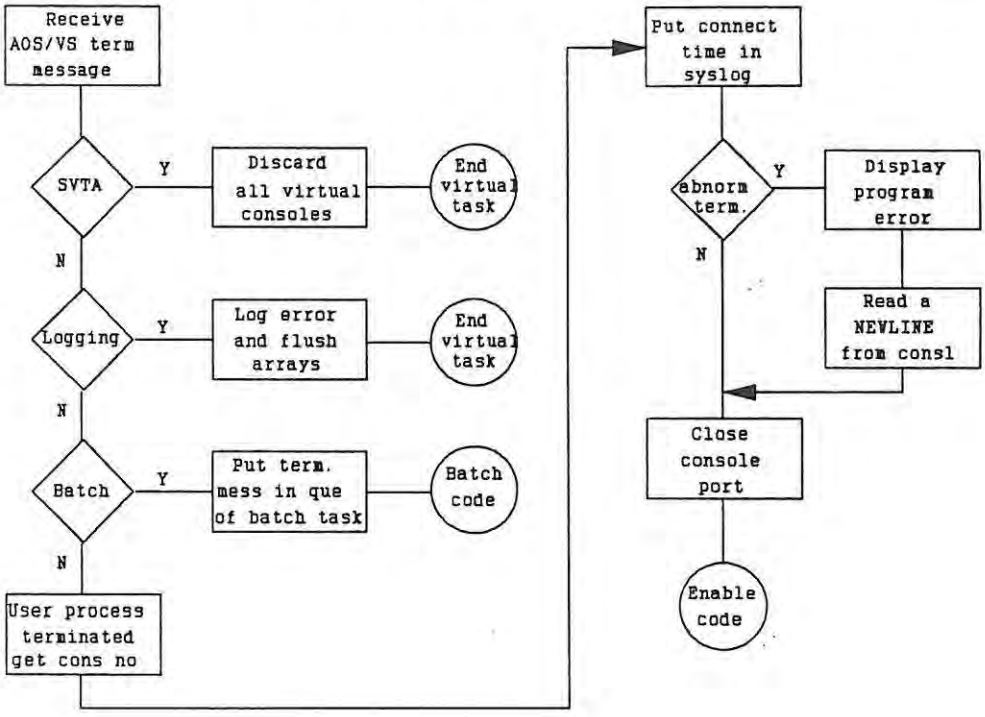


If the program is to run on the local machine or a request has been received from a remote MCP for connection then the temporary file containing the additional ID is created in :PER and the user's program is started with the AOS/VS privileges specified in the profile. If the program is running on the local machine then the virtual task suspends until a process termination message is received from AOS/VS when execution continues in the user program termination block discussed in 4.3.5. If this is a request for connection from a remote MCP then a confirmation message is sent to that MCP. See figure 4.5.

4.3.5 User program termination

This is the code block that is executed when a termination message is received from AOS/VS for one of MCPs sons. Firstly the PID is identified. It could be the SVTA process, the MCP logging process, a user program executing on a terminal or a user batch job.

Figure 4.6 - Flow chart of the user program termination code block



If SVTA terminated then all the virtual consoles are discarded and their arrays cleared. If the logging process terminated then the error is recorded in the log file and the arrays cleared. If a batch job terminated then the batch task is instructed to re-schedule (see chapter 7).

If a real time program terminates then an entry of connect time is created in the system log. If the program terminated abnormally then the error message is displayed to assist with debugging. The console port is then closed to drop modem control signals and a branch made to the console enable code block. This process is depicted by the flowchart in figure 4.6.

4.3.6 Physical tasks

All the code paths described above are located partly in the main task and partly in the STARTCON task. Any slow or long code such as disk access is placed in the subordinate task. This is to ensure that the main task can service the IPC port quickly enough to give fast response times to users. Furthermore any communication with PMGR results in a returned IPC message which is received by the main task. This means that many entry points to the subordinate task are required.

As discussed in chapter 3, communication between the tasks takes place by means of queues. The queues are processed by the subordinate task on a first come first serve basis. Both the console and queue data tables are in shared memory common blocks and are thus available to all tasks. The information passed from the main task to STARTCON in the queue includes:

a) Console number — needed as an index into the console data table.

b) Entry point — indicates the section of code to be executed.

In order to ensure that when the queue is empty the subtask does not loop needlessly, the task suspends on a TQREC inter task receive. Whenever the main task places an entry in the queue it issues a TQXMT to activate the

subordinate task if it is suspended.

The entry points to STARTCON are as follows:

ENTRY POINT	FUNCTION
1	Fetch the user's profile from disk and perform security checks. Instruct PMGR to input new password if necessary else skip to entry point 2.
2	Return from new password entry. Fetch LOGON.MESSAGE from disk and instruct PMGR to output it.
3	Return from logon message display. Handle remote MCP connection if applicable else create additional ID file and start user process.
4	Fetch LOGOFF.MESSAGE file from disk and instruct PMGR to display it along with the console disabled message.
5	Remote MCP has signalled that the user process is running - connect to it.
6	Request from remote MCP received, start the user's process and notify remote.

Each task has an AOS/VS scheduling priority. The main task is priority 1 (the highest priority), STARTCON has a priority of 5. The priority is relatively high to ensure that terminal users get good response.

5. The MCP PC environment

The PC has become an integral part of the computer equipment used in companies today. This is due primarily to the shrinking costs of PCs and the powerful software available for them. Considering the small cost difference between PCs and terminals it now makes economic sense to procure PCs in place of terminals for main frame applications.

The advent of local area networks and the OSI[16] communications standards will ensure a greater and greater role for PCs in corporate data processing in the future. These views are supported in a paper by Golstein, Heller, Moss and Wladawsky-Berger on directions in cooperative processing between work stations and hosts [GOL84]. The paper discusses the transparent interchange of data between PCs and main frames. The costs of establishing a local area network, particularly in distributed companies, is high and it is thus not always desirable to connect all PCs to the main machine by means of LAN.

For the above reasons support of PCs using asyncronous communications is highly desirable. Benefits of such a facility include:

a) A terminal emulator on the PC can obviate the need for terminals.

b) Many PCs can share a printer serviced by a queue on the main computer thus reducing peripheral costs.

c) File transfer over the async line can allow the PC user to access corporate data without the expense of a local area network.

d) Files can be transferred from PC to PC via the main frame thus minimizing data fragmentation and improving utilisation of the corporate data resources.

e) File translation between main frame formats and formats usable by PC packages.

---

[16] Open Systems Interface

f) A communications protocol can allow programs on the main frame and PC to talk to each other thus distributing work from the main frame and maximising utilisation of the PC.

As mentioned previously no PC support is provided by EXEC. The only software provided by DG for PC communications is a standalone package called CEO CONNECT. This package is not user friendly and provides the following functions :

a) A terminal emulator.

The emulator is not memory resident. This implies that if the user wishes to use terminal emulation then he has to exit whatever program he is executing. Furthermore the information on the screen from the last emulation session is lost.

b) File transfer facilities.

The package requires that the user log onto the DG, set up a search list to include the portion of the software package resident on the DG, go back to the PC and transfer the files. This is not user friendly and furthermore the communications protocol is not efficient. The transfer of a single file can take up to 10% of a 10 MIP MV20000 processor. If a number of PC users start transferring files then poor response for all users will result. The protocol used in MCP takes a maximum of 3% with an average of 1.7% to transfer the same file.

By comparing these functions to the possible benefits listed above it can be seen that CEO CONNECT leaves a lot to be desired as a PC support tool.

This chapter covers MCP's support for PCs. All of the facilities mentioned at the start of this chapter are supported. This is achieved by software in MCP and a memory resident addition to the BIOS on the PC. This chapter is divided into a discussion on the communications protocol used between the DG and the PC, the functionality of MCP support of PCs, The functionality of PC software to communicate with MCP, security, and finally program structure.

5.1 PC-MCP communications protocol

In designing a communications protocol, it is critical to optimize the use of the hardware on the main frame. The IAC16 asyncronous controller is programmed to support terminals. Terminal I/O normally consists of large quantities of buffered output and small data sensitive reads. Thus when performing data sensitive I/O, buffering of the input data takes place on the IAC board and is only passed back to the main CPU when a delimiter is encountered. PMGR receives this buffer and passes it across to the program as an IPC message. This is very efficient as the entire read takes place independent of the main CPU.

Binary I/O can also be used in which case any valid byte value from 0 to 255 (provided 8 bit communications is used) can be transmitted across the line. The problem is that no buffering or intelligence of the IAC is used. This means that each character received results in an interrupt of the main CPU. It appears that CEO CONNECT uses binary communications and that this is the main reason for its poor performance.

Using data sensitive I/O is all very well but the number of valid byte values that can be transmitted across the line drops to between 32 and 126. This is because any character less than 32 is a control character and can cause problems such as "control+S" which stops output from the IAC. The IAC masks the eighth bit to zero which means that any character greater than 127 cannot be transmitted accurately. Finally 127 is the delete character which causes the character preceding it in the buffer to be deleted.

This is not necessarily a major problem as most of the characters sent lie between 32 and 126. The problem is solved by adding an extra character to signify a value out of range and then adding 32 to control characters, subtracting 80 from characters between 126 and 197 and subtracting 150 from characters greater than 197. This means that out of range characters result in a two character code being sent. However the increase in performance on the DG more than compensates for the additional character (see figure 5.2).

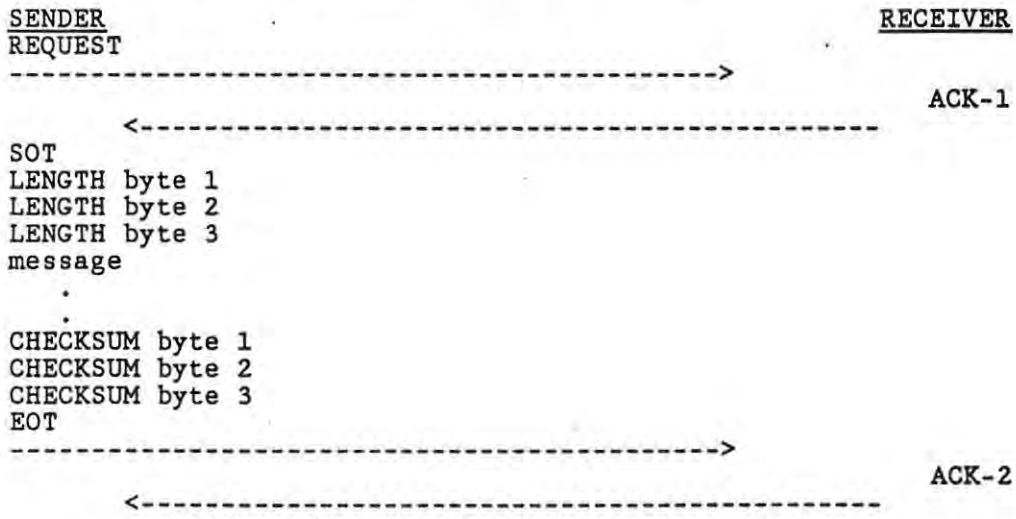
As the line between the PC and the DG is the biggest bottleneck to fast communications, character compression was also seen to be a necessity. This is achieved via a three character code. The first indicating that compression is taking place, the second is the number of characters, and the third is the actual character.

As asyncronous lines are not guaranteed error free, packetisation was considered necessary. Packets with a length count and checksum are employed to ensure data security.

The paper by A Tanenbaum and R van Renesse on distributed operating systems [TAN86] discusses methods of communication between computers. Briefly three methods can be used, namely Request-ACK-Reply-ACK, Request-Reply-ACK and Request-reply. The problem with the second and third methods are that if there is processing involved in generating the reply then no acknowledgment of the request is received by the sender for some time. For this reason the first option has been adopted for this protocol.

This is represented diagrammatically by figure 5.1

Figure 5.1 - MCP-PC communications protocol



As can be seen from figure 5.1 the sender is in control of the protocol and starts the transfer by sending a REQUEST. In order to convey information on the message type, 6 REQUEST characters have been provided for. They all have an identical

function as far as the protocol is concerned but have significance to the receiving program. Upon receipt of a REQUEST character the receiver sends an ACK-1 character. This is different from the ACK-2 character to ensure synchronization on noisy lines. If an invalid REQUEST character is received then a NAK is sent which restarts the protocol. When the sender receives the ACK-1, the packet is sent. As shown the packet is made up of a SOT (start of text) character followed by the message length as three ascii digits followed by the message itself followed by the 8 bit checksum as three ascii digits followed by an EOT (end of text) character. Once the receiver has received the packet, the length and checksum are checked and, if both are valid, an ACK-2 is sent to the sender. Both the ACK-2 and EOT characters can take on multiple values to allow additional information to be transferred between the machines in one transaction. If an error is detected in the packet then a NAK character is sent which restarts the protocol and decrements the retry count.

Eighteen of the thirty-two control characters are not used by the IAC in normal data sensitive input and can thus be used by the protocol. Two are used for switching to terminal mode and printer selection, the rest are used by the protocol as shown in figure 5.2.

Figure 5.2 - Control characters used by the protocol

Protocol	Character	Meaning
REQUEST	5	REQUEST and specify function 1 to receiver
	9	REQUEST and specify function 2 to receiver
	11	REQUEST and specify function 3 to receiver
	18	REQUEST and specify function 4 to receiver
	23	REQUEST and specify function 5 to receiver
	24	REQUEST and specify function 6 to receiver
NAK	21	Negative acknowledge - error in transmission
ACK-1	6	Acknowledge receipt of REQUEST
ACK-2	26	Acknowledge correct receipt of message
	"bit 6"	ACK-2 and error number in bits 0-5
SOT	2	Start Of Text
EOT	4	End Of Text and more messages coming
	25	EOT and no more messages
n/a	1	following char >= 127 so add 80
	8	following char < 32 so subtract 32
	28	following char >= 198 so add 150
	29	Character compression - next char is the count, following is the actual char

From figure 5.2 it can be seen that the sender can send a message to the receiver along with 6 functions to tell the receiver what to do. The sender can also tell the receiver if this is the last message or not via EOT. The receiver can tell the

sender if the message was processed successfully or not. If not then up to 63 error numbers indicating the problem can be returned via the "bit 6" option. All of this can be done inside the protocol without extra messages.

The PC protocol routines are contained in the BIOS extension explained in section 5.3 and are thus available to applications programs. The DG protocol routines are contained in a set of ring 6 resident utilities and can thus be used by applications programs as well as MCP. This allows applications programs to be developed on the DG and PC to perform transaction by transaction communications.

The protocol routines return a number of errors to the calling program. These are summarised as follows:

a) Errors returned from send

ERROR	MEANING
0 - 63	Message sent successfully but not processed successfully. These errors are specified by the receiving program.
32	Sendlock on - Sender is currently receiving a message and so cannot send.
64	REQUEST character not in range 1 to 6.
96	Too many retries - message could not be transferred.
128	Timeout - no response from remote machine.

b) Errors returned from receive

ERROR	MEANING
0	Message received successfully - more coming.
25	Message received successfully - last message. This allows the receiving program to perform housekeeping such as closing files.
128	Timeout - no response from remote machine.

By using this protocol it has been possible to utilize the full power of the IAC board and reduce system overhead to 20% of that used by CEO CONNECT.

5.2 MCP functionality

This section examines the manner in which MCP uses the protocol to satisfy the facilities discussed at the start of chapter 5. Section 5.3 discusses the functionality provided on the PC.

5.2.1 Users view:

The PC user sees a slightly different picture than the terminal user. MCP allows two possible modes for a PC. The default mode is PCNET mode. In this mode all the functionality of PC communications discussed in section 5.3 can be used. When running the emulator the following message will be displayed on the screen:

Switching to PCNET mode. Print queue LP1 .....

LP1 is the name of the MCP print queue to which his files will be sent when printing from the PC.

The second mode is terminal mode. In this mode the port behaves exactly like a terminal as described in chapter 4. The user can toggle between the two modes by typing ENTER when running the terminal emulator.

When in PCNET mode the user can type control+N from the emulator to enter a new queue for printing. The user can select any print queue on any networked machine running MCP.

5.2.2 System managers view:

A console port is enabled for a PC device by using the PCENABLE command. This is a special version of the ENABLE command discussed in chapter 4.1. The command takes an extra parameter which specifies the printer to be used as a default print device for PC print files. If a PC name is specified in the MCPLOCATIONS file for a port then when the ENABLE USER command is used the device is enabled as a PC automatically. In this case the default printer is taken from the MCPLOCATIONS file. The other terminal commands

(DISABLE, TERMINATE, CLEAR, RELEASE and LOCATION) function as described in chapter 4.

When in PCNET mode MCP will accept requests from the PC to perform a number of functions, including: output of data to a print queue, retrieval of files from a DG machine, retrieval of file names in a directory, fetching of information on other PCs in the network specified by node and PC name, retrieval of the contents of print queues, storing of a file in a DG directory, sending of data to another PC, retrieving data from another PC, inspecting queued files from other PCs, direct starting of programs on the DG, checking of username-password pairs, sending of print queue manipulation instructions to MCP, and changing of the print queue to which data will be sent. These functions are used by the PC software and are not directly visible to the user. In addition a command is provided for users in terminal mode to send a file to a PC's queue. Each function is discussed in more detail below.

a) Send data to a DG print queue

This function allows information to be sent to a DG print queue for printing. The print queue to which the data will be sent is set when the console is enabled by the systems manager and can be changed by the user either by using the control+N option discussed above or by using the change print queue option discussed under m) below.

The BIOS extension running on the PC intercepts data to be printed from the BIOS call, formats it into buffers of 70 bytes and sends it to the DG using this function. Because the data is intercepted on the BIOS call, PC-DOS is unaware that printing is not being output to the local printer and thus redirection of output is transparent to PC software running under PC-DOS. See section 5.3 for more detail.

b) Fetch directory contents matching the specified template

This function allows the PC to request MCP to send a list of all files in a directory on any networked DG. The buffer contains the network node name of the DG machine (or LOCAL for the DG to which the PC is connected), the

name of the directory, the template against which files should be matched, a user name and password to ensure that access is allowed to that directory. The files are returned from MCP in 80 character buffers, 8 file names per buffer.

This function is used to provide a user friendly environment where the file names in a directory can be displayed for the user. This enables the user to select those files he wishes to transfer using arrow keys in a similar fashion to most common PC software.

c) Fetch a file from a DG

This function allows the PC to request MCP to send the contents of a file in any directory on any networked DG. The buffer contains the network node name of a DG machine, the name of the directory, the name of the file to be transferred and a user name and password to ensure that access is allowed to that file. The records are returned from MCP with a five digit count of records remaining. This allows the PC program to inform the user of the percentage of the file transferred on an ongoing basis.

d) Fetch network information on PCs by PC name

This function allows the PC program to validate a remote PC name or template specified by the user. Each PC is identified to MCP by a PC name. This is similar to the network node name used by the XODIAC network software. The name is specified in the MCPLOCATIONS file and is used by MCP to route data from one PC to another. The PC name or template is sent to MCP which then returns with all PCs found with a name that corresponds. Along with the name MCP returns the DG network node to which the PC is connected, the print queue being used by that PC and the location of the device. This allows the PC program to validate the PC name entered or to allow the user to select the PC to which data is to be sent. The function is normally used prior to the transmission of a file to another PC.

e) Fetch the contents of a DG print queue

This function allows the PC to obtain the contents of any print queue on any DG included in the MCP printer network (see chapter 6). The buffer contains the name of the queue. MCP returns the sequence number, status flags, destination, forms and file name of each entry in the queue. This function together with function 1 allow DG print queues to be displayed and controlled from PC-DOS.

f) Send a file to another PC's queue

This function allows a PC to send data to another PC. As there is no guarantee that the remote PC is switched on or that it is running the appropriate software, the data is not sent directly to the PC but is placed in an MCP queue on the DG machine to which the other user is connected. The other user may interrogate his queue and fetch files from it at a later stage.

The buffer sent to MCP contains the DG network node to which the PC is connected, the PCs name, the file name of the file that is being sent and a notes field. The notes field is displayed to the remote user when he views his queue and can be used to provide a commentary on the data in the file. Once this buffer has been sent, the data is sent record by record to the relevant queue.

g) Send data to a file on the DG

This function allows the PC to send data to a file in any directory on any DG. The buffer sent contains the DG network node, the directory on that machine, the name of the file in which the data is to be stored, a user name and password to ensure that access to that directory is allowed and a flag. After this buffer, the data is sent record by record.

The flag is used to instruct MCP on how the DG file is to be handled and makes provision for append or overwrite of data and shared or unshared access. Shared access is achieved by setting the file's ACL to "+,OWARE".

Unshared access is achieved by restricting the file's ACL to the sending PC only.

h) Get contents of own queue

This function allows the PC to obtain the entries in its queue of files spooled from other PCs. The send buffer consists of a network node template and a PC name template thus allowing files to be selected for particular groups of PCs. MCP then sends all queue entries corresponding to the templates. The queue data returned includes the DG network node and PC name from whence the file was sent, the name of the file, the date and time it was sent and the notes field entered by the remote user when sending the file.

This allows the PC software to display queue contents to the user and allow him to select those files he wants to transfer to his PC.

i) Get a file from own queue

This function allows the PC to fetch a file from its own queue. The send buffer contains the file name to be retrieved and a flag. The flag allows the PC to instruct MCP as to what should be done with the file on successful completion of the transfer. The options are remove file from queue and leave file in queue for re-retrieval at a later date. MCP then returns the data record by record with a five digit field indicating the number of records remaining. This allows the PC program to inform the user of the percentage of the file transferred on an ongoing basis.

j) Log onto MCP

This function allows a PC to log onto MCP directly without going through the normal logon screen. This enables PC programs to log onto the DG and start programs running direct from PC-DOS. The PC would now normally be switched into terminal emulation mode to allow work to start on the DG (see section 5.3). When programs that are started using this function terminate, MCP automatically sends an instruction to the terminal emulator to terminate and

return control to PC-DOS.

Another use for this function is to start programs on the DG that will send information to the PC or receive information from it. This is the only MCP PC function that is required if a user is going to do transaction processing between the two machines via a special set of programs. The send buffer contains the user name and password to be logged on.

k) Check a user name and password pair

This function checks the user name and password passed in the buffer for validity and returns the appropriate error code. This function allows MCP security to be included in PC applications programs.

l) Send a queue manipulation command

This function allows one of the three queue manipulation commands (cancel, hold and unhold) to be issued against any queue in the MCP printer network (see chapter 6). The command, queue name and sequence number are included in the buffer and the appropriate error is returned.

m) Change the default print queue

This function allows the PC to change the DG print queue to which printing will be sent. This function is intended to be used in conjunction with function a) above. The send buffer contains the name of the new print queue.

n) Reassign console port

This function is intended for use with specialised transaction processing applications which require direct control of the async port connected to the PC. The send buffer contains an IPC file name and a message. On receipt of this message, MCP will disable and release the console and send the message to the IPC specified. The intention is for the process monitoring the IPC to take over control of the port. When the application is complete the process will instruct MCP to re-enable the port.

o) Transfer a file form the DG to a PCs queue

This function is accessed via the PCQUEUE command to MCP and allows a user in terminal mode to send a file to his or someone else's PC queue. This command is especially useful when reports are run in batch to generate data for the PC. Once the PCQUEUE command has been issued the file can be retrieved on the PC via function i) above.

If a problem is encountered by MCP in processing the above functions then an error is returned to the PC program. These errors are detailed in figure 5.3.

Figure 5.3 - Errors returned by MCP

ERROR	MEANING
0	Function completed successfully
1	Invalid user name / password pair
2	Invalid network node or directory
3	File does not exist
4	Access to data or directory denied
5	Invalid queue name
6	Write failed (normally because disk space exhausted)
7	Unable to access PC queue
8	Destination PC does not exist
9	Invalid message format
10	IPC does not exist
11	Invalid sequence number
12	Remote MCP not running
13	Sequence number not found in specified queue
14	Command already in effect
25	End of data send from MCP (end of file)
32	Sendlock on - DG still sending data, PC may not send
64	Invalid REQUEST number
96	Too many retries on data transmission - bad line
128	Timeout - DG not responding

errors 25 to 128 are protocol generated.

This concludes the PC support functions offered by MCP. They are not intended to be used directly by the user but rather by programs running on the PC to service the user. With the appropriate PC software they provide the facilities discussed at the beginning of the chapter. The software for the PC follows in the next section

5.3 PC functionality

The paper on tightly coupled work stations by D Chess [CHE84] discusses the modification of BIOS interrupt addresses to allow a memory resident program to intercept calls from PC-DOS to the BIOS and to perform a different function. This allows the memory resident program to operate transparently as far as PC-DOS and application programs are concerned.

This approach has been used on the PC side of the MCP-PC interface. An assembler program called MCPCOM is started as part of the AUTOEXEC.BAT file. MCPCOM is memory resident and occupies 8k of memory. The program re-defines the BIOS interrupt for async communications (interrupt 20). This has allowed a number of extra functions to support the MCP communications protocol and a D211 terminal emulator to be added. A more detailed discussion follows in section 5.3.1.

The print out re-direction to a DG print queue and the terminal emulator are catered for by MCPCOM. The rest of the facilities to enable file transfer are catered for by a program written in pascal entitled MCPFTS (MCP File Transfer System). A more detailed discussion follows in section 5.3.2.

5.3.1 MCPCOM functionality

More information on the following discussion can be found in the IBM PC hardware reference manual [IBM86] and the book entitled 8088 Assembler Language Programming [ASS85] by A Singh and WA Triebel.

When an interrupt is generated for Async communications to the BIOS, The registers are set up as follows; AH contains the function number, AL contains the parameter to be passed and DX contains the async port to be used. The functions offered by the BIOS include the following:

FUNCTION	MEANING
0	Initialise the UART with the parameters in AL
1	Output the character in AL
2	Receive a character into AL
3	Return the comms port status into AX

Functions 0, 2 and 3 remain unchanged in MCPCOM. The additional functions offered include the following:

a) AH=1 - Output the character in AL to the DG print queue

As this is a modification of a standard BIOS function it is transparent to the user program and PC-DOS. Obviously it would be inefficient to send a message to MCP for each character that is output. For this reason MCPCOM buffers output until a carriage return - linefeed sequence (end of a print line) is detected and then transmits the buffer. It is necessary to transmit at the end of each print line because MCPCOM has no way of knowing when output will end.

b) AH=4 - Enter the D211 emulator

This function allows the D211 emulator to be started from a program. Execution of the program continues on exit from the emulator.

c) AH=5 - Write a character to the screen in D211 format

This function allows PC programs to output characters to the screen in D211 format. All the D211 control codes for underline, cursor positioning etc are supported. This simplifies conversion of programs from DG to PC.

d) AH=6 - Read a character from the keyboard in D211 format

This function allows PC programs to input characters from the keyboard in D211 format. All the D211 keyboard codes for function keys, arrow keys etc are supported. This simplifies conversion of programs from DG to PC.

e) AH=7 - Initialise the D211 emulator

This function re-initialises the D211 emulator. It clears the saved screen and all the emulator's variables such as cursor position.

f) AH=8 - Send a message to the DG

This function implements the PC version of the send protocol discussed in section 5.1. The message buffer pointed to by DS:CX is sent to the DG using the REQUEST number specified in AL. The length of the message is contained as the first byte of the message buffer. This is in line with most PC language string variables (eg Turbo Pascal). The error value is returned in AX. Whenever the protocol is in use a status line is displayed at the bottom of the screen informing the user of the action currently being performed, and the error rate on the receive and transmit lines.

g) AH=9 - Receive a message from the DG

This function receives a message from the DG into the buffer pointed to by DS:CX using the protocol discussed in section 5.1. On completion, the first byte of the message buffer will contain the length of the message. This is in line with most PC language string variables (eg Turbo Pascal). The error value is returned in AX.

h) AH=10 - Re-direct PC printer output to the async port

Some PC packages have their own drivers for asyncronous communications and some have no async support at all. This creates a problem in that there is no guarantee that the BIOS asyncronous driver will be used and thus no guarantee that the output will be intercepted by MCPCOM and converted into the communications protocol for sending to the DG. This function has been included to allow re-direction of parallel printer output to the async port for transmission by MCPCOM to the DG. This insures that all data sent to the parallel printer is sent to the DG and is thus compatible with most PC packages. A program called

DGPRON is provided for the user to activate this function.

i) AH=11 - Reset PC printer data to parallel printer

This function cancels the affect of h) above. A program called DGPROFF is provided for the user to activate this function.

In addition to the extra async functions MCPCOM also contains a memory resident D211 emulator. This can be invoked or exited at any time by pressing the ALT key and the two SHIFT keys simultaneously. This is achieved by re-defining the clock interrupt. On receipt of a clock interrupt the old interrupt address is called to ensure that PC-DOS timing is not corrupted and then the state of the ALT and SHIFT keys is checked before returning from the interrupt. If the keys are all pressed then a branch is made to the emulator or if the emulator is already running then control is returned to the user's program.

The emulator maintains an extra screen in memory and thus when the emulator is entered the screen is restored to its state at the end of the last emulation session. When the emulator is exited the screens are again swopped thus restoring the display as it was before the emulator was entered. This allows the user to switch between the emulator and a PC program at will in a similar fashion to SIDE KICK[17].

In order to allow the emulator to function without the risk of loosing characters, UART interrupts are enabled and input from the DG spooled to a buffer. The buffer is then cleared as and when possible by the emulator.

The emulator provides a number of extra features not offered by a normal D211, namely; print passthrough from the DG to the PC printer and file transfer between the machines initiated from a DG application program.

Some of the ALT+key codes generated by the keyboard are used to allow the user to perform functions not related to the DG. These include ALT+E to exit

[17] A utility program supplied by Borland

from the emulator (equivalent to ALT+SHIFT+SHIFT), ALT+B to generate a break code to the DG, ALT+P to change the port number (this allows the emulator to switch between two machines connected to different ports) and ALT+L to switch the emulator between ON-LINE and LOCAL. In local mode all characters typed are echoed to the screen without being transmitted to the DG.

As the D211 only has 24 lines on a screen the bottom line is used for status information. Information displayed includes a summary of valid ALT commands, the current async port in use, the baud rate, parity, data bits, stop bits and whether the emulator is ON-LINE or in LOCAL mode.

5.3.1 MCPFTS functionality

MCPFTS uses the functions of MCPCOM to provide the user with a friendly but powerful file transfer system. In addition, file format conversion is done to allow files transferred from the DG to be converted to PC package format on transfer. The menu screen is shown in figure 5.4.

Figure 5.4 - MCPFTS menu screen

```
PC name : PCROB     MCP File transfer system     Console  : @CON132
Node    : SX20      ------------------------     Printer  : LP1
PC Port : COM1              Rev 7.6              Location : SALES
                          Main menu

          Option              Meaning
          ------              -------
     ---> 0       - - -      Exit back to MSDOS
          1                   Fetch files from a DG machine
          2       - - -      Send files to a DG machine
          3                   Print files on a DG printer
          4       - - -      View/fetch files sent from another PC
          5                   Send files to another PC
          6       - - -      Display DG print queue entries
          7                   Control DG print queue entries
          8       - - -      Log onto the DG directly
          9                   Change the PC port number

                  Enter the option you require : 0
```

For options 1 to 3 and 5 the user is asked for a directory (defaulted to current directory) and a file template (defaulted to *.*). If option 1 is being used then he will also be asked for a user name and password to maintain security. Option 4 will ask for a PC name template.

Once this information has been entered, all file names matching the template are retrieved from the specified directory or queue and displayed on the screen in alphabetical order. The user can then use the arrow keys to move the block cursor to the file(s) he wishes to transfer. To select a file the cursor is positioned on the file and the space bar pressed. The file name will then be displayed in high intensity. If there are too many files to fit onto one screen then the user may scroll the screen using the arrow keys or the page up and page down keys.

Once file selection is complete, the selected files are displayed on the screen in two columns. Fields are provided at the top of the screen for the destination directory and the user name and password in the case of sending files to the DG. Next to each file name a number of fields are provided. The first field allows the user to change the name of the file that will be created during transfer. The default is the same name as the source file. The second field allows the user to specify what action should be taken if the destination file already exists. The two options allowed are overwrite (default) and append. The other fields vary depending on the type of transfer and are as follows:

i) File transfer from the DG to the PC.

In this instance an extra field is provided to specify format conversion. The options include text format, Binary format, Data Interchange Format (DIF) and LOTUS 123 format. Text format converts the DG newline character to a carriage return and line feed character. Binary format does no conversion, DIF format converts the file to the DIF format used by a number of PC packages for data transfer and LOTUS 123 format converts the file into a format readable by LOTUS. These formats allow files to be transferred directly from the DG to most PC packages. The default for files with no extension is text, otherwise binary.

ii) File transfer from the PC to the DG.

In this instance an extra field is also supplied for format conversion. Only two options are supplied, namely; text format and binary format. Text format converts carriage return and line feed characters into the DG newline character. Binary format does no conversion. The default for files with no extension is text, otherwise binary.

iii) Print files on a DG printer.

No additional fields are provided. All transfer takes place using binary format.

iv) File transfer from the queue of files sent to this PC.

In this instance a field is provided to specify whether the entry should be removed from the queue after transfer or not. The options are yes (default) or no.

v) File transfer to another PC.

In this instance a field is provided for notes. The field is free format and has no default. Entries in this field will be displayed to the remote user when he views his queue.

Once these fields have been entered data transfer begins. A bar chart showing the percentage of the file transferred is displayed at the bottom of the screen. This keeps the user informed of transfer status. The files to be transferred are placed at the top of the screen with the one currently being transferred highlighted. Once a file has been transferred the status of that transfer is placed next to the file. This allows the users to easily see if a transfer failed due to security violation, disk full, bad line etc.

5.4 Security

Disk security is not a feature of PC-DOS. This is probably because PC-DOS was designed for a single user environment and thus it is not required. For this reason no security has been included on the PC side of the line. The DG on the other hand is a multi user environment and has great need of security.

When the PC is in terminal mode on the DG, all the security issues discussed in section 4.2 apply. When in PC mode the following measures are applied.

Firstly a special directory had been provided for users to use as a scratch directory called :PUBLIC. MCP does not enforce security in this directory and all files transferred from PCs are created with an ACL of +,OWARE (ie no access restrictions).

If the file(s) are to be transferred from a directory other than :PUBLIC then the user name and password sent from the PC are checked. If valid then the ACL of the directory in which the file(s) reside is checked against the user name. The user name must have read access in the case of a transfer to the PC or write access if the file is to be transferred from the PC.

In the case of a transfer to the PC, the ACL of the file is then checked. The user name must have read access to the file.

Files queued for other PCs are handled exclusively by MCP and are given no ACL. This means that only a superuser can access them and thus gives the PC users a guarantee of privacy. The queues themselves are stored in internal files maintained by MCP and are not directly displayable.

5.5 Program structure

This section covers the program logic behind the PC environment. The intention is not to cover the program in detail but to rather give an overview of the logic used.

As discussed in chapter 3, at least one virtual task is created for each active async port. The virtual tasks for PC devices run partly in the main task and partly in the MCPPCIO task. The interface between the tasks will be covered at the end of this chapter. The discussion on the virtual tasks will ignore the physical tasks for the time being.

When in terminal mode the PC virtual tasks make use of the code blocks discussed under terminals in chapter 4. This discussion will therefore be limited to the PC mode logic.

There are seven main code blocks for PC data transfers, namely Request handling, PC print data receive, PC to DG transfer, DG to PC transfer, PC to remote PC queue transfer, queue to PC transfer and direct logon.

5.5.1 Request handling

All PC virtual tasks start in this code block and then branch to the other code blocks. The primary function of this code block is to handle the communications protocol, to pass messages from the PC to the relevant code block to process them and to send messages from the other code blocks to the PC.

All virtual tasks using this code block start off suspended on a read of the PC port. All the protocol codes listed in figure 5.2 are declared as data sensitive delimiters thus ensuring that control is returned immediately to MCP from PMGR when a code is encountered.

Once communications with the PC begin, a timeout of 30 seconds is set on the port to ensure that no hangup of communications occurs. If a timeout occurs then the current function is assumed to be complete and housekeeping is carried out (ie closing files etc).

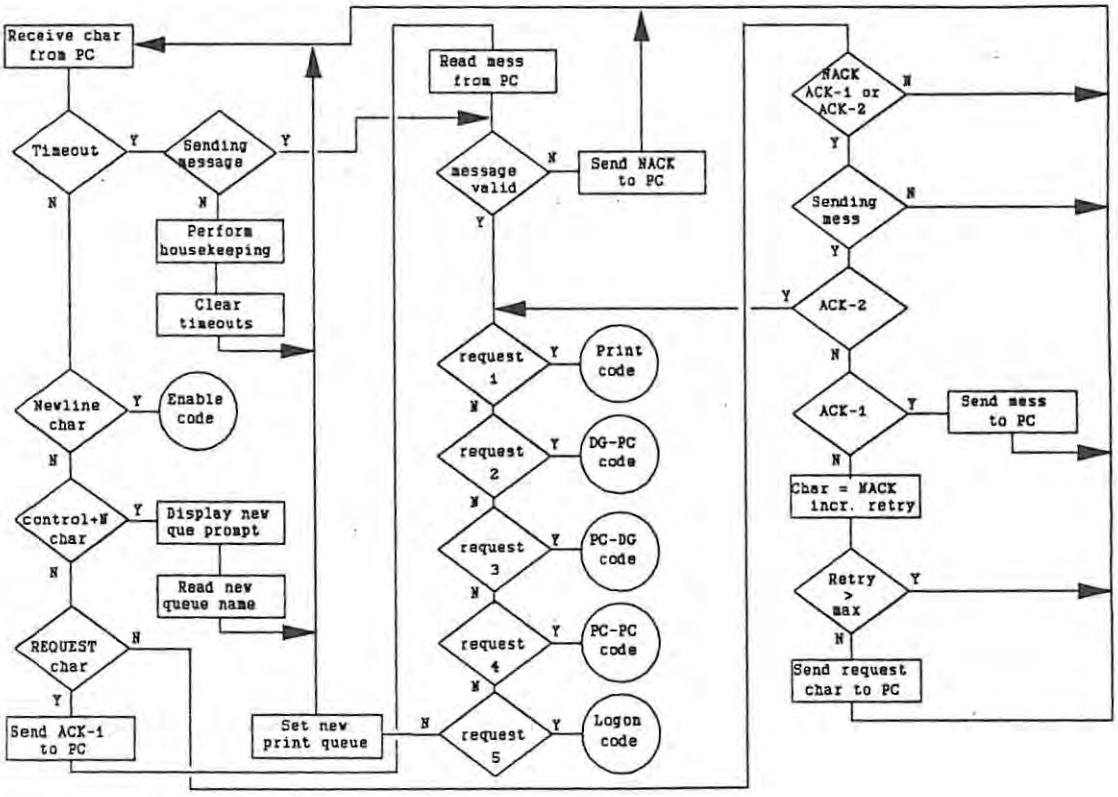
If a new line character is received then a branch is made to the terminal enable code block. If a control+N character is received then the user is prompted to enter the name of the new print queue for PC printouts.

If one of the protocol REQUEST characters is received then an ACK-1 is sent and a message is input from the PC. If the message is valid an ACK-2 is sent to the PC otherwise a NAK is sent. Once a valid message has been received a branch is made to the relevant code block to process it.

If an ACK-1, ACK-2 or NAK character has been received then, provided a send is in progress, the next step in the protocol is executed. In the case of the receipt of an ACK-1 the message is sent to the PC. If an ACK-2 has been received then a branch is made to the relevant code block to process the next message. If a NAK has been received then the retry count is incremented and if the maximum retries have not been exceeded, a new REQUEST character is sent to the PC.

A flow chart of the above logic is shown in figure 5.5.

Figure 5.5 - Flow chart of the PC request handling code

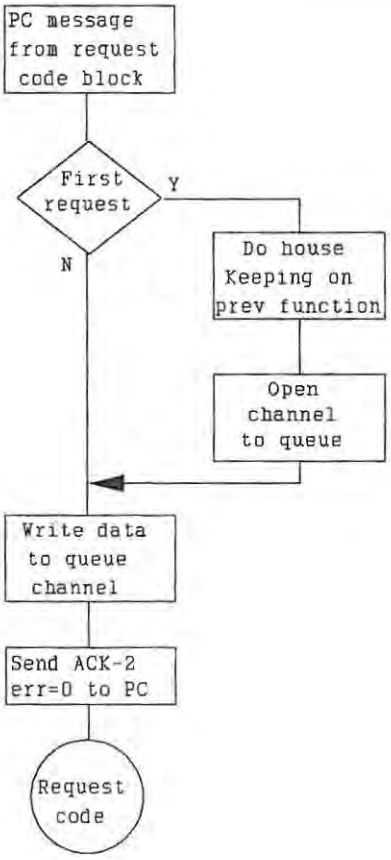


5.5.2 PC print data receive

This block first determines if this is the first record for printing. If so then housekeeping is performed on the previous function and a channel is opened to the print queue currently assigned to this PC. This results in a temporary file being created to receive the data. After this data is written to the file record by record until either a timeout or a different function request from the PC causes housekeeping to be performed. Upon successful completion of the write a "bit 6" ack is rent to the PC with error code 0.

The housekeeping code closes the channel to the queue which allows printing of the temporary file to start. Once the file is printed it is automatically deleted. See the discussion on printers in chapter 6 for more detail. A flow chart of this logic is shown in figure 5.6.

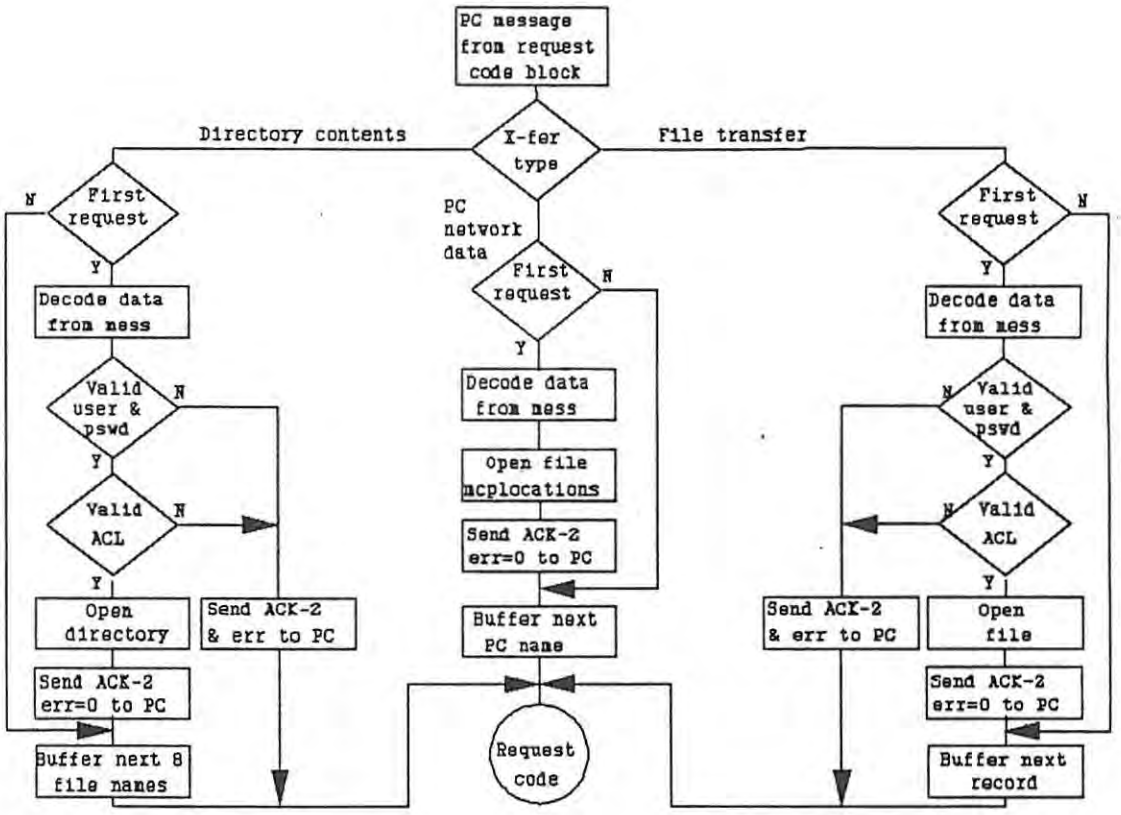
Figure 5.6 - Flow chart of the PC print data code



5.5.3 DG to PC transfer

This code block caters for file, directory and PC network data transfer to the PC. The message is decoded to determine which of the three transfer types is required. If this is the first request for this function then the data required to initiate the transfer is extracted. This consists of network node, directory, file name user name and password or in the case of PC network data the network node and/or PC name template. PC network data transfers will be discussed separately from here on.

Figure 5.7 - Flow chart of the DG to PC transfer code



If the directory is not :PUBLIC then the user name and password are checked for validity. If not valid then an error is returned to the PC otherwise the ACL of the directory is checked. If the user has access to this directory then the file ACL is checked. If not then an error is returned to the PC.

If all the ACL checks pass then the file is opened and the first record sent to the PC.

If this is not the first request for this function then the next record is read and sent with error=0 to the PC. When end of file is encountered then the last record is sent with error=25 and housekeeping performed.

In the case of PC network data transfer the MCPLOCATIONS file is accessed on the network node specified and data extracted for all PCs matching the template. This information is sent to the PC in a similar manner to files and directory contents discussed above.

A flow chart of the above logic is shown in figure 5.7.

5.5.4 PC to DG transfer

This code block caters for file transfer to the DG and connection requests to another process. The message is decoded to determine which of the two functions is required.

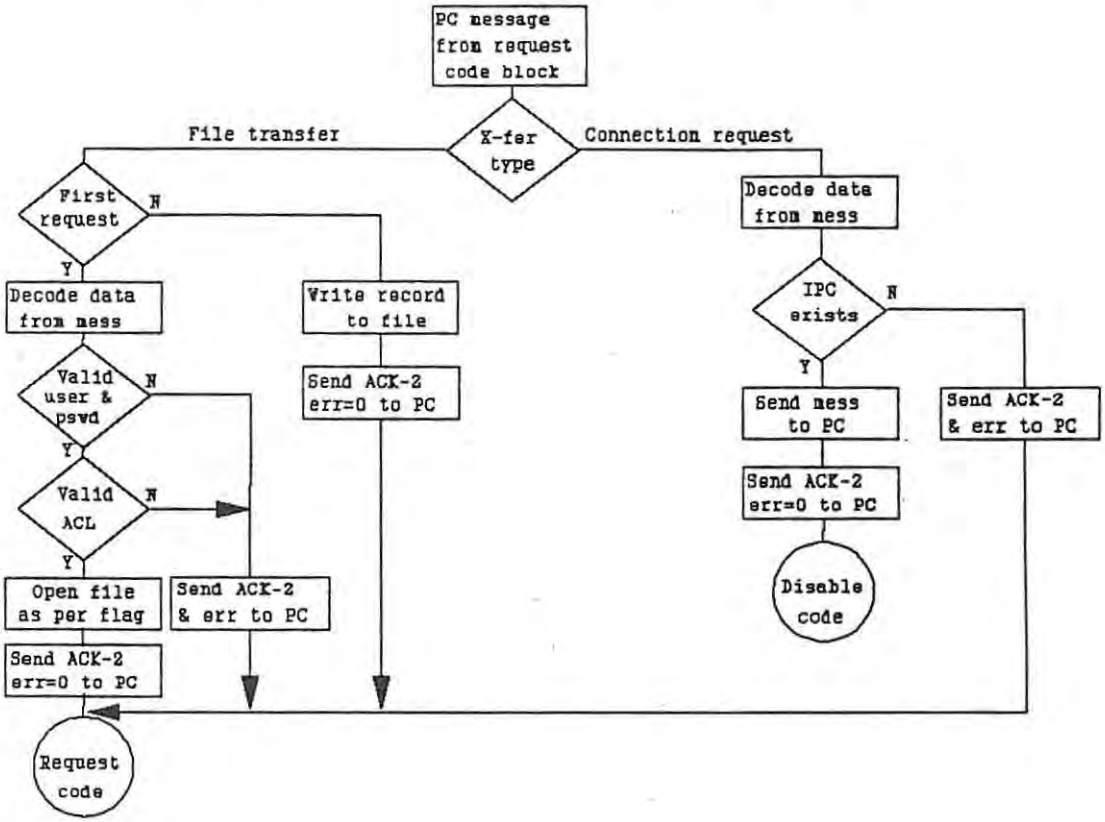
In the case of a file transfer a test is made for first request. If so then the data required for the transfer is extracted from the message. This includes the network node, directory, file name, user name, password and storage flag. If the directory is not :PUBLIC then the user name and password are checked for validity. If not valid then an error is returned to the PC otherwise the directory ACL is checked to ensure that the user has write access. If he does then if the file already exists its ACL is checked. If no security violations are detected then the file is opened either for append or overwrite depending on the storage flag. A branch is then made to the request code block to obtain the first record.

Records are then received from the PC and stored in the file and a "bit 6" ACK-2 sent to the PC with error=0. When the end of file is reached then housekeeping is performed.

In the case of a connection request, the IPC name and IPC message are extracted from the message. An attempt is then made to send the IPC message to the IPC specified by the PC. If successful then an ACK-2 with error=0 is sent to the PC and the port disabled from MCP.

A flow chart of the above logic is shown in figure 5.8.

Figure 5.8 - Flow chart of the PC to DG transfer code



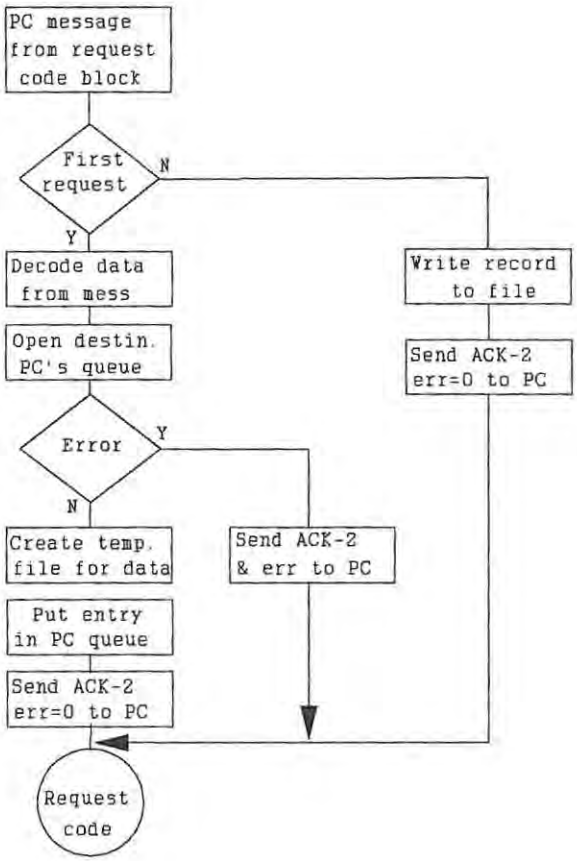
5.5.5 PC to queue transfer

This code block caters for transfer of a file from a PC to another PC's queue. If this is the first request for this function then the data required for the transfer is extracted from the message. This consists of network node, destination PC name, file name and notes.

An attempt is then made to open the PC's queue on the DG machine pointed to by network node. If this is successful then a unique temporary file is created to hold the data and an entry made in the queue, pointing to this file. The notes along with information on the source PC is also stored in the queue. If all of this is completed successfully then a "bit 6" ACK-2 is sent to the PC with error=0. The records are then received from the PC and placed in the temporary file. When the transfer is complete, housekeeping closes the file.

A flow chart of the above logic is shown in figure 5.9.

Figure 5.9 - Flow chart of the PC to queue transfer code



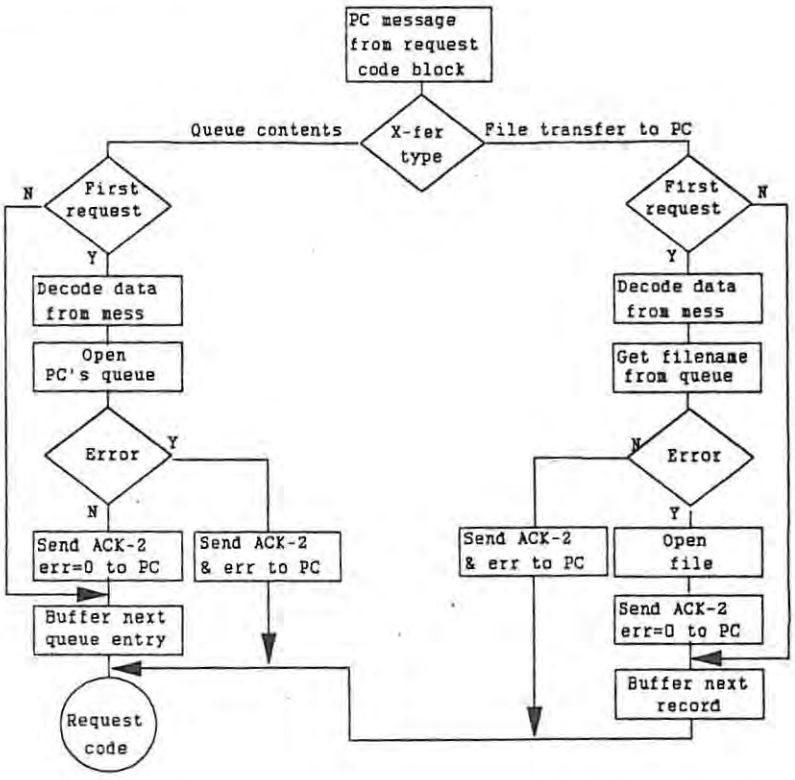
5.5.6 Queue to PC transfer

This code block caters the transfer of queue data and queued files to the PC. The message is decoded to determine which type of transfer is required. If this is the first request for the function then the data required to initiate the transfer is extracted. In the case of queue data, the network node and PC name template and in the case of a file the queue number and the remove flag.

If queue data is requested then the queue is opened and, provided no errors occur, the entries matching the template sent to the PC.

If a queued file is requested then the queue is opened and the temporary file name for this entry extracted from the queue. The file is then opened and the data transferred in a similar manner to the DG to PC transfer code block. A flow chart of this logic is shown in figure 5.10.

Figure 5.10 - Flow chart of the queue to PC transfer code



5.5.7 Direct logon

This code block extracts the user name and password from the message sent from the PC. These are then validated. If they are not valid then an error is returned to the PC else a branch is made to the user logon code block described in chapter 4.

5.5.8 Physical tasks

All the code paths described above are found partly in the main task and partly in the MCPPCIO task. Any slow or long code such as disk access is placed in the subordinate task. This is to ensure that the main task can service the IPC port quickly enough to give fast response times to users. Furthermore any communication with PMGR results in a returned IPC message which is received by the main task. This means that many entry points are required to the subordinate task.

The request handling code block described in 5.5.1 requires no disk access but does require a large amount of traffic to PMGR and thus has been located in the main task. As the main task has the highest AOS/VS scheduling priority, fast response for the communications protocol is also assured. All the other code paths with the exception of the direct logon code are disk intensive and have thus been placed in MCPPCIO.

As with the STARTCON task discussed in chapter 4, a queue is used to pass work from the main task to MCPPCIO. The console and queue tables are situated in shared memory common blocks and are thus accessible by both tasks. Data passed to the subordinate task in the queues includes:

a) Console number — needed as an index into the console data table.

b) Entry point — indicates the section of code to be executed.

c) Message buffer — holds the message received from the PC.

d) Message length — the length of the message.

e) Request number — the request sent by the PC.

Branching to the correct piece of code to process the contents of the queue is achieved by means of a combination of entry point and PC request number. There are three entry points, namely:

ENTRY POINT	FUNCTION
1	This is the housekeeping entry point. It is executed when a new request from the PC or a timeout is encountered. The PC request number passed in this case is the previous request number and is used to determine what files to close and what variables to initialise.
2	This is the received message entry point and is executed whenever a message is received from a PC. The request number is used to determine what code block to execute.
3	This is the "message sent" entry point and is executed whenever a message has been sent to the PC. Again the request number is used to determine what code path to execute to generate the next message.

For each code path the PC request number is used to determine which code block to execute. The PC request numbers are as follows:

PC REQUEST NO	FUNCTION
1	Print data code block
2	DG to PC transfer code block
3	PC to DG transfer code block
4	Queue to PC transfer code block
5	PC to remote PC queue transfer code block

Each task has an AOS/VS scheduling priority. The main task is priority 1, MCPPCIO has a priority of 4. The priority is relatively high to ensure that

PC messages are completed before a timeout occurs. This is necessary because MCP only sends the ACK-2 character once the transaction is complete so that any errors encountered can be fed back to the PC. This high priority does not seriously impact on the response of the system as most of the work done by the task involves disk I/O which causes the task to suspend.

6. The MCP printer environment

The primary purpose of a printer environment is to allow many users to share one physical printer in an orderly fashion while providing an environment where the printer "appears" to belong to each user individually. This is achieved by means of print queues. The user can either send files to the printer for printing or open the print queue as though it was a printer port and write data to it record by record. In this case, when the channel to the print queue is closed then the data written to it is printed.

While EXEC provides this functionality adequately, there are other requirements that a user has of a printer that EXEC does not provide. These include:

a) Modern dot matrix and laser printers have considerable intelligence built in that can and should be exploited. This includes print format commands such as italic printing, letter quality printing, tab stops, condensed and expanded print. Page formatting commands include margin setting and automatic perforation skip. As was discussed in section 2.7 the user is caught in a "catch 22" situation under EXEC in that page formatting is controlled via software and is only available when printers are in text mode. However in text mode no control characters can be sent to the printer thus preventing use of the printer features. In binary mode control characters can be sent to the printer but then no page formatting is provided and the user must get to grips with the programming codes for the device.

This dilemma is resolved by MCP in the following manner. The system manager creates device files for each printer type used on the system containing all the programming codes for that printer. Format files are then created that allow different page and print formats to be entered. The user specifies one of these formats when the file is submitted for printing. When the file is printed, MCP matches the format to the printer codes held in the device file and programs the printer accordingly.

This has the additional advantage that no software is required on the DG to control page formatting and thus overhead is reduced.

b) If special forms are loaded on a printer then only files destined for that form should be output. All other files should be held in the queue until the paper is changed. Under EXEC this is achieved by limiting printing to a particular form. The problem is that the operator must issue the command to change forms. When printers are located in the computer room this is an effective method of control but when the printers are numerous and distributed throughout the company at user locations, as is the case in a real time environment, then the user has to phone the operator and request him to change forms every time the paper is changed. This is frustrating and inefficient.

The approach taken under MCP is to allow the user to restrict printing on a printer to those files queued with a particular format file via the SETFORM command. The restriction is then cancelled by issuing a SETFORM NONE command or changing the restriction to another format file. Two options exist for implementing this command.

One is to allow only the user who issued a SETFORM to change it. The problem with this approach is that if the user forgets to release the printer after printing is complete then the printer is effectively frozen for print jobs from other users or with different formats.

The second option is to allow any user to issue a SETFORM command to change or release the restriction. The problem with this approach is that a user can change the forms before the paper is changed thus causing output to be printed on the wrong paper.

Of the two options the second is regarded as the better solution. The worst case for this option is that some printing paper is wasted. As the printers are located in the same location as the users this is not likely to occur frequently.

c) As discussed in the Tanenbaum and van Renesse paper on distributed operating systems [TAN86], in a distributed operating system the user should not be aware of the different machines in the network. This principal applies as much to printers as it does to terminals and PCs. Under EXEC the network

node name and print queue name have to be specified as part of the print command which is consistent with the network approach implemented by DG.

MCP adopts a more distributed approach by allowing the system manager to create a printer network. This is done by entering those machines that should share printers in a special MCP file called MCP_NET. Once this file has been created, the user can access any queue on any machine by name only. This is achieved by maintaining a record of all print queues in the network on each machine. Files submitted for printing on printers connected to remote machines are re-directed to that print queue automatically. All queue manipulation commands are re-directed in a similar manner.

d) A final issue not of direct concern to the user is the method of driving the printers. On EXEC a program called XLPT is started for each printer. The work done by these routines is minimal in that blocks are read from disk and passed to PMGR for printing. On systems with large numbers of printers the system overhead in terms of scheduling and memory is out of proportion to the functionality of the program. Extra system overhead is also incurred every time a printer is started or stopped because an XLPT program has to be loaded or terminated.

In MCP printers are handled via virtual tasks executing re-entrant code. This means that printers are serviced sequentially but as the vast majority of the time is spent in PMGR outputing the data, no degradation in print speed is experienced.

The following discussion highlights the functionality of the printer environment and the program structure used to achieve it. No mention is made of security as access restriction to print queues was not considered necessary or desirable. The only security enforced by MCP in the printer environment is format file restriction which has already been discussed in b) above.

6.1 Functionality of the printer environment

As has been intimated in the previous discussion, the MCP printer environment attempts to address the deficiencies inherent in the EXEC printer environment

without sacrificing any of the functionality. The discussion is divided into the users view and the system managers view.

6.1.1 User's view:

The user can use the queue in two ways. Either files can be queued for printing with the PRINT command issued from the CLI or the queue can be opened as a file and records written to it. In this case printing starts once queue file is closed. Commands that can be used by the user to control queued files include:

a) PRINT command

The print command is actually a CLI macro that provides users with help, allows multiple files to be queued simultaneously, accepts templates which are expanded into file names to be queued. When all the files are determined they are sent to MCP one by one. This is achieved by means of the CLI command CONTROL @MCP that allows messages to be sent to the MCP IPC port.

The options that may be selected by the user when queuing files are as follows:

SWITCH	MEANING
/QUEUE=queue name	Files are sent to the specified queue (no default - must be specified).
/FORMS=format name	Program the printer with the print and page formats specified in this file (default use printer's default print format).
/HOLD	Do not print file - hold it in the queue until the user issues an UNHOLD command (default do not hold - print immediately).
/NOTIFY	Display a message on the user's terminal when printing is complete (default no not notify).
/NORESTART	If the machine goes down or the printer is stopped while this file is printing do not

	re-start it when printing resumes (default restart printout).
/DELETE	Delete this file when printing is complete (default do not delete).
/PRIORITY=nnn	Set the priority of this print job (default 127).
/COPIES=nn	Specify the number of copies of the file to print (default 1).
/HEADERS=n	Specify the number of header pages to print in front of the file from 0 to 2 (default 1).
/TRAILERS=n	Specify the number of trailer pages to print after the file (default 0).
/DESTINATION=	Specify the printout destination string to be printed in expanded print on the header page.

When the file is submitted to the queue MCP returns a confirmation message to the user.

b) Display queue contents

The user can display the contents of the queue by means of the DQ command. As with print above this is a CLI macro that allows the user to specify switches and results in a QDISP command being sent to MCP.

The normal information that is displayed includes status of the queue (ie if it is open or closed) and a list of the files in that queue. Information displayed per file includes the sequence number of that queue entry, the flags (corresponds to the PRINT options without arguments as shown above), the destination of the printout and the file to be printed. An asterisk is displayed next to each queue entry that is currently being printed.

The options that may be selected by the user when displaying queue contents are as follows:

SWITCH	MEANING
/QUEUE=queue name	Specify the print queue whose contents are to be displayed (default all queues).
/SUMMARY	Display summary information only. Use of this option causes only the queue name, status and number entries in the queue to be displayed.
/VERBOSE	Display extra information on queued files. In addition to the normal information the format file, number of copies and priority are displayed.

c) Hold a queue entry

This command allows a user to prevent immediate printing of a queue entry. The entry will be left in the queue until an unhold command is specified.

d) Unhold a queue entry

This command releases a previously held queue entry for printing.

e) Cancel a queue entry

This command allows a queue entry to be deleted without printing.

f) Setform

This command allows the user to restrict printing on a queue to a specific format file. The restriction can be lifted by specifying a format file of "none".

If the queue is opened as a file and records written directly to it then a temporary file is automatically created to accept the data. Once the program closes the channel to the queue the file is submitted for printing. The delete flag is set so that when printing is complete the temporary file is deleted. The format file is set to the default format file for that printer

and the destination is set to the user name of the user process. The priority is set to 127 and copies set to one.

6.1.2 System manager's view:

The system manager can manipulate queues and printers. Queue commands with the exception of the CLOSE command can only be issued if the queue is closed and printing is not in progress on the queue. Queue commands are as follows:

a) Create a queue

This command allows a queue with the name specified to be created. A file with this name is created in the :PER directory. Users can open this file to output direct to the queue from programs.

b) Delete a queue

This command removes a queue name created by CREATE.

c) Purge a queue

This command allows all the entries in a queue to be removed.

d) Open a queue

This command allows a queue to be opened for use. Once the queue is opened users can start submitting files for printing.

e) Close a queue

This command closes a queue and thereby prevents the users from submitting entries to it.

Once the queue has been set up the system manager must allocate resources to process it. Many options are open to him. One printer can be allocated to each queue, Many printers can be allocated to one queue or one printer

can be allocated to many queues. Printers are allocated to queues via the START command. The start command accepts the following arguments; the console port number to which the printer is attached, the queue name that it is to process and the device type of the printer. The device type is used to generate the printer's programming codes.

Once the printer is started for one or more queues, the defaults are specified. These consist of the following:

a) Cleanup file. The contents of this file are copied to the printer after each printout. This is used mainly for laser printers to ensure that they are reset to a known state after printing of a word processing file.

b) Default format file. This is the format file used when no format is specified by the user when a file is submitted for printing. It is normally set up for standard paper size and normal print.

c) Number of headers. This specifies the number of header pages to be printed in the range 0 to 2 and can be overridden by the user during printing. an example of a header page is shown in figure 6.1.

d) Number of trailers. This specifies the number of trailer pages to be printed in the range 0 to 2 and can be overridden by the user during printing. an example of a trailer page is shown in figure 6.2.

Once the defaults are specified the printer is enabled for printing via the CONTINUE command. If the ENABLE USERS command is used to activate peripherals then the queue name, device and defaults are fetched from the MCPLOCATIONS file and the entire process explained above is carried out automatically by MCP for each printer. This considerably reduces the complexity of starting the machine as well as speeding up the process considerably.

Figure 6.1 - A typical header page produced by MCP.

```
1111111111111111111111111111111111111111111111111111111111111TESTFILE

                              OPERATOR

                        QUEUE=LP1         DEVICE=@CON120

                SEQ= 24536    QPRI=127    LPP=68    CPL=132    COPIES=1

                              FORMS=B600.FRM

                        PRINTED on 7-10-1988 at 12:19:41

                PATH=:NET:SXDEV:SXDEV3:TST:MCP:SOURCE:TESTFILE

                              TESTFILE

1111111111111111111111111111111111111111111111111111111111111111111111

                        M S & A   NETWORK NODE : SX20

                              MCP REV 7.7
```

Note : The destination is displayed at the top of the page (OPERATOR in this example) and is defaulted to the user name unless changed by the user in the PRINT command.

Figure 6.2 - A typical trailer page produced by MCP.

```
1111111111111111111111111111111111111111111111111111111111111111111111

                              OPERATOR

                        QUEUE=LP1         DEVICE=@CON120

                SEQ= 24536    QPRI=127    LPP=68    CPL=132    COPIES=1

                              FORMS=B600.FRM

                PATH=:NET:SXDEV:SXDEV3:TST:MCP:SOURCE:TESTFILE

                              * END *

1111111111111111111111111111111111111111111111111111111111111111111111

                        M S & A   NETWORK NODE : SX20

                              MCP REV 7.7
```

In addition to the commands already mentioned to activate a printer, a number of additional commands are provided for ongoing control of the printer environment. These include:

a) Pause printer. This command allows the printer to be paused so that defaults can be changed or the printer stopped. Printing will cease at the end of the current print job. A continue command will allow the printer to resume printing.

b) Stop printer. This command is the opposite of start and de-allocates a printer from a queue. When a printer is stopped the console port is released by MCP.

c) Flush printer. This command causes the job currently being printed to be discarded.

d) Restart printer. The print job currently in progress is restarted from the beginning.

e) Silence printer. This command prevents messages from being sent to the master console whenever a printout is started (this is the default condition on starting a printer).

f) Unsilence printer. This command causes a notification message to be sent to the master console whenever a printer starts a new print job.

g) Spoolstatus. This command informs the system manager as to what job is currently in progress for a queue or printer. The defaults for each printer assigned to the queue are also displayed.

Apart from controlling the printers the system manager is responsible for creating format files. This is done via a utility in the system managers tool box discussed in chapter 9. The options that can be selected in a format file are as follows:

a) Characters per inch from 5.5 to 16.5.

b) Lines per inch from 2 to 12.

c) Form length in lines.

d) Vertical margin (top and bottom).

e) Horizontal margin (left and right).

f) Tab stops.

g) Font number (for laser printers).

h) Print quality (normal or letter quality).

i) Print style (normal or italic).

j) Print type (normal or elite).

k) Print size (normal or double height).

l) Print orientation (portrait or landscape).

m) Spacing (fixed or proportional).

6.2 Program structure

This section covers the program logic behind the printer environment. The intention is not to cover the program in detail but to rather give an overview of the logic used.

The single biggest problem with MCP is the way in which direct program output to a queue is handled. As was discussed in chapter 2 when a queue is opened by a program as an output file, the AGENT re-directs the output to EXEC. This is probably done via the EXEC IPC port. This poses a problem to MCP as no source

code for the AGENT is available and thus it has not been possible to re-direct the output to MCP instead of EXEC. If the assumption that the data is sent to EXEC via the IPC port is correct then it should be possible to write a program that looks like EXEC to create and read an @EXEC IPC port and interpret the messages. This would then allow the appropriate code to be inserted in MCP to receive and process these messages from the AGENT.

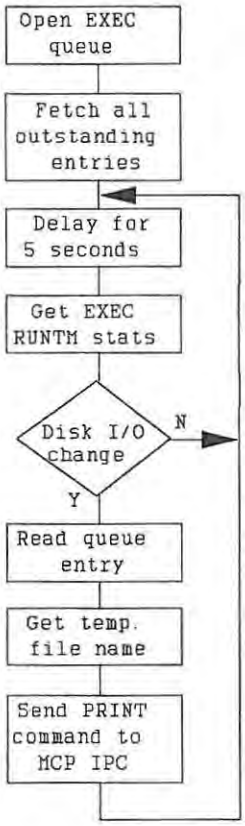
This has unfortunately not been possible as the machine on which MCP has been developed runs 24 hours a day and can not be taken down for experimentation.

The approach that has thus been adopted to achieve this functionality is to leave the EXEC process running purely for the purpose of processing the direct output from programs. EXEC places this output in a temporary file and creates an entry in its queue for the temporary file produced. MCP then picks up this queue entry and prints the file. This is achieved by regularly fetching the runtime statistics for EXEC via the ?RUNTM system call. This call gives the CPU seconds, disk I/O and memory usage of a process. If the disk I/O has changed since the last time that data was retrieved then it means that EXEC has updated its queue. MCP has a file pointer to the last file entered in EXEC's queue and can thus simply read the next entry with one disk read. While not an elegant solution this method seems to be the best until more information about the EXEC-AGENT interface can be found.

The system manager need not be aware of EXEC's existence because when queues are created or deleted MCP automatically instructs EXEC to duplicate the command and thus guarantees consistency between the two systems. As no printers are started under EXEC, the EXEC queue entries remain until the system is re-started.

As discussed in chapter 3, at least one virtual task is created for each active async port. The virtual tasks for print devices run partly in the main task, partly in the MCPPRT task and partly in the PRTDISP task. An additional standalone task called MCPREAD is used to detect and transfer direct output temporary files from the EXEC queue. The logic of this task has already been discussed and is shown as a flow chart in figure 6.3

Figure 6.3 - Flow chart of the EXEC to MCP queue transfer (MCPREAD task)



An additional standalone task is used to obtain information on the printers to be included in the printer network from remote machines called MCPNET. This task obtains the network node names of all machines included in the printer network from the MCP_NET file created by the system manager. The remote MCPs are then requested to return all their print queues. The remote print queues are included in the local MCP's queue tables. Any referance to these remote queues will result in the local MCP re-directing the command to the remote MCP. Once all the queues on all the machines specified in the MCP_NET file have been obtained, the MCPNET task suspends itself.

The interface between the other tasks will be covered at the end of this chapter. The discussion on the virtual tasks will ignore the physical tasks for the time being.

The main code blocks executed by printer virtual tasks are the printer activation code block, the queue scheduling code block, the file print code block and the command processing code block. These are discussed in more detail below.

6.2.1 Printer activation code block

This code block deals with the activation and de-activation of a console port as a print device. As with the terminal enable and disable code, this is very PMGR intensive and is located in the main task.

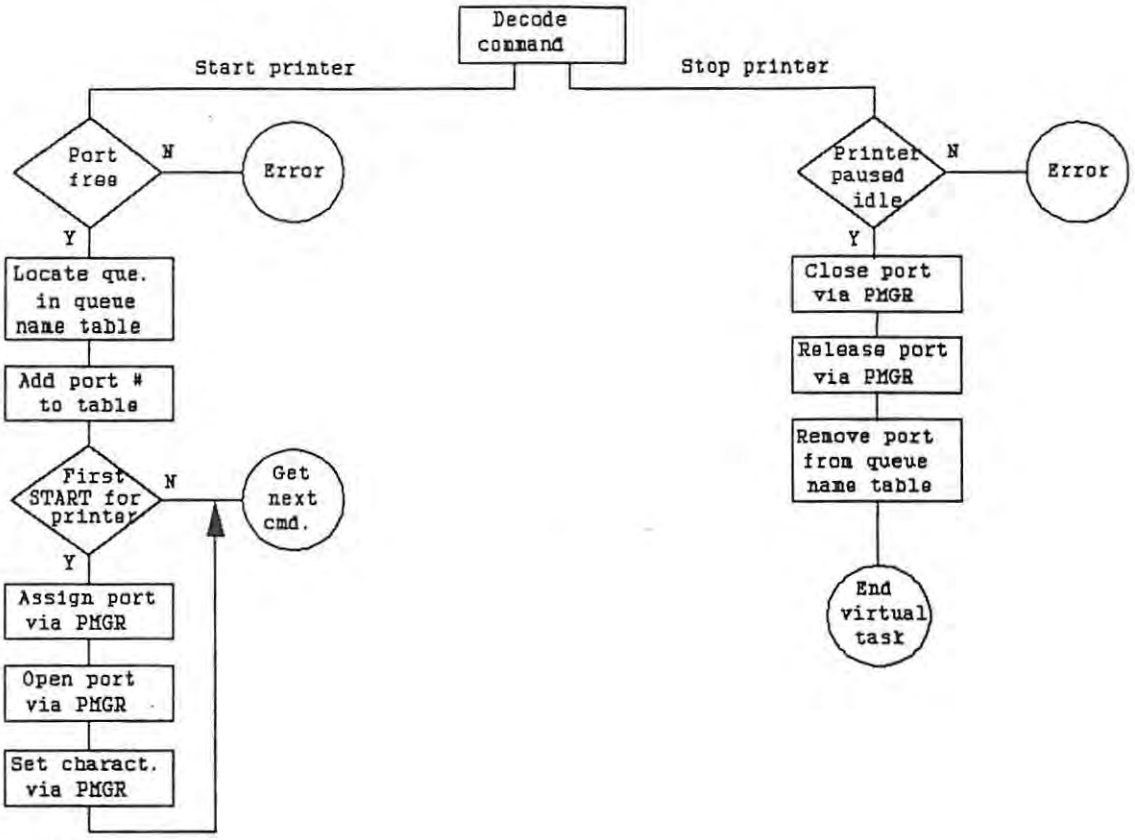
An array of queue names and console ports is maintained in the console data table which allows a many to many relationship to be constructed between queues and printers. When a START command is received, the port is checked to ensure that it is not already in use and if not then an entry is made in this array. If the queue already has printers allocated to it then this port is added to the list. Otherwise an entry is made for this queue and port. This information is used by the scheduling code block described in section 6.2.2.

If this is a subsequent START command for the same printer but to a different queue then processing stops here. If this is the first START for this printer then the port is then assigned and opened via PMGR. The port characteristics are then set for a print device. When the defaults have been received from the system manager and a CONTINUE command has been received then a branch is made to the scheduling code to start printing.

When a STOP command is received to de-activate this printer a test is made to ensure that the printer is paused and idle. If so then the console port is closed and released via PMGR and the port removed from the queue name array.

A flow chart of the above logic is shown in figure 6.4.

Figure 6.4 - Flow chart of the printer activation code block



6.2.2 Queue scheduling code block

The queue entries are held in a table in shared memory. This has a number of advantages. Firstly, AOS/VS always flushes shared memory pages to disk when a program terminates, even in the case of program failure. This means that no program disk I/O need be done to access the print queues. If the machine has spare memory then no disk I/O need be done at all to access the queues.

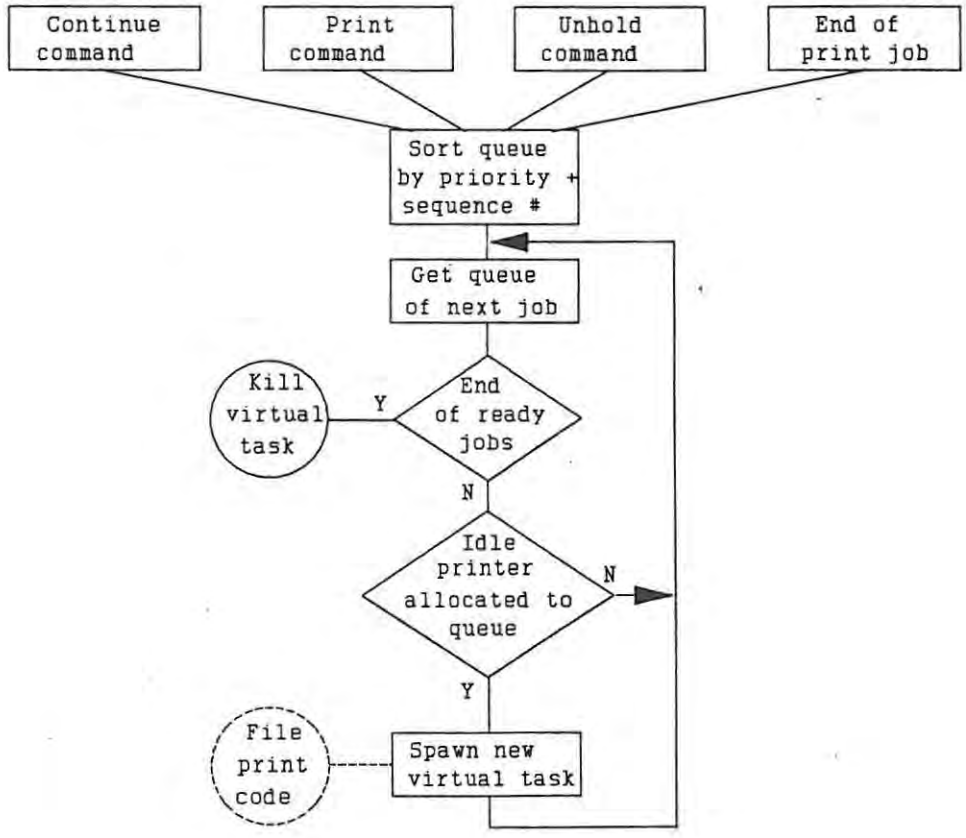
Secondly scanning and sorting the queue for scheduling can be done quickly in memory. Finally shared memory allows all subroutines within MCP as well as other processes that wish to interrogate the queues to do so.

The scheduling code block is executed under the following conditions; a printer is "continued", a file is submitted for printing, a print job completes and finally a user "unholds" a queue entry.

Firstly the jobs that are ready for printing (ie no hold flag and not currently printing) are sorted by priority followed by sequence number. This guarantees that jobs with the highest priority are printed first and jobs with the same priority are printed on a first come first serve basis.

An attempt is now made to match a print job with an idle printer allocated to that queue. For every match a virtual task is spawned which executes the file print code block discussed in section 6.3. This is an example of the type of task management used in PMGR.

Figure 6.5 - Flow chart of the queue scheduling code block

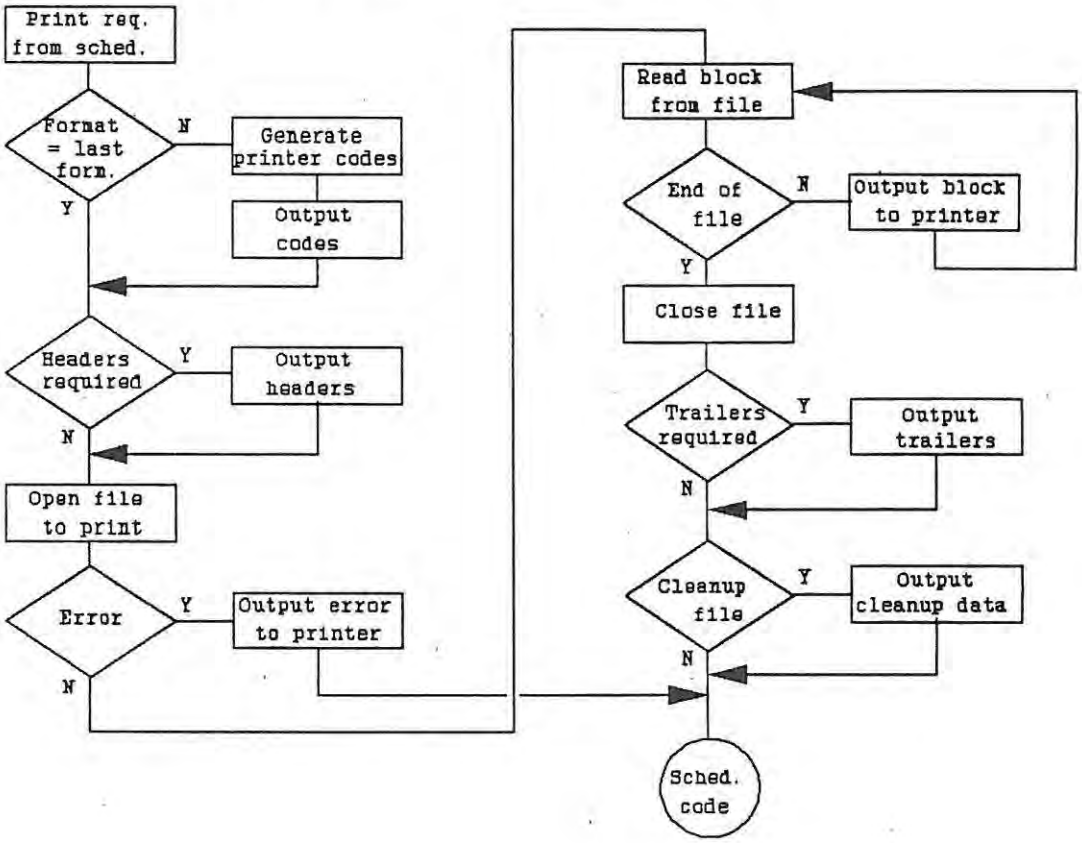


This implies that in the case of multiple printers for one queue, many jobs in a queue will be printed simultaneously. In the case of one printer processing multiple queues the jobs will be printed according to priority and sequence number. In other words MCP treats the queues as though they were one. Figure 6.5 shows a flow chart of this logic.

6.2.3 File print code block

This code block outputs files to the printer. First the format file is compared to that used in the last printout. If different then, by merging it with the device file, a new set of codes are generated and then output in order to program the printer.

Figure 6.6 Flow chart of the file print code block



Once the printer is set up, the header(s) are output if required. The file to be printed is then opened. If an error is encountered then this is sent to the printer for the user's information. If the file was opened successfully then the data is read block by block and output to the printer.

Once end of file is encountered, the trailer(s) are output if required and the file closed. If a clean up file is specified then it is output. After this the queue entry is deleted and a branch made to the scheduling code block to get the next queue entry for printing. Figure 6.6 shows a flow chart of this logic.

6.2.4 Command processing code block

This code block is mainly used to check the validity of commands from the system manager and users. If valid then the appropriate variables are updated.

All queue commands (open, close, create, delete and purge) result in a similar command being sent to EXEC. If the EXEC command is not executed successfully then MCP will reject the command. This is to ensure that EXEC and MCP queues always remain in synchronization.

6.2.5 Physical tasks

Five tasks are used by the print environment, namely; the main task, MCPREAD, MCPNET, MCPPRT and PRTDISP. MCPREAD and MCPNET are standalone tasks and have already been covered in the introduction to chapter 6, hence will not be discussed here.

All disk I/O and slow code paths are located in MCPPRT and PRTDISP. This is to ensure that the main task can service the IPC port quickly enough to give fast response times to users.

The printer activation code block described in 6.6.1 requires no disk access but does require a large amount of traffic to PMGR and thus has been located in the main task. All the other code paths with the exception of the command

processing code are disk intensive and have thus been placed in MCPPRT.

The PRTDISP task is responsible for displaying data to the user and has been created for the following reason. Information is sent to a user's console via the ?SEND system call. AOS/VS only provides enough memory for fifty messages per console. Displaying of a large print queue or spoolstatus for all printers could well result in more than fifty messages being sent to the user's console. When this happens then the extra messages are simply discarded. To prevent this the sending program must pause regularly to allow the AOS/VS buffers to clear. If this code was in the MCPPRT task then while the pause was in progress no output to printers would be generated and poor printing response would result. This together with the fact that queue data is requested frequently by the users justifies the creation of the extra task.

As with the STARTCON task discussed in chapter 4, a queue is used to pass work from the main task to MCPPRT. This task in turn uses a queue to transfer work to PRTDISP. The console and queue tables are situated in shared memory common blocks and are thus accessible by all tasks. Data passed to the MCPPRT task in the queues includes:

a) Console number — needed as an index into the console data table.

b) Entry point — indicates the section of code to be executed.

c) Command buffer — holds the command received from the user.

d) User's PID — the process ID that sent the command. Needed to that the reply can be sent to the correct console.

Branching to the correct piece of code to process the contents of the queue is achieved by means of the entry point. There are seventeen entry points, namely:

ENTRY POINT	FUNCTION
1	Queue scheduling. This is the code block discussed in section 6.2.2.
2	Generate and output printer format code. This is the start of the print file code block discussed in section 6.2.3.
3	PMGR return from outputing format codes. Open file to be printed and output first part of header if required.
4	PMGR return from 3. Output second part of header.
5	PMGR return from 4. Output third part of header.
6	PMGR return from 5. Output fourth part of header.
7	Read a record from the file and output it until end of file is reached. PMGR return points back to 7.
8	Same as 7 above. Used for alternate error branch on return from PMGR in main task.
9	Close file and output cleanup file if required and first part of trailer if required.
10	PMGR return from 9. Output second part of trailer.
11	PMGR return from 10. Output third part of trailer
12	PMGR return from 11. Delete queue entry and file if /DELETE requested then branch to queue scheduling.
13	Spoolstatus requested - pass it on to PRTDISP.
14	Quiet point in printing reached. Force AOS/VS to flush print queue pages to disk.
15	Print command received. Check syntax and place in queue.
16	Queue display command received. Check syntax and pass it on to PRTDISP.
17	MCP termination command received. Flush queues to disk and stop all printers.

As has been mentioned the PRTDISP task displays print information to the user's terminals. It is called from MCPPRT which does all syntax checking of commands. Communications is also via a queue which contains the same information as the MCPPRT queue, namely; console number, entry point, command buffer and User's PID.

There are only two entry points into this task. One for displaying spoolstatus information in which case the data is formatted and output for each printer requested in the command with a one second delay between each printer. The second is for queue display in which case a three second delay is used after every twenty queue entries.

The AOS/VS task scheduling priorities given to the tasks are as follows. The MCPPRT task has a priority of 10. This task is disk and asyncronous I/O constrained and thus does not warrant a higher priority. Experience has shown that this priority is sufficient to drive all the printers simultaneously without printers pausing. The PRTDISP task also has a priority of 10 to give reasonable response to user requests for queue data. The MCPREAD task has a priority of 12. This is the second lowest in the system. If transfer of files from the EXEC queues is delayed by a few seconds due to activity in the other tasks no real functionality is lost. MCPNET has a priority of 30 and shares the lowest priority with the BATSIG task (see the batch processing environment in chapter 7).

7. The MCP batch processing environment

One of the most critical responsibilities of a system manager is to control machine capacity utilisation and to maintain fast response for real time users. In a paper entitled "A comparative study of system response time on program developer productivity" by G Lambert [LAM84] the differences in user productivity were measured for response times of 2.23 seconds and 0.84 seconds. With the shorter response times user productivity went up 64% and work output went up 58% without using extra resources. The paper argues that additional machine capacity is justified in the light of improved user productivity. If this is so, then if the existing capacity can be stretched to achieve the work throughput required without sacrificing response time for real time users, a significant saving of capital investment will result. This can be achieved with a flexible batch environment that has the ability to relieve the real time environment of most of the CPU intensive work and thus guarantee good response times for real time users.

Many of the ideas that have been incorporated into the MCP batch processing environment were developed as a result of the problems associated with controlling real time reporting. When the user is given a report writing tool, the system manager's control over capacity utilisation is severely reduced. Many users can compile and request large complex reports at once which will bring any system to its knees. Furthermore if too many reports are extracted from one data base simultaneously then, due to excessive disk seeks, the time taken to extract the data will be longer than if the reports were executed sequentially.

Process scheduling priority or class scheduling is not a solution to this problem as certain reports are high priority and must be extracted as soon as possible. It is not practical for the system manager to determine and control priorities for these reports.

The obvious solution to these problems is to resort to batch processing. There are two critical needs that must be met. Firstly the system manager must be able to dynamically control the amount of resources allocated to the batch environment without affecting the users (apart from turn around times for batch jobs). Secondly the user must have a mechanism to ensure that important batch jobs are processed quickly. Queue priority is not a solution for this need as there may already be a

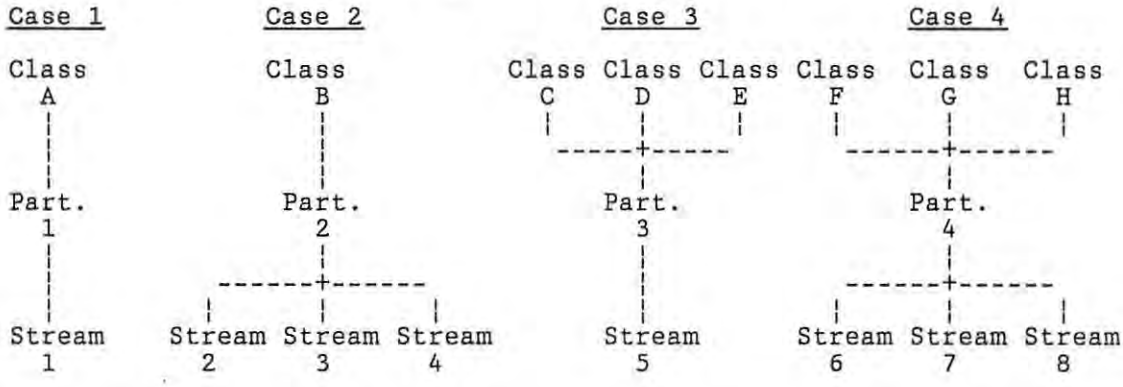
number of jobs queued by other users at a high priority that will be processed before this job can be processed thus not allowing the important job immediate access to resources. In other words users from different departments must be isolated. Users in one area can be expected to co-ordinate job priority between themselves but in a large company inter-departmental co-ordination is not practical.

One of the major problems with using batch processing when real time is available is to persuade the users to use it. It is more satisfying to watch a report being extracted and generated than to submit it to batch and receive a printout at some time in the future. The method used in MCP to overcome this difficulty is to charge a penalty for real time reporting in the charge out system discussed in chapter 8. Faced with a far higher cost for real time reporting the users soon resort to batch.

When viewed in the light of these needs the EXEC batch environment is found to be lacking. When the development of MCP started, EXEC offered one batch queue with four streams. Each stream could process a job. This meant that only four batch jobs could run simultaneously which is hopelessly inadequate in a large company if batch is to be used for reporting. The maximum number of streams has since been raised to two hundred but still processed from one queue. As has been pointed out one queue is not sufficient to meet the user's needs for high priority jobs in a large multi-department company.

All of this has lead to the following philosophy for the MCP batch processing environment. Firstly the batch system provides up to 26 classes. Each class corresponds to a queue. This is the user's view of the system as jobs are submitted to a class. The classes are grouped into partitions. Each partition can have from 1 to 26 classes assigned to it. Each partition can also have up to 200 streams allocated to it. Each stream can execute one job at a time. This is the system managers view of the system and allows him to allocate streams to each partition depending on machine utilisation without affecting the user (apart from job turn around time). In summary, the function of a partition is to isolate the system manager and user views of the batch system by creating a many to many relationship between classes and streams as shown by figure 7.1.

Figure 7.1 - Possible configurations for the MCP batch environment



Case 1 represents the trivial case of one class serviced by one stream. Case 2 corresponds to the environment provided by EXEC with one class (or queue) serviced by many streams. Case 3 allows for many classes serviced by one stream and case 4 for many classes serviced by many streams.

Normally two partitions are allocated to each department. One with a structure as in case 2 above for high priority jobs and one as in case 4 above for normal work. A class is given to each area for normal work which allows users in that area to view and control their own queue. This structure fully satisfies the needs expressed at the beginning of the chapter as the system manager can control system utilisation by varying the number of streams per partition. The user can ensure that an important job is run quickly by submitting it to his department's case 2 class and normal jobs are isolated by area thus allowing the users in that area to control their usage of the batch environment.

Scheduling of this environment is carried out as follows. In a single class environment when a stream becomes available the job with the highest priority is run next or in the case of jobs with equal priority, the first job to be submitted is run next. In a multi class environment streams are allocated to the classes in a round robin sequence. This ensures a fair allocation of resources to all classes in the partition. Scheduling within a class is the same as for a single class environment.

Further discussion on the batch environment is divided into three areas, namely functionality of the batch environment, security and program structure.

7.1 Functionality of the MCP batch environment

As has been intimated in the previous discussion, the MCP batch environment attempts to provide a flexible environment to support the real time environment of the DG computer. It is a complete departure from the environment created by EXEC which will not be mentioned further. The following discussion is divided into the user's and system manager's view of the batch environment.

7.1.1 Users view:

The user submits batch jobs to a class for execution via the BATCH command. Classes are protected by security and thus only certain classes are available to each user. The batch job consists of a file containing a number of CLI commands or a single CLI command line specified as arguments to the BATCH command. The options that are available as switches when issuing the BATCH command are as follows:

SWITCH	MEANING
CLASS=A to Z	This switch is the only mandatory switch and specifies the class in which the job is to run.
AFTER=run time	This switch allows the run time of the job to be specified. A number of formats are recognized. After=yy-mm-dd:hh:mm:ss allows the job to be run on a specific day at a specific time. After=hh:mm:ss means run the job today at specified time. After=+hhh means run the job at current time + hhh hours. For example AFTER=+24 would allow the job to be run one day later (default is ASAP).
INSERT	Take input for the job command file from subsequent lines entered from the console. Input of job commands is terminated with a single right parenthesis ")" and a NEW LINE. If this switch is used then arguments to the command are ignored (default is get job commands from arguments

to the command).

JOB=job name — Gives this job a name. This name will be displayed in MCPBATCH queue queries (default is no name - only sequence number).

LIST=file — Set the generic list file of the batch process to "file" (the default is "@NULL" ie discard all list output).

NOTIFY=file — Cause MCPBATCH to send a message to the file or device specified in "file" when the job is completed. If "=file" is omitted then the message will be automatically sent to the terminal (default is do not notify).

OUTPUT=file — Set the generic output file of the batch process to "file". This is normally set to a printer so that results can be viewed (default is the system printer in the computer room).

PRIORITY=nnn — Give this job queue priority "nnn" (0<n<256). Jobs in a class will be scheduled according to this value (default priority is 127).

When the user's job has completed a printout of the job commands, status and output generated is sent to the printer or file specified by the OUTPUT= switch. If the NOTIFY switch was used then a message will be sent to his console to indicate that the job is complete.

A number of other commands can be issued by the user to control the running of his jobs. Only that portion of the command needed to identify it uniquely need be entered. Commands generally have more than one format and the following conventions are used to indicate the syntax:

CAPITALS — This is a key word and should be entered as such, but abbreviations are allowed.

small letters	This implies input from the user with the indicated meaning.
[ ]	Contents of the brackets are optional and if omitted a default will be assumed.
{ }	These brackets indicate a choice of two or more options.

User commands available are as follows :

a) DISPLAY {CLASS} [{class}]
{USER } [{user name}]

This command allows the user to display information on jobs queued. By selecting {CLASS} a report on entries for a particular class can be obtained. By selecting {USER} a report for a particular user can be obtained. If {class} or {user name} are left out all entries in the queue will be displayed.

b) STATUS {CLASS } [partition]
{STREAM}

This command allows the user to view the structure of the batch environment. By selecting {CLASS} a report on classes assigned to the specified partition can be obtained. By selecting {STREAM} the streams assigned to the specified partition will be reported. If [partition] is left out then information for all partitions will be displayed.

c) CANCEL seq no.

This command allows the user to cancel a previously batched job. If a job has not yet started running it will be instantly removed. If a job is running then it will be terminated and then removed. Note that unless the user is operator he may only cancel his own jobs.

d) HOLD seq no.

This command allows the user to hold a previously batched job for later execution. Note that unless the user is the operator he may only hold his own jobs. A job can be held by both the user and the operator in which case both must unhold the job before execution will continue. If the job was already active when the hold request was received then execution will be abandoned and the job will be held in the queue.

e) UNHOLD seq no.

This command allows the user to unhold a previously held job. Note that unless the user is operator he may only unhold his own jobs. A job can be held by both the user and the operator in which case both must unhold the job before execution can start.

7.1.2 System manager view:

The system manager is responsible for setting up the batch environment. This involves assigning classes to partitions and specifying the number of streams to service each partition. In addition class security attributes must be specified.

Each class in MCPBATCH can have a number of attributes associated with it. These attributes will determine the security environment for jobs running under this class. Class attributes are as follows :

ATTRIBUTE	MEANING
Process type	This specifies the memory paging type for processes run in this class. AOS/VS process types are resident, pre-emptible and swapable.
Process priority	The AOS/VS scheduling priority for processes running in this class.

Privilege mask — The AOS/VS security privileges (see section 2.6.2) that jobs in this class will receive will be a mask of these privileges with the user's privileges (see section 7.2 for more detail).

Access template — Only users whose user names match this template will be allowed to submit jobs to this class.

Stop time — At the specified time this class will stop executing new jobs. Current jobs will still be allowed to finish.

Start time — At the specified time this class will start executing jobs again.

The commands available for the system manager to control the batch environment are as follows:

a) ASSIGN {CLASS partition,class}
{STREAM partition,stream}

This command allows classes and streams to be assigned to partitions. By selecting {CLASS} a class may be assigned to a partition. This process is repeated until all the required classes have been assigned. By selecting {STREAM} streams may be assigned to a partition. A class or stream may be de-assigned from a partition by assigning that class or stream to partition "0".

b) SET {TYPE class,{S}}
{P}
{R}
{PRIORITY class,n}
{PRIVILEGE class,{H}}
{M}
{L}
{ACCESS class,access template}
{STOP class,hh:mm}

{START class,hh:mm}

This command allows the characteristics to be set for a particular class. By selecting {TYPE} a class may be set to Swapable, Pre-emptible or Resident. By selecting {PRIORITY} the process priority of a class may be set. By selecting {PRIVILEGE} the maximum allowable privilege for a class may be set. By selecting {ACCESS} an access template for the class may be entered. By selecting {STOP} the time this class will stop processing jobs can be entered in the format hours:minutes. By selecting {START} the time this class will again start processing jobs can be entered.

c) PAUSE {CLASS class} [,A]
{PARTITION partition} [,A]

This command allows the operator to stop job scheduling for a class or partition. By selecting {CLASS} a class may be paused. By selecting {PARTITION} an entire partition may be paused. If jobs are running in the partition / class to be paused and ",A" is not specified then MCPBATCH will notify the operator and will pause when these jobs are completed. If ",A" is specified then the jobs will be aborted, the partition / class will pause immediately and the jobs re-queued for later execution (unless the job was batched with /R=N).

d) CONTINUE {CLASS class}
{PARTITION partition}

This command allows the operator to continue a class or partition that has been paused. By selecting {CLASS} a class may be continued. By selecting {PARTITION} an entire partition may be continued.

e) FLUSH {STREAM stream}
{CLASS class}
{PARTITION partition}

This command allows the operator to flush a stream, class or partition. By selecting {STREAM} a stream may be flushed. By selecting {CLASS} an entire class may be flushed. By selecting {PARTITION} an entire partition may be flushed.

f) VERBOSE class

This command allows the operator to set a class to verbose reporting. This means that detailed information on each job started and completed is displayed on the master console. If "class" is omitted then all classes will be assumed.

g) BRIEF class

This command allows the operator to set a class to brief reporting. This means that simple messages will be sent to the master console on start of batch jobs. If "class" is omitted then all classes will be assumed.

h) SILENCE class

This command allows the operator to inhibit all reporting to the master console for a class. If "class" is omitted then all classes will be assumed. This is the default setting for operator messages.

i) UNSILENCE class

This command allows the operator to activate reporting to the master console for a class. If "class" is omitted then all classes will be assumed.

7.2 Security

As users can run any job under the batch environment, security is an important issue. Firstly a user can only use the batch environment if his user name has access to a class. Assigning user access to classes is restricted to the system

manager only, thus guaranteeing batch class integrity and preventing users from one department from using the batch resources allocated to another department.

As discussed in section 2.6 AOS/VS maintains a 16 bit privilege word for each process. Each class has a 16 bit privilege word associated with it. This is set by the system manager. When a user batches a job, MCP obtains its privilege word and stores it in the batch job queue. When the job is started the class privilege word is logically "anded" with the user's privilege word. The batch job is started with the resultant privilege word. This means that the batch job will get only those privileges granted to both the class and the user. This ensures that the user will never obtain more privileges in batch than he has in real time.

The working directory, search list and default ACL are all set to the same as the user's real time settings at the time of batching the job. This creates the correct environment for the job to run and also ensures that AOS/VS disk security is not breached.

Process priority and type are set to those of the class irrespective of the user's settings. This prevents the user from obtaining a higher AOS/VS scheduling priority than that set by the system manager.

The system manager can also use the start and stop commands to prevent jobs from running during certain times, for example during backups.

Apart from the above restrictions the job runs as though it was a real time program with the exception that input is taken from the job commands file and output is sent to the printer or file specified by the OUTPUT= switch rather than to a terminal.

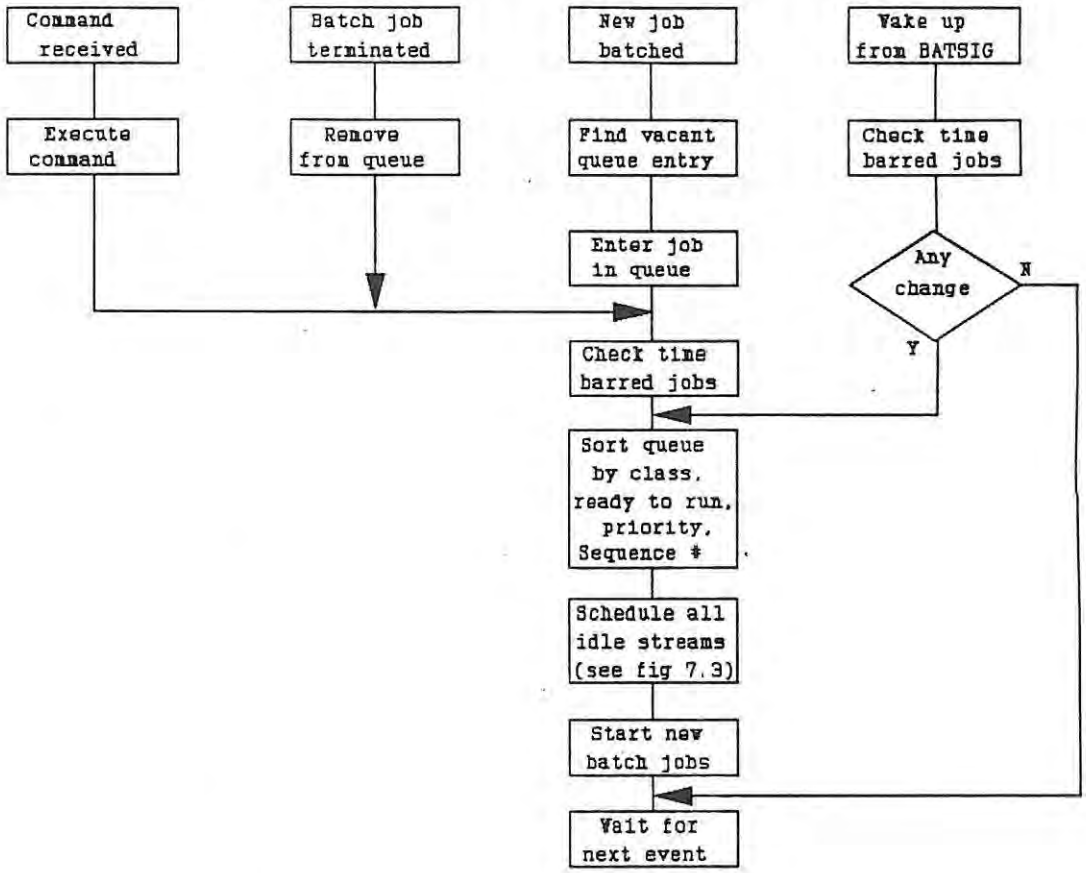
7.3 Program structure

The batch environment is supported by the main task, MCPBATCH and BATSIG. The role of these tasks in the batch environment is discussed in more detail in section 7.3.2. Unlike the other areas in MCP the batch environment does not make use of virtual tasks. As no PMGR interface is required, virtual tasks are not necessary. Instead the system is event driven. The main events are the batching of a new job,

the termination of a running job, a wake up signal from the BATSIG task and the receipt of one of the other commands covered in section 7.2.

These commands are fairly trivial and either update MCP's data arrays or cause the displaying of information on a user's console. Attention will therefor be focused on the first three events. A flow chart of their interrelation is shown in figure 7.2.

Figure 7.2 - Flow chart of the MCP batch system

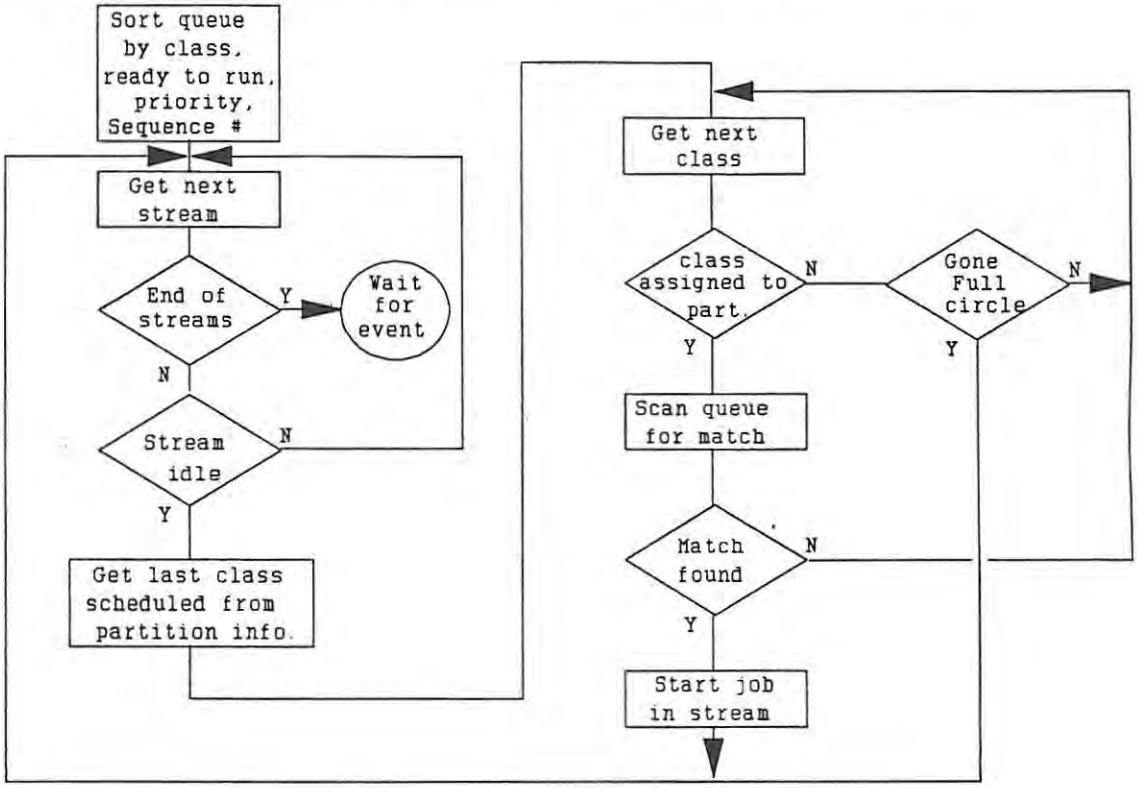


The following discussion is divided into two sections, namely; class scheduling code block and physical tasks.

7.3.1 Class scheduling code block

The batch scheduling algorithm must cater for a variable number of classes and streams per partition. As has been mentioned each stream can execute one job at a time. The allocation of streams within a class is ranked by priority and then "first in first out" for jobs of the same priority. When more than one class is assigned to a partition then streams are allocated on a round robin basis to the classes.

Figure 7.3 - Flow chart of the MCP batch scheduling system



The scheduling algorithm works as follows. On entry to the scheduling code block all jobs that are ready for execution are sorted by class, priority and sequence number. The next idle stream is then located. If no more idle streams are found then the program suspends pending the next event. If an idle stream is found then the last class to which a stream was allocated is identified in the partition to which the idle stream belongs. The next

class allocated to this partition is then scanned for jobs that are ready for execution. If no job is ready the next class is scanned. If a job is found for execution then the job is started in this stream and the next idle stream is located. See figure 7.3.

7.3.2 Physical tasks

As has been mentioned under the printer environment in chapter 6, all job queue, class, partition and stream data is held in tables in shared memory. This means that AOS/VS will page the information to disk thus saving MCP the overhead of disk queue management. This also facilitates fast queue sorting and access for scheduling and reporting to the user. After the scheduling code block has been executed and new jobs have been started, a request is sent to AOS/VS to flush modified pages to disk. This insures that if the machine should go down unexpectedly due to power or hardware failure then the data in shared memory will be saved.

The main task plays a very minor role in the batch system in that it only relays messages destined for the batch system to the task queue for MCPBATCH. All further processing takes place in the MCPBATCH task. The queue for communicating between the main task and MCPBATCH contains the following information:

a) Process ID of process sending command

b) User name of process sending command

c) The command sent

d) Type of command (either sent by user, wake up or batch job termination).

The MCPBATCH task contains the majority of the code used for the batch system. This includes the scheduling code, the job queuing code, security control, system manager command processing and display of information to the user. The same constraints discussed in section 6.2.5 apply to the

displaying of information on user consoles but due to the low activity and non critical nature of the batch environment no display task is considered necessary. The reason for the low activity of the batch environment is that all user jobs are run as son processes and thus do not take resources from MCP.

The BATSIG task sends a wake up signal to the batch environment once a minute. Since the batch system is only activated on receipt of an event, if a wake up was not generated by BATSIG then the system would only be activated when a new job was submitted, when a job terminated or when another command was given by a user. This means that no mechanism would exist for starting jobs that had been queued for running at a specific time. The wakeup signal from BATSIG causes a re-schedule and thus ensures that the jobs are run at the correct time.

The AOS/VS task scheduling priority given to the tasks in the batch environment is as follows. The MCPBATCH task has a priority of 10. This is relatively low due to the low activity and non-critical nature of the batch environment in terms of response. BATSIG has a priority of 30. This is the lowest priority on the system as no real functionality is lost if the signal is delayed by a few seconds due to activity in the rest of MCP.

8. The MCP accounting environment

The traditional method of charging for computer resources has been to allocate costs to an overhead cost centre. In a batch environment this is satisfactory as the user has no involvement in running programs. However in a real time environment the situation changes radically. If no direct charge out of resources is implemented then the user is likely to overuse the resources by running programs of questionable worth. Two methods of handling computer costs are normally used in a real time environment. The first is direct charge out. In this instance the costs of the computer department are charged out to the users based on usage of resources. Any decision to expand the DP resources is taken by top management. The second is the profit centre approach. In this instance the computer department operates as a separate company that charges for its service and uses the profits to procure new hardware and growth for the department. In both cases a mechanism for calculating the costs to be charged to each user is needed. This is an issue that is frequently ignored in operating systems and is certainly not catered for in AOS/VS. This chapter discusses the facility provided by MCP to meet this need.

The paper by L McKell, J Hansen and L Heitger entitled "Charging for Computing Resources" [MCK79] discusses some of the mechanisms used for charging for resources. Two philosophies for charging are discussed in this paper.

The first is direct charge out with exact recovery of costs every month. The major weakness of this approach is that rates are adjusted to ensure exact recovery. This means that if a user takes great pains to minimize his usage in a period and other users happen not to use the machine heavily during that period then his portion of the costs will actually go up. This will not be perceived to be fair by the users and will generate negativity towards the system and the computer department.

The second is a pricing model with the rate fixed at a value that will give approximately the return required to cover costs, while at the same time allowing the user to achieve cost savings through effective machine usage. This is the philosophy adopted in the MCP accounting system.

In his paper entitled the pricing dilemma [H0069], J Hootman proposes the following principals.

a) Resource utilitisation. A reasonable load should generate sufficient revenue to meet goals.

b) User control. User charging rates should be applied to resources over which he has direct control.

c) Demurrage. A user should be charged for resources made unavailable to others as a result of his requirements, even though he was not using the resources directly.

   This point could easily conflict with point b) above as resources which are made unavailable to others due to his use are often a result of system weaknesses outside the user's control. This principal has therefore been discarded in terms of the MCP implementation of charge out.

d) Understandability. The user should be able to understand and use the charging structure in evaluation and estimation.

A number of additional principals not stated in the above paper but relevant to the AOS/VS environment have also been included.

e) Penalty charges for inefficient use.

   One of the major objectives of a charging system should be to encourage efficient use of the system and thus maximize the company's return on investment on computer resources. An example of inefficient use is to access data in a data base via the wrong index. In this case 5000 records might be read to get 10 records of useful information whereas if the correct index had been used then only 10 records would have been read. The impact of this on machine capacity is enormous. The problem with penalty charges is the determination of inefficient use.

f) Charging for shared resources.

In the AOS/VS environment there are several processes such as data base controllers that are used by a number of users and consume large amounts of resources in relation to the resources used by the user process.

g) Charging for development resources directly attributable to a group of users.

Charging for maintenance and development resources is one of the most contentious areas in a charge out system. A perception of fairness is the key factor. The type of issue causing acrimony is the charging of all users for programs developed for a single user.

h) Capacity monitoring.

One of the key requirements for effective planning of future capacity growth is a knowledge of current capacity utilisation and trends. This allows computer department management to plan pro-actively rather than reactively. As has been pointed out in chapter 7, the costs of poor response time in terms of user productivity are high and can be avoided by effective capacity planning. Since all the data needed for capacity management are collected as part of the calculation of charge out it makes sense to incorporate this facility in the accounting system.

The following discussion details the Implementation of these principals in the MCP accounting system. The discussion is divided into three areas, namely; charge out method, capacity control and program structure.

8.1 Charge out method

Charges for computer resources are allocated to cost centres consisting of a three digit number. The charges are divided into three major categories, namely; fixed costs, peripheral costs and variable costs.

a) Fixed costs.

This is a fixed amount charged to a cost center for a month. The amount is entered by the system manager and simply reflected as such in the user's account. The purpose of this cost is to charge a cost center for work done for them that is not reflected in the normal computer usage. For example development of a new system. If the system is for use by more than one cost centre then the portion to be absorbed by each cost centre would be negotiated beforehand.

b) Peripheral costs.

This is a fixed amount charged to a cost centre based on the peripherals in use in that area. The actual peripherals are accessed from the MCPLOCATIONS file and a cost allocated for each peripheral based on the monthly lease cost for the terminal itself, the DG port and the communications to link the two. As the MCPLOCATIONS file has to be kept up to date for the day to day running of MCP the peripheral charges will always be correctly apportioned.

c) Variable costs.

This is the area in which the majority of the costs are incurred. It is designed to meet the principals mentioned in the introduction to this chapter. Users are assigned to cost centers via a cost centre file maintained by the system manager. This file contains every user name on the system, the cost center to which it is allocated and the name of the system that it executes. In general the equation used to calculate variable costs per user is as follows:

Cost = Rate x (resource used + allocation of group resource
+ system resource allocation + development and maintenance
resource allocation)

The equation is explained as follows:

i) The rate for each resource is kept in a file that may be changed by the system manager after negotiations with the users. The rate is set

to comply with principal a).

ii) The resources that are monitored and charged are CPU, disk I/O, memory, elapsed time, disk space and tape usage. Many charging systems also charge for pages printed but as the printers are situated at user locations in a real time environment and paper is therefor already paid for by the users, this represents double accounting.

iii) The allocation of group resource refers to processes that are used by more than one user but not by all users such as a data base controller for a particular database. The allocation is calculated as follows:

$$\text{group resource allocated to this user} = \frac{\text{shared process resource} \times \text{user resource}}{\text{total resource for all users in group}}$$

This calculation is performed every time the machine activity is scanned (every minute) which gives the fairest allocation to each user possible because only those users using the shared resource in that time interval are charged. The users in a group are identified from the system name in the cost centre file. A shared process is identified by having a user name equal to the system name in the cost centre file.

iv) The system resource allocation refers to the system processes that are run to support all users for example MCP itself. The allocation is calculated in a similar fashion to group resources above, namely:

$$\text{system resource allocated to user} = \frac{\text{system resource} \times \text{user resource}}{\text{total resource for all users}}$$

A system resource is identified by the reserved system name GLOBAL in the cost centre file.

v) The development and maintenance resource allocation refers to work done by programming staff on existing systems. The allocation is calculated as follows:

dev. resource allocated to user = $\frac{\text{programmer resource x user resource}}{\text{total resource for all system users}}$

This allocation is done on a monthly basis when the accounts are generated and results in each user paying a share of development and maintenance resources dependant on his use of the particular system in that month. A development or maintenance resource is identified by DEV followed by the system name in the cost centre file.

Note that no penalties for inefficient use are charged on allocated resources.

The calculation of the user resources used is made up of the actual resources used and the penalties incurred. A file of factors for each penalty is maintained by the system manager thus allowing the severity of the penalty in terms of usage to be varied depending on the capacity constraints being experienced at that time.

The calculation of the different resources varies and will be discussed separately.

i) CPU resource. This is measured in seconds of CPU time used and is calculated as follows:

CPU = [actual CPU + (real time reporting CPU x factor)
+ (batch reporting CPU x factor) + (CEO CPU x factor)]
x TOD factor x Peak period factor

This equation makes provision for penalties to be imposed for real time reporting, batch reporting (normally this is small to encourage use of batch - see discussion in chapter 7) and use of CEO[18].

In addition the entire resource may be scaled up or down by the Time Of Day factor and the peak period factor. The time of day factor

[18] Comprehensive Electronic Office. An office automation product supplied by Data General that is expensive in terms of CPU resources.

allows different charges to be imposed during working hours and at night. The peak period factor allows three peak periods to be identified and additional charges to be levied during these times. Both these factors are used to encourage users to smooth the load over the day.

ii) Disk I/O resource. This is measured in blocks of data read or written to disk and is calculated as follows:

DISK = [actual blocks + (reporting wastage x factor)]
x TOD factor x Peak period factor

This equation makes provision for penalties to be imposed for inefficient reporting. The reporting wastage is defined as records read minus records selected. This implies no penalty for reports that read exactly the needed records. The size of the penalty grows with the number of records read unnecessarily. The information is obtained by forcing PRESENT[19] to call a program that sends the information to the MCP accounting system. It is a powerful motivator for users to optimise their reporting programs. In Middelburg Steel and Alloys Steel division, when the accounting system was introduced, the capacity used for reporting dropped from 70 % to 40 % in three months. The Time Of Day and Peak period factors are the same as those discussed under the CPU equation.

iii) Memory resource. This is measured in pages of memory used per CPU second and is calculated as follows :

MEMORY = [actual pages + (physical page faults x
factor)] x TOD x Peak period factor

This equation makes provision for penalties to be imposed for programs using excessive memory. Physical page faults occur when the program's working set cannot fit into physical memory and the system is forced to swop pages from disk. This penalty is designed to control users of

[19] The Data General report writer

systems such as APL who can set huge working set sizes thereby slowing the system for all. The problem with charging for memory is that for normal program users it contravenes principle b) in that they have no control over its use. Furthermore if a shortage of memory exists on the system then all users will experience physical page faults and thus be penalised again in contravention of principle b). For these reasons the rate for memory is normally set very low with a high penalty for page faults to control APL users.

iv) Elapsed time. This is a measure of the idle time a user spends logged onto the system and is measured as follows:

Elapsed time = time logged on - (CPU seconds * factor)

The equation makes provision for penalising users who log onto the system and do nothing thereby consuming memory and scheduling resources needlessly. The factor is set so that the average work session results in no charge. If a user works very efficiently then a rebate could result from this equation.

v) Disk space. This is a measure of the disk space in blocks used to store the user's files. It has no penalty factors associated with it. The space is expressed in blocks and the rate set at the monthly lease costs per block of disk space.

vi) Tape usage. This is a measure of the number of tape restores done in a month. It is calculated as the number of tapes loaded for the user multiplied by the rate. The objective is to discourage the user from haphazard deletion and subsequent restoring of files. No charge is made for backup which is outside the user's control.

The raw data on CPU, disk I/O, memory etc is accumulated as follows. CPU, disk I/O and memory data are retrieved from AOS/VS. A ?GPID system call is used to obtain all active PIDs, a ?GUNM system call is used to obtain the user name of each PID, a ?GPRNM system call is used to get the program that is running (this allows PRESENT, BATCH PRESENT and CEO to be identified),

a ?PNAME system call is used to obtain the process name of each PID (this allows the system to be identified for data base controllers) and finally a ?PSTAT system call is used to get the CPU seconds, disk I/O, memory pages and physical page faults for each PID. These values are then subtracted from the values obtained one minute earlier to obtain the activity during this minute.

Information on reporting wastage is obtained from a sub-program started by PRESENT. Elapsed time is obtained from the MCP terminal code and tape usage from the tape restore CLI macros.

Every month the accounts are generated for each cost centre. These consist of a number of reports detailed below.

a) A cost centre summary showing the total fixed, peripheral and variable costs (broken down into the areas discussed above) for each user in that cost centre. This report is intended for user management to monitor costs in their area.

b) A peripheral breakdown showing the actual peripherals in the cost centre and the charges for each. This allows the user to check the accuracy of these charges.

c) A system summary report for each system used in that department. This report shows all the users of a system and the amount charged to each one. This allows users to see the split of charges over all the cost centres using a system. It assists in creating an impression of fairness.

d) A detailed penalty report for each user. This report shows the values for all the penalty items and the subsequent penalty costs incurred. It gives the user the information needed to take action to reduce the penalty costs.

e) A detailed reporting wastage report. This report shows each PRESENT program run by the user, how many times it was run, the records read,

records selected and waste percentage. It allows the user to identify the programs that need attention.

These reports ensure that principle d) is adhered to in that the user understands why he was charged, what he was charged for and how he can improve the situation.

8.2 Capacity control

As was discussed in principal h), control of capacity is a vital part of the computer management's responsibilities. Information on all the processes running on the system has to be obtained for charge out and can be utilised for capacity management as well with little extra overhead. The system stores summary information on average and maximum CPU, disk and memory usage for each half hour on an ongoing basis. This allows trends for the above mentioned resources to be monitored. The information stored is shown in figure 8.1. Note that for summary reporting one record is written for the total system.

If a problem is suspected then detailed reporting can be requested by issuing the following command to MCP:

SCAN DETAIL

In this case the above information is stored for each process running on the machine for each half hour. The user name, program and process name are also stored. This allows detailed analysis of capacity usage to be conducted and steps taken to control the major offenders. It does however result in considerably increased overhead to process and store all the information (approximately 3 % CPU on the half hour and large quantities of disk space) and is thus not intended for continual use. The command SCAN BRIEF switches the system back to summary mode. The information stored is shown in figure 8.1.

Figure 8.1 - Capacity data stored by MCP

DATA ITEM	MEANING
User Name	The user name of the process (detailed capacity monitoring only).
Program	The program being executed (detailed capacity monitoring only).
Process	The process name of the process (detailed capacity

	monitoring only).
Date	Format YYYYMMDD.
Time	The time expressed as a decimal fraction.
CPU	The CPU seconds used in the last half hour.
Average CPU	The average of the half hourly CPU values since the process was started.
Cumulative CPU	The cumulative average CPU used since the process started.
Maximum CPU	The highest CPU usage measured during the minute scan since the process was started.
I/O	The disk I/O blocks in the last half hour.
Average I/O	The average of the half hourly I/O values since the process was started.
Cumulative I/O	The cumulative average I/O used since the process was started.
PPM	The memory pages used per millisecond of CPU usage.
Average PPM	The average of the half hourly PPM values since the process was started.
Cumulative PPM	The cumulative average PPM used since the process was started.
Total PF	The total page faults that occurred during the half hour. This includes both Physical and logical page faults. Physical page faults indicate swapping to disk, logical page faults occur in memory only.
Average TPF	The average of the half hourly TPF values since the process was started.
Cumulative TPF	The cumulative average TPF since the process was started.
Disk PF	The disk (ie physical) page faults that occurred in the last half hour.
Average PF	The average of the half hourly DPF values since the process was started.
Cumulative DPF	The cumulative average DPF since the process was started.

This information is stored in a format readable by PRESENT which allows the system manager to extract and format data at will.

In addition to this facility a number of standard graphs are produced monthly to assist with the early detection of capacity problems. Figures 8.2 to 8.10 are examples of these graphs and show the actual data accumulated during the first six months that this system ran in the Steel Division of MS&A. They show clearly the effect of the accounting system on the utilisation of the machine.

From figures 8.2 and 8.3 it is evident that the cost to users has dropped significantly. As can be seen from graph 8.5 the drop in month 6 was mainly due to an improvement in efficiency of reports. This is also reflected in graph 8.4 which shows the sharp decline in reporting CPU used. As can be seen from graphs 8.4 and 8.6 the decline in costs in months 7 and 8 was due mainly to the use of batch reporting and night time usage.

Figure 8.2 - Graph of total variable costs

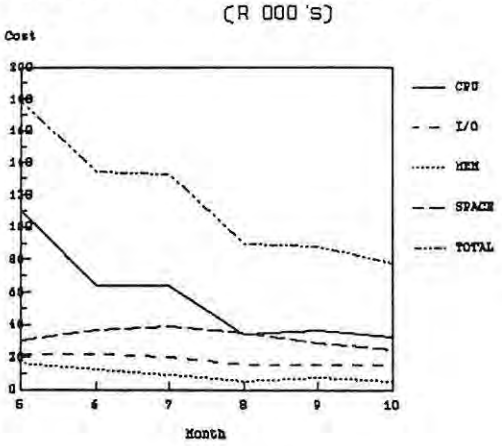


Figure 8.3 - Graph of variable costs per system

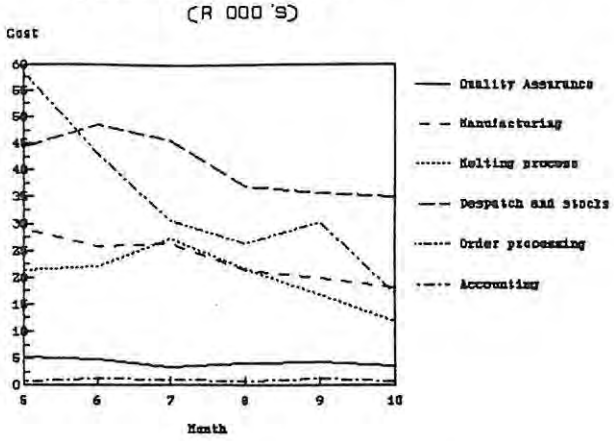


Figure 8.4 - Graph of batch vs real time reporting

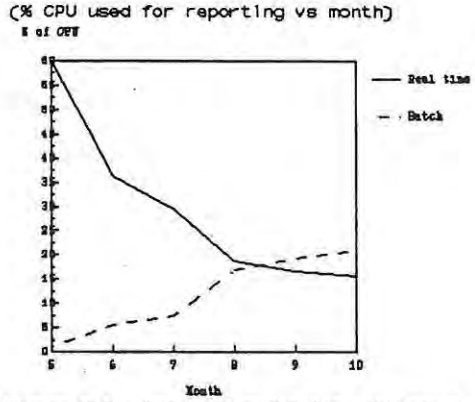


Figure 8.5 - Graph of wasted I/O for reporting

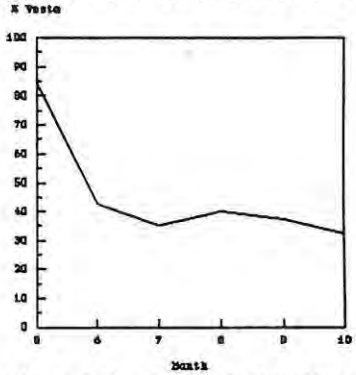


Figure 8.6 - Graph of night time CPU usage

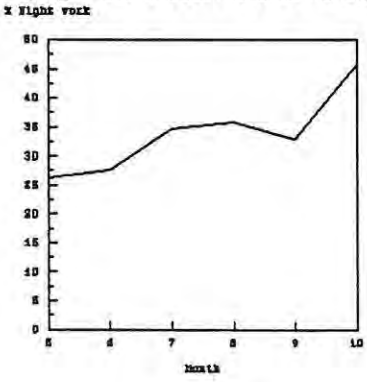
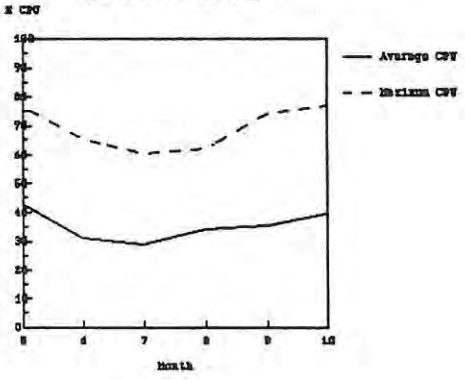


Figure 8.7 - Graph of Capacity trends



An interesting trend is shown in graph 8.7. The capacity utilisation, after an initial drop, is steadily increasing whereas the costs are still dropping. This indicates a more efficient usage of resources. This has in fact been observed with a noticeable improvement in system response in the face of a steadily increasing work load.

Graphs 8.8 and 8.9 show the capacity usage by day in October. The under-utilisation of the machine on weekends is evident. From these graphs the periods of maximum load and poor response can be seen. Graph 8.10 shows the average load over a day in October. It is clear that the machine is under utilised between 3 am and 7 am and between 5 pm and 9 pm. If capacity were to become seriously constrained then users would be encouraged to use these periods for reporting by means of the peak period penalty.

Figure 8.8 - Graph of average CPU

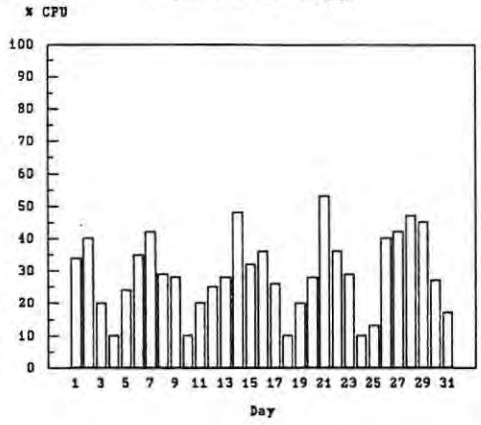


Figure 8.9 - Graph of maximun CPU

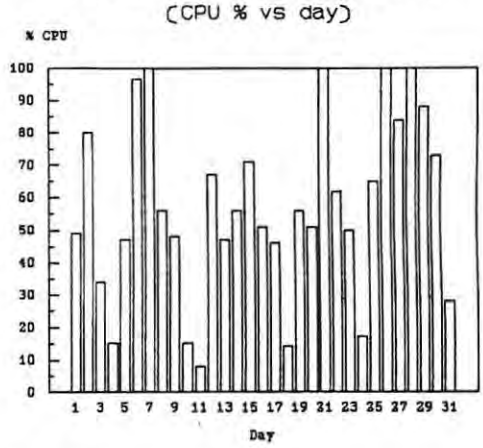
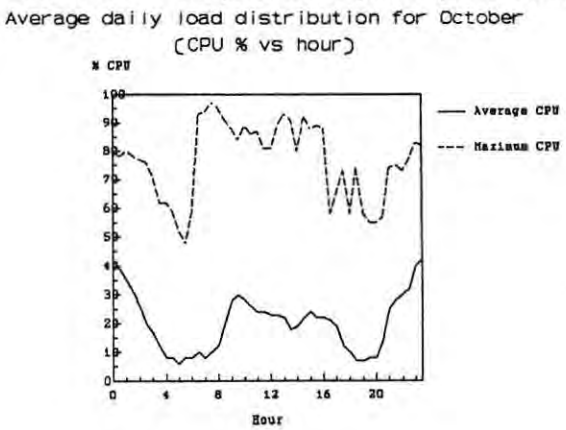


Figure 8.10 - Graph of average daily load distribution



8.3 Program structure

The MCP accounting system exists as a separate process to MCP. This has been done for one main reason, namely the accounting process uses large amounts of disk, especially when detailed capacity monitoring is requested. It is thus quite possible for the directory in which the information is stored to become full thus causing the accounting system to fail. If it were a subordinate task to MCP then the entire MCP system would fail with it. This is not considered acceptable and thus it is started as a subordinate process. MCP controls this process entirely and thus the system manager still sees one system.

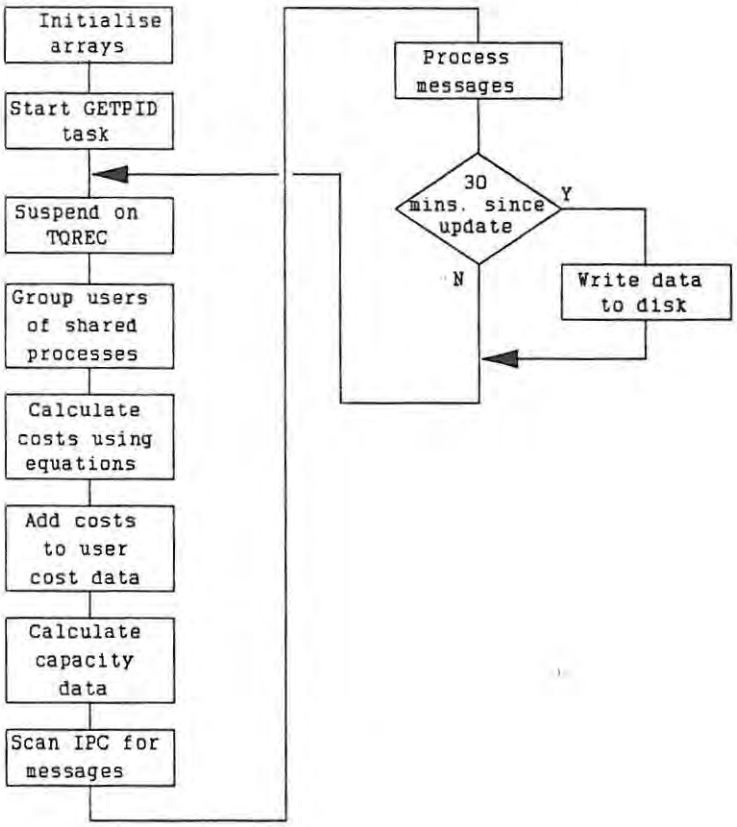
Communications with the accounting process take place through an IPC port. Through this port it receives commands from MCP, Terminal connect times from MCP, reporting wastage from the various PRESENT processes and tape restore information from the restore macro run by the operator. Commands sent by MCP include, brief capacity monitoring, detailed capacity monitoring and stop processing. All other information necessary to monitor the processes is received direct from AOS/VS.

The process has two tasks, the main task which does most of the processing and a GETPID task which retrieves process information from AOS/VS. The information is placed in shared common blocks for communications between the tasks. Synchronization is achieved by means of the task communication calls TQREC and TQXMT. It is essential that the process information is obtained from AOS/VS in exactly one minute intervals. If the interval is not constant then the capacity monitoring data will not be accurate. This is the reason for using a separate task to retrieve this information.

The flow of the main task can be summarised as follows. On starting, the arrays are initialised and the GETPID task started. The main task then suspends on a TQREC to the GETPID task. Once process data has been retrieved from AOS/VS then the utilisation of each shared process by the users of these processes is calculated, the charge equations applied and the resultant costs added to the costing data for each user. The capacity information is then calculated and written to the capacity files. The IPC port is then scanned using the "do not wait" option and all messages sent to the accounting process in the last minute

received and processed. Every thirty minutes the information is written to disk to insure against loss of data due to system failure. The task then suspends on a TQREC for the next set of process data from the GETPID task. This logic is shown in the flow chart in figure 8.11.

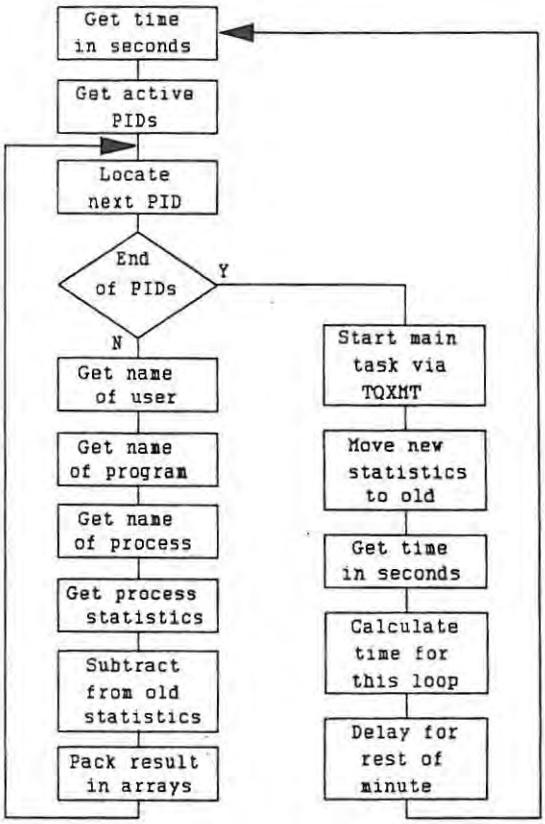
Figure 8.11 - Flow chart of the main accounting task



The GETPID task executes the following loop. First the time is retrieved from AOS/VS in seconds. Then all the active processes are obtained from AOS/VS. For each PID the user name, program name, process name and process statistics are obtained. The process statistics are subtracted from those obtained in the last loop to get activity during this minute. This information is then packed into the shared memory arrays. The main task is then activated via a TQXMT task communications call. The time is again obtained from AOS/VS and the first time subtracted from it. This gives the time taken to execute this loop which is subtracted from 60 to determine the number of seconds to wait before re-executing

the loop. This process guarantees that the loop is executed exactly every 60 seconds. This logic is shown in the flow chart in figure 8.12.

Figure 8.12 - Flow chart of the GETPID task



9. The system manager's toolbox

As has been intimated in chapters 4 to 8, the system manager has a major role to play in the effective running of MCP. It is therefore important that an effective, easy to use, toolbox be provided for him. This chapter discusses briefly the tools that have been developed to support MCP. The majority have been grouped together under one system called MCPSMI[20].

The rest of the discussion in this chapter is divided into additional MCP commands for the system manager, log files maintained by MCP, emergency shutdown procedures and MCPSMI.

9.1 Additional MCP commands for the system manager

There are a few extra commands provided by MCP for the system manager that have not been discussed in the previous chapters. These are:

COMMAND	MEANING
READ	Read the contents of the MCPLOCATIONS file into memory. This is used to implement changes made to the file.
DIE	This command brings MCP down. MCP will terminate and disable all terminals and PCs, pause and stop all printers and batch streams, stop the accounting system and kill itself. This will cause AOS/VS to flush all shared memory pages to disk which will ensure that all the queues and console port arrays are in a consistent state when the system is re-started.
BROADCAST template,mess	This command allows the system manager to send a message to either a group of users specified

[20] Stands for MCP System Manager's Interface.

by the template or to all users if a template of "+" is specified. It is used to warn users of a shut down or to inform them of an important event.

LOGGING [ON or OFF]

This command allows logging of IPC messages to MCP to be switched on or off. If logging is on then every message from PMGR, remote MCPs, the system manager, users and any other source is written to disk. This drastically reduces the speed of MCP but is invaluable for debugging of the system.

SCAN [ON or OFF or BRIEF or DETAIL]

This command allows the accounting process described in chapter 8 to be controlled. The ON and OFF arguments allow the accounting process to be started or stopped. MCP automatically starts the accounting system as part of its initialisation. The BRIEF and DETAIL commands cater for brief and detailed capacity monitoring.

MESSAGES [ON or OFF]

This command allows all messages from MCP to the master console to be switched off or on. The terminal, PC, printer and batch environments all send messages to the master console on activity being performed. These messages can be switched off for each environment or device using the SILENCE command. This command provides a blanket silence facility. It is used when work is to be done on the master console that should not be interrupted.

HELP

This command provides the system manager with a summary of all commands available for each

environment.

9.2 Log files maintained by MCP

MCP maintains four log files. These are as follows:

a) MCPSCANLOG. This file is produced by the accounting process. It informs the system manager of any errors encountered. The most important information contained in this file are user names that are encountered that do not have an entry in the cost centre file. This allows the system manager to keep the cost centre file accurate.

b) MCP_ERR. This file contains all recoverable errors encountered by MCP. The file can give early warning of a bug or provide pointers to the cause of a problem. The format of the data is as follows:

Date and time
An error message explaining what happened
The AOS/VS error code
The values of the MCP flags
The contents of the IPC packet last received
The contents of the IPC message buffer

c) MCP_LOG. This file contains a record of all activities performed by MCP. The file is used to check for attempts to breach security and the data stored is geared to facilitate this. Each record has an activity code which allows reports to be extracted for particular activities. Activities logged include:

All operator or system manager commands received
Invalid username / password pairs entered
User logon at a terminal
User logoff at a terminal
PC file transfers
Print requests
Batch requests

d) MCP_OUT. This file is assigned to the MCP process's generic output port. When the system is started the start time and date are written to this file. If a fatal error occurs when running a process then AOS/VS outputs the error code and trace back information to the subroutine and program line causing the error to the generic output file. This means that if MCP should terminate on a fatal error or panic then the information required to diagnose the problem will be contained in this file. This information along with the MCP_ERR file mentioned in b) above is generally enough to solve the problem. If it is not then the contents of MCP's variables are contained in the shared memory file and can be inspected by MCPMONITOR or some other tool to further diagnose the cause.

9.3 Emergency shutdown procedures.

Whenever an error condition occurs that requires AOS/VS intervention to safeguard system integrity, normally referred to as a "panic condition", AOS/VS forces the program or task to branch to a trap location. In the fortran libraries this location is referred to as "ULB?TRAP".

Under normal conditions this trap causes the program to terminate which would in turn cause AOS/VS to automatically terminate all sub-processes of MCP. This means that all users, batch jobs, printing, accounting etc would terminate without warning which is not a desirable situation.

To prevent this situation an assembler routine has been included in MCP that replaces the normal code for ULB?TRAP. When a panic occurs then this routine is executed. The routine monitors a shared memory variable called STOPFLAG. as long as this variable has the value 0 the routine will output a message to the master console, delay for a minute and repeat. The message sent to the master console is "**** MCP has panicked. Get users off and type MCPCRASH. ***". When STOPFLAG is set to a value other than 0 then the routine will terminate MCP.

MCPCRASH is a program that communicates directly with the MCRSH task in MCP. The MCRSH task is suspended waiting for a signal from MCPCRASH. When the signal is received it attempts to bring MCP down in as orderly fashion as possible.

Because MCRSH does not know which task has caused the panic, it tries to stop all tasks in the same way as the DIE command. A request is sent to the main task to disable all terminals and PCs. The MCPPRT task is instructed to stop all printers and the MCPBATCH task to stop the batch system. A command is sent to the accounting system to instruct it to terminate. As the operator will have asked all the users to stop work before executing MCPCRASH no real damage will be incurred if the task does not succeed in stopping the system normally. The MCRSH task waits for 5 minutes for the other tasks to terminate and then sets STOPFLAG to 1 which will cause the assembler routine trap to be freed and the system to terminate with the trace back output to the MCP_OUT log file.

9.4 MCP System Manager's Interface

This program is intended to provide the system manager with a tool for day to day control of the environment created by MCP. The programming is strait forward and the discussion will thus be limited to its functionality. Security of MCPSMI is vital as a person with access to this system can bypass most of the security features built into MCP. The program is thus only executable from the system managers user name and must have the AOS/VS super user privilege. The system managers user name is normally limited to one logon at a particular location (see security in chapter 4).

The program makes use of the fact that MCP's variables are stored in shared memory to gain direct access to MCP. This is particularly useful for the MCPMONITOR function discussed in section 9.4.4.

Access to the system is via menus. The main menu on entering the system is shown in figure 9.1

Figure 9.1 - The MCPSMI main menu

```
PROGRAM [MCPSMI]                                    DATE [48-1988]
                    MCP SYSTEM MANAGER INTERFACE
REV     [7.7]                MAIN MENU

              OPTION    MEANING

                0 - - - Exit from system
                1       Maintain MCP system files
                2 - - - Edit / create MCP user profiles
                3       Enter the MCP accounting system
                4 - - - Enter the MCP monitor
                5       Send a command to MCP
               88 - - - Get help with this menu

              Enter the desired option [  ]
```

Each of these options will be discussed in more detail in the following sections.

9.4.1 Maintain MCP system files

As the heading implies this option allows the system manager to control the files that MCP needs to provide the environment discussed in chapters 4 to 8. The sub menu shown in figure 9.2 is displayed,

Figure 9.2 The file maintenance sub menu

```
PROGRAM [MCPSMI]                                    DATE [48-1988]
                    MCP SYSTEM MANAGER INTERFACE
REV     [7.7]          SYSTEM FILE MAINTENANCE

              OPTION    MEANING

                0 - - - Exit to previous menu
                1       Edit the MCP screen logo
                2 - - - Edit the MCPLOCATIONS file
                3       Edit the MCP logon message
                4 - - - Edit the MCP shut down message
                5       Edit the MCP printer network definition
                6 - - - MCP command access control
                7       Edit / create MCP printer formats
               88 - - - Get help with this menu

              Enter the desired option [  ]
```

Option 1 refers to the file that MCP will use to construct the logon screen for the terminal users. As discussed in chapter 4 this allows the screen to be tailored to a particular companies requirements.

Option 2 refers to the MCPLOCATIONS file that forms one of the central control mechanisms of the system. As has been discussed in the previous chapters this file allows terminals to be controlled by location rather than individually and makes location security possible. The file identifies the network names of all PCs, the device type, headers and trailers of printers and allows the system manager to activate all peripherals with one command, namely ENABLE USERS. Due to the frequency of use of this information it is held in shared memory and is only accessed from the file on system start up or when the system manager issues a READ command to MCP. On completion of this option the system manager will be asked whether the changes should be immediately affected in MCP. If the answer is yes then MCP automatically issues a READ command to MCP.

Option 3 refers to the message that is displayed to all users when they log onto the system. Once the new message is created, the system manager may elect to make it effective immediately or at a later date.

Option 4 refers to the message that is displayed to a user when his console is disabled. As with option 3 the system manager may implement the new message immediately or at a later date.

Option 5 refers to the file of DG network nodes to be included in the MCP printer network. Once a machine is included in this file all the print queues will be accessible to all users in the network.

Option 6 refers to a utility that allows users to be given access to specific MCP commands. It is normally used to grant standby personnel access to the less dangerous commands such as terminal enable.

Option 7 refers to the print format and device files that allow MCP to program printers. Another submenu is displayed to allow either device, format or cleanup data to be entered. The menu is shown in figure 9.3.

Figure 9.3 The file maintenance sub menu

```
PROGRAM [MCPSMI]                                    DATE [48-1988]
                    MCP SYSTEM MANAGER INTERFACE
REV      [7.7]        FORMAT CONTROL UTILITY

          OPTION    MEANING

            0 - - - Exit to previous menu
            1       Generate or edit a print format
            2 - - - Generate or edit a device
            3       Generate or edit a cleanup file
           88 - - - Get help with this menu

          Enter the desired option [  ]
```

Generate or edit a print format allows the print and page formats to be entered as described in chapter 6.

Generate or edit a device requires the system manager to enter the printer codes that correspond to the different character pitches, line spacings, margins, letter quality and normal printing, tab stops etc.

Generate or edit a cleanup file allows the system manager to enter a set of codes that will restore the printer to a known state after graphics or word processing data has been sent to the printer.

9.4.2 Edit or create MCP user profiles.

As the heading implies this option allows the system manager to maintain the profiles of users of the system. On entering this option the sub menu shown in figure 9.4 is entered.

Figure 9.4 - The MCPSMI profile editor menu

```
PROGRAM [MCPSMI]                                    DATE [48-1988]
                    MCP SYSTEM MANAGER INTERFACE
REV      [7.7]        PROFILE EDITOR MENU

              OPTION     MEANING

                0 - - - Exit to the previous menu
                1       Create a new user profile
                2 - - - Copy an existing profile
                3       Edit an existing profile
                4 - - - Delete a profile
                5       Display passwords for a user name
               88 - - - Get help with this menu


              Enter the desired option [  ]

                    USER NAME [          ] (All options)
                    PASSWORD  [      ]     (Options 1-4)
 PASSWORD TO BE COPIED FROM [      ]     (Option 2)
```

Options 1 to 3 all result in the same three screens being used. The first screen is the environment screen and allows the system manager to set up the environment in which the user will run when using this profile. This includes the program to be run, the additional ID for menu security, the initial IPC to be sent to the process, the working directory, the default ACL and the search list.

The second screen is the AOS/VS process privilege screen and allows any of the privileges described in section 2.6 to be given to the user along with process scheduling priority, number of sons, process type, maximum memory working set size and logical address space size. For systems using AOS/VS class scheduling the class scheduling locations can also be entered.

The third screen allows the MCP security privileges to be entered. These are discussed in chapter 4 and include change password privilege, virtual console access, location restriction, modem access, force monthly password change, user name / password limit, time restrictions and network node on which the user should run.

Option 4 displays the environment screen and asks for confirmation that this is the correct user name password pair to be deleted.

Option 5 allows a user name template to be entered and displays all passwords for user names matching the template.

9.4.3 Enter the MCP accounting system

As with the other main menu options, the sub menu shown in figure 9.5 is entered.

Figure 9.5 - The MCPSMI accounting menu

```
PROGRAM [MCPSMI]                                    DATE [48-1988]
                  MCP SYSTEM MANAGER INTERFACE
REV      [7.7]          ACCOUNTING MENU

          OPTION    MEANING

            0 - - - Exit to the previous menu
            1       Edit penalty factors
            2 - - - Edit variable costing rates
            3       Edit fixed costs per cost centre
            4 - - - Edit the cost centre table
            5       Edit peripheral hardware allocation
            6 - - - Edit peripheral hardware costs
            7       Re-calculate peripheral costs
            8 - - - Produce monthly accounts
            9       Produce account summary
           10 - - - Produce capacity graphs
           11       Generate a capacity file for reports
           88 - - - Get help with this menu

          Enter the desired option [  ]
```

Option 1 allows the penalty factors explained in chapter 8 to be modified.

Option 2 allows the rates whereby the costs are derived to be modified. These will be set to approximately recover the costs of the department.

Option 3 allows fixed costs to be allocated to each cost centre.

Option 4 allows user names and systems to be added to the cost centre file.

Option 5 allows peripherals to be allocated to cost centres.

Option 6 allows costs to be allocated to each type of peripheral.

Option 7 does a recost of peripheral charges. This allows the data entered in options 5 and 6 to be verified.

Option 8 produces the monthly accounting reports described in chapter 8 for distribution to the users.

Option 9 produces a summary of the reports.

Option 10 produces the capacity monitoring graphs discussed in chapter 8. The user is asked to enter the device type and destination to allow the graphs to be plotted on plotters, the terminal, laser printers etc.

Option 11 allows a subset of the costing data base to be extracted for reporting via PRESENT.

An additional program not included on the menu has been provided for doing an entire re-cost of the months usages. This allows penalties and rates to be adjusted and costs modified in the case of unrealistic charges.

9.4.4 Enter the MCP monitor

The monitor has been provided to facilitate easy monitoring of the MCP environment in real time. This facilitates easy debugging of user problems, function key control over MCP, examination of MCP's shared memory variables, monitoring of MCP task activity, monitoring of inter task queues and debugging of MCP. Debugging is facilitated because shared memory is flushed to disk when MCP crashes, the monitor can be run and the contents of variables at the time of the crash examined.

The monitor consists of a number of screens that display different information about MCP. Each screen is selected by a function key. Once

in a screen the cursor can be moved to a console and the commands described in chapters 4 to 7 sent to MCP for that console via the function keys. The screen is updated with the latest information from the MCP variables every 10 seconds. This is the default cycle time and can be changed. An explanation of each screen follows.

a) Consoles on MCP.

This screen, by default, shows forty consoles that are currently logged on at a time. The console number, total number of PMGR messages transmitted for this console, MCP status flag, location, PID and user name are displayed for each console. The screen can be paged to see the next forty consoles. The "A" key can be used to toggle between a display of all consoles enabled and just active consoles. By moving the cursor to the relevant console and pressing the appropriate function key, any of the terminal commands described in chapter 4 can be issued for that console.

b) Users on MCP.

This screen displays detailed information for twenty users that are logged on at a time. The information includes the console number, PID, connect time, user name, program name, directory, process type, AOS/VS scheduling priority, the AOS/VS privileges assigned, the MCP privileges assigned, the maximum number of sons allowed, the network node of the machine that the user is working on and the location of the console. As with a) the cursor can be located on a user and MCP terminal commands issued via function keys.

c) MCP flag map.

This screen displays the MCP status flag for each console, arranged into a matrix of twenty consoles across and twenty consoles down thus allowing four hundred consoles to be

displayed simultaneously. The MCP status flag corresponds to the PMGR vectors and thus gives an accurate indication of the status of each console. The meaning of the status flag at the cursor position is displayed on the bottom of the screen. This is the screen most often used to debug user problems. The cursor can be positioned on the status flag of any console and terminal, PC or printer commands issued for that console port via function keys.

d) PMGR activity map.

This screen is similar to the flag map in c) except that the number of calls to PMGR that have been made since the last screen update is displayed. Again this information is displayed for four hundred consoles at a time. The screen indicates PMGR utilisation at a given time.

e) Printers on MCP.

This screen displays forty printers at a time and can be paged in a similar fashion to a) and b) above. The information displayed includes the console port number, the total number of PMGR calls for this printer, the MCP status flag, the format file currently in use and the file being printed. The cursor can be located on a particular printer and MCP printer commands issued via function keys.

f) Subordinate task queue map.

This screen displays the contents of the subordinate task queues. Different queues can be selected via function key. The queues that are displayed include terminal task queues, printer task queues and batch task queues. Each queue entry is represented by two numbers, namely; the console number to which the entry refers and the entry point vector. The information is used for debugging and checking that the system is balanced in

that no work build up occurs in any task. If the queue for a particular task grows too long then the task scheduling priority of that task is re-considered. The longest queue observed during a two week period was for the MCPPRT task which grew to seven entries.

9.4.5 Send a command to MCP

This option allows MCP commands to be constructed in an easy to use, self documenting fashion. All the available commands are displayed as a menu. The system manager enters the number next to a command and is then asked to supply the parameters one by one with all the options for each parameter explained in detail. The option is normally used for system training.

10. Conclusion

Design and programming of the system has taken place over a period of five years as a part-time project, and has resulted in 19456 lines of FORTRAN 77, 62 lines of DG Assembler, 1892 lines of 8086 Assembler and 3246 lines of Turbo Pascal being written. MCP is now running as a commercially viable system on several DG machines in the Middelburg Steel and Alloys group. These include two MV20 000s, one MV15 000, one MV10 000, one MV4 000 and one MV2 000. The system has run continuously without problems for the last six months (apart from shut downs for planned maintenance).

While objective measurements were recorded to determine the performance characteristics of MCP, the extent to which the system satisfies the needs of the system manager and the users is a subjective assessment. When a paper on the MCP system was presented at the 1987 NADGUG conference in Las Vegas [TEN87] and again in South Africa at the 1988 PERUG[21] conference, Data General users commented that the system addressed many of the problems caused by the limitations of the EXEC environment in their installations. The Steel Division system manager and the computer users within the division have commented that they have found the MCP environment easier to use, more powerful and more flexible that the EXEC environment. From these comments and the experience of using the system within MS&A, it might be reasonable to conclude that the MCP system has achieved its main objective, namely to provide an integrated environment that is more in line with modern peripheral technology and real time user's needs than EXEC.

The rest of this chapter is divided into a discussion on the performance characteristics of MCP and a discussion of some of the improvements that could be made to the system.

10.1 Performance characteristics of MCP.

The following graphs show the CPU, disk I/O and memory resources used by MCP during a day on the Steel division MV20 000 which supports 87 terminals, 27 PCs and 23 printers. Each of the three resources was measured every 10 seconds and

[21] Perseus User Group conference held in May 1988.

averaged for half hour intervals over the day. These figures were then averaged for a five day working week to obtain an average daily load of MCP on the system. The line for the maximum CPU represents the maximum cpu usage over the half hour interval. The graphs are shown in figures 10.1 to 10.3.

Figure 10.1 - CPU utilisation by MCP and PMGR

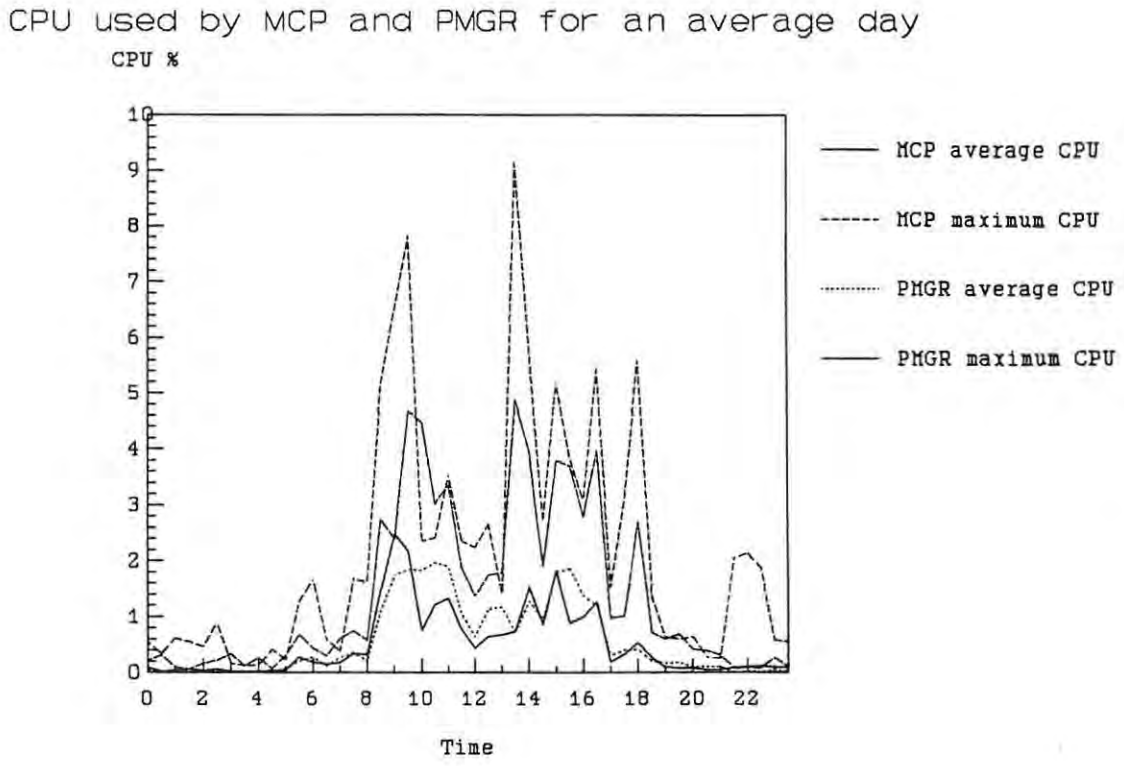


Figure 10.1 shows the MCP utilisation of the CPU. The figures for PMGR have also been included because of the close interaction between the two systems. User programs also use PMGR so a direct relationship cannot be drawn between the two. The graph shows that the average CPU utilisation never exceeds 3 %. This is considered to be very satisfactory considering the amount of work that MCP is doing. The peak in MCP utilisation between 8 am and 9 am can be attributed to office workers logging onto the system. The load for the rest of the day is due to printing and PC transfers. It is interesting to note that PC file transfers cause short periods of relatively high CPU utilisation (3% per PC) which causes the maximum line to fluctuate but that the average usage for the half hour is not

significantly altered. The peak in the maximum CPU line at 10 pm is due to the batch system running jobs submitted by users for night time execution.

Figure 10.2 - MCP disk utilisation

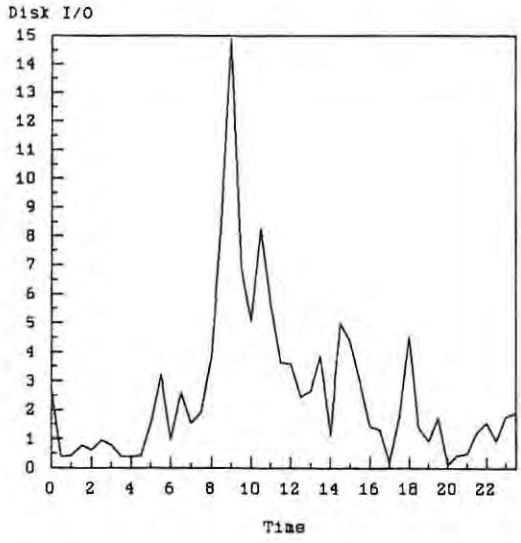
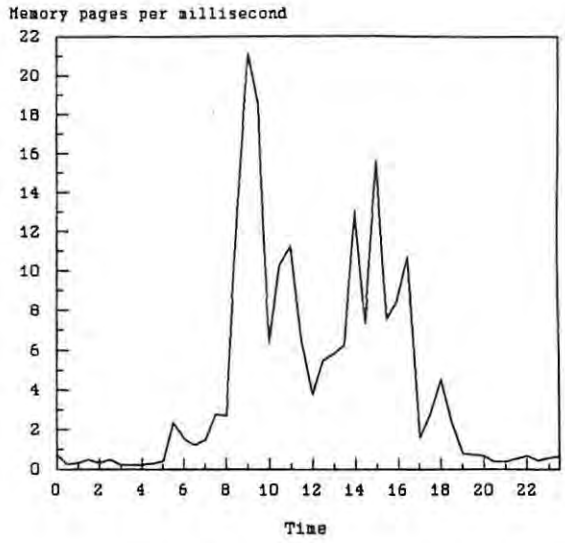


Figure 10.3 - MCP memory utilisation



Figures 10.2 and 10.3 show the MCP disk and memory utilisation. Both these graphs follow the average CPU utilisation quite closely. In both cases the load on the system is small.

As has been mentioned in chapter 5 the system load to transfer a file was 3% under MCP as opposed to 10% under CEO CONNECT. The overhead for file transfer is incurred mainly in PMGR which has to contend with large amounts of I/O. These figures were therefore measured late at night when no users were on the system. The figures were worked out as the sum of the MCP / CEO CONNECT and PMGR utilisation of the CPU during the transfer. Due to the growing number of PC users this saving is of major importance to system response.

10.2 Improvements to MCP.

In retrospect a number of improvements can be made to MCP. These include:

a) Memory organisation.

Currently the console and queue data tables are comprised of arrays grouped into common blocks. The problem with this is that when the information for one console (ie. one row of the table) is accessed, arrays in different pages of memory are accessed. This increases AOS/VS memory paging and requires a larger working set of memory pages for MCP. The problem could be solved by grouping each row of the table into contiguous memory. This is not a structure supported by FORTRAN 77 and thus would require either assembly language programming or packing of variables into strings. This could well result in more overhead than paging depending on memory utilisation.

b) Console table size.

Currently the console table size is fixed at 300 consoles. This is seen as excessively large for small systems and too small for the MV40 000 machines now in production. An improvement would be to allow the size of the table to grow dynamically using direct memory management calls to AOS/VS to increase memory as the table grows.

c) A dedicated task for the IPC port.

Currently the MCP IPC port is read by the main task. This means that in times of high activity, IPC messages get spooled to disk due to the main task's inability to read the messages fast enough. A better solution would be to create a separate task to monitor the IPC port and place messages in a queue for the main task to process.

d) Inter-task queues.

Currently inter-task queues are tables in memory with the entry point variable used by the subordinate task to identify work to be done. When the work is complete the entry point is reset to zero. The DG instruction set offers some very powerful linked list manipulation instructions which would be more efficient than scanning the table for work.

e) Communications with MCPs on remote machines.

Currently communications with remote MCPs is achieved via the IPC. This means that the AGENT has to intercept the message and send it to the remote machine via XODIAC. An improvement would be to create a task to receive messages direct from XODIAC. This would eliminate the disk I/O associated with IPCs and would eliminate the role of the AGENT. In addition the distributed aspect of MCP could be improved with packets being sent over the LAN for PC names etc thus eliminating the need to access the MCPLOCATIONS files on remote machines.

f) Direct record output from programs to print queues.

The current practice of using EXEC to collect the data and then transferring it to MCP print queues is not ideal. If the AGENT-EXEC interface can be worked out by "hacking", then a better solution would be to take over the interface and perform the function within MCP. This would be more efficient as it would allow the total discarding of EXEC. In addition, extra functionality not included in EXEC could be added such as allowing extensions to the queue name opened by the program to select different print format files.

MCP is in use and is constantly being enhanced to cater for more advanced user needs and new technology. Due to time constraints on the part of the author development will, in future, be taken over by a team of people in the system support department of MS&A.

Bibliography

AOS86/1 "AOS/VS Internals student handout" (1986), Data General Corporation, Westboro, Massachusetts.

AOS86/2 "AOS/VS System Concepts" (1986), Data General Corporation, Westboro, Massachusetts.

ASS85 Singh, A and Triebel, WA (1985), "8088 Assembler Language Programming: The IBM PC", Prentice - Hall, Englewood Cliffs, N.J.

BOE75 Boettner, DW Alexander, MT (1975) "The Michigan Terminal System", Proceedings of the IEEE, Vol 63, No 6, June.

CHE84 Chess, DM (1984) "A tight coupling of workstations", IBM Systems Journal, Vol 23, No 3.

DOS84 "Disk Operating System Referance Manual" (1984), International Business Machines Corporation, Boca Raton, Florida.

ECL86/1 "Eclipse MV/20000 series system elements Technical description" (1986), Data General Corporation, Westboro, Massachusetts.

ECL86/2 "Eclipse MV family 32-bit systems principles of operation" (1986), Data General Corporation, Westboro, Massachusetts.

GEN81 Gentleman, WM (1981) "Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept", Software - Practice and Experience, Vol. 11, 435-466.

GOL86 Goldstein, BC Heller, AR Moss, FH and Wladawsky-Berger, I (1984) "Directions in cooperative processing between workstations and

hosts", IBM Systems Journal, Vol 23, No 3.

HEB80 Hebbard, B Grosso, P Baldridge, T Chan, C Fishman, D Goshgarian, P Hilton, T Hoshen, J Hoult, K Huntley, G Stolarchuk, M Warner, L (1980) "A penetration analysis of the Michigan Terminal System", ACM Operating Systems Review, Vol 14, No 1, January.

HOO69 Hootman, JT (1969) "The pricing dilema", Datamation, Vol 8, August.

HOW86 "How to generate and run AOS/VS" (1986), Data General Corporation, Westboro, Massachusetts.

IBM86 "IBM PC Technical reference manual" (1986), International Business Machines Corporation, Boca Raton, Florida.

LAM84 Lambert, GN (1984) " A comparative study of system response time on program developer productivity", IBM systems journal, Vol 23, No 1.

LIN76 Linden, TA (1976) "Operating system structures to support security and reliable software", Computing Surveys, Vol 8, No 4, December.

MAN86 "Managing and operating the XODIAC network management system" (1986), Data General Corporation, Westboro, Massachusetts.

MCK79 McKell, LJ Hansen, JV Heitger, LE (1979) "Charging for Computing Resources", Computing surveys, Vol 11, No 2, June.

TAN86 Tanenbaum, AS and van Renesse, R (1986) "Distributed Operating Systems", Computing Surveys, Vol 17, No 4, December.

TEN87 Tennant, RS (1987) "Data General as a corporate solution", NADGUG, Las Vegas, October.

Appendix E

Answers to external examiner's comments

This appendix provides short answers to the comments raised by the external examiner. The original comments are reproduced in **bold** print at the start of each point.

1. **p.36: The explanation of virtual tasking is not very clear. It would also have been instructive to know whether the virtual tasking introduced any additional difficulties (bugs, pitfalls) during development.**

   The concept of a virtual task is a difficult one to explain. Another way of looking at it is to consider a virtual task as an extension of a PMGR task inside MCP. Figure 2.3 on page 17 shows that PMGR creates a task for each request it receives. If this is coupled with figure 3.3 on page 31 then it can be seen that on completion of a request, PMGR returns a confirmation message to MCP identifying the console and the vector sent to PMGR by MCP at the start of the request. This allows MCP to branch to the correct code block to continue processing for that device and also to map in the correct memory variables for that device. All of this allows one task in MCP to send requests to PMGR for many devices. The resultant tasks in PMGR send conformation messages back to the one task in MCP. As the majority of the time is spent in PMGR waiting for I/O, the single task in MCP does not become a bottle neck (in those cases where a danger of a bottle neck in MCP exists subordinate tasks are used). Thus even though there is only one physical task active in MCP, the appearance of a task for each device is created and hence the concept of virtual tasks.

   In terms of development difficulties the major danger areas were ensuring that code was truly re-entrant and that all variables were unique to either a console port or a code block. Diagnosing problems in the latter area was one of the reasons that shared memory was used for MCP's variables. In order to minimise the risk of indexing errors into the console port dependant variable arrays, strict standards were used throughout the programs. The first array index was always used for console port numbers

and the same variable, namely "ICN", was always used to hold the index value for the console port currently being processed.

The other main area of difficulty was ensuring that the code blocks were re-entrant and that the vectors into the various code blocks remained consistent. As the system grew and the number of vectors into and out of the various code blocks increased, the danger of code blocks working for one vector trail but not another increased. This required careful attention to be paid to generating truly re-entrant code as debugging a problem for a code block that serviced many consoles on different vector trails became difficult. In some cases the bugs could not be traced thus requiring that the code block be totally re-written.

2. **Some comments on the use of Fortran 77 as a systems programming language would have been instructive. Did the language itself introduce complexities? If he had a choice, and with hindsight, would Mr Tennant have used the same language again?**

Some of the aspects involved in using Fortran 77 are discussed on page 33. At the time that MCP was started Fortran was the main language used by Data General for systems programming. This resulted in a number of advantages to using Fortran. Firstly considerable effort had been invested by DG to make the language efficient. Secondly the interface between Fortran and AOS/VS was more advanced than most of the other languages offered by Data General. Thirdly the tasking environment offered under DG Fortran was very powerful.

In hindsight "C" would have been a better choice of language. Recently DG have changed to using "C" for systems programming and thus the advantages mentioned above for using Fortran now apply equally to the DG implementation of "C". In addition "C" memory structures would have allowed some of the problems mentioned on page 159 to be avoided.

3. **p.47: It is mentioned as one of EXEC's deficiencies that a person who gains access to the master console or super user privileges can obtain other users' passwords (if they are not encrypted). What is the difference**

between "gain access to the master console or super user privileges" under EXEC and becoming system manager under MCP? Both of these allow access to other users' passwords.

The first major difference is that MCPSMI can be restricted to only run on a specific console with a specific user name that has super user privileges. This severely restricts the ability to become system manager.

Under EXEC if someone manages to get super user privileges (one way of doing this is to gain access to the master console) they can go to the ":UPD" directory that contains the user profiles and display the profile file for a user and thus obtain his password. The procedure is very quick and can be performed in less than a minute. Another way would be to write a simple program to extract the passwords and to get the program run by a super user (such as asking the operator to run it).

Under MCP the passwords are all encrypted and held in one file which makes it difficult for a super user to identify the correct password and decrypt it. Thus the only practical method of obtaining passwords under MCP is to gain access to a specific terminal and to use the correct user name with super user privileges. This seems to be a more secure environment than that discussed in the preceding paragraph.

4. **p.47: What steps are taken to protect user profiles?**

This leads on from the discussion under the previous question. All profiles are located the ":UPD" directory. This directory has no ACL which means that only a super user can gain access to it. Under EXEC the profile files are given the same name as the user. This means that if a person obtains super user privileges then it is easy to identify which profiles belong to which users and to change them to his own ends. Under MCP an encrypted file is maintained that contains the user name, password and profile file name. Thus a person who obtains super user privileges will not be able to identify which profile belongs to which user unless he can decrypt the file containing user name, password and profile file name which is unlikely.

Furthermore the profile files are dumped from memory in block format which means that the file format will not be easy to decipher.

5. **p.60: The description of the PC-MCP communications protocol could have been clearer. It is for example not immediately clear that the extra character signifying an out of range character can in fact be one of three characters. Furthermore, what is the significance (if any) of the two mappings of out of range characters? What checksum algorithm is used? Will it for example detect two errors (see 7 below)? Presumably the protocol is Mr Tennant's own design. It would have been interesting to see some kind of comparison between this and other similar protocols.**

As is pointed out on page 61, the protocol is based on a standard protocol structure. The reasons for not exactly following an existing protocol are as follows:

a) The most important reason was to exploit the intelligence of the DG asyncronous I/O hardware. This results in fewer I/O interrupts to the main CPU and hence greatly reduced overhead. In order to achieve this the protocol had to fit within the bounds of normal DG data sensitive I/O. The range of data that can be safely sent lies between 32 and 126. In order to map the full range of 0 to 255 into this range in fact required three mappings, one for characters less than 32 and two for characters greater than 126. The reason for two mappings for characters greater than 126 is that the available range of 95 values between 32 and 126 cannot accommodate the 129 possible values between 127 and 255.

b) The second reason had to do with reducing the number of messages transferred over the asyncronous link. Although all characters less than 32 were excluded from the protocol, there were some of these characters that could be used in data sensitive reads. This allowed for multiple values for REQUEST, ACK and EOT (see figure 5.1 on page 61). This in turn allowed information to be transferred as part of the protocol itself thus greatly reducing the number of messages between the machines.

The checksum algorithm used is an arithmetic sum of all the characters in the message which is then truncated down to the least significant three digits. Upon receipt of the message both the checksum and length are checked and if both pass then the message is considered correct by the protocol but not necessarily by MCP (see 7 below).

6. **p.72, point (a): No mention is made of what happens when the buffer becomes full before CR/LF is encountered.**

This was an oversight. If the buffer grows beyond 72 characters without a CR/LF being encountered then the buffer is sent and cleared.

Subsequent to the submission of the thesis this piece of code was enhanced to do away with the CR/LF check as this resulted in many small messages being sent which was not efficient. Instead a special timeout has been incorporated based on the PC's clock interrupt. If no characters are received for printing during the timeout period then the contents of the buffer are sent to the DG.

7. **p.88: The request code received from the PC determines what code block will be executed (control coupling). Is the protocol and code robust enough to recover from a corrupted request number received over the network? For example if one of a sequence of request type 3 messages is corrupted to a request type 2 (and a second error in the message cancels the checksum error), would the transfer be aborted gracefully?**

In terms of the protocol, provided the checksum and length checks pass, the message will be accepted. The example cited here would therefor pass the protocol. It would however not be accepted by MCP as the requirements for terminating a type 3 transfer and initiating a type 2 transfer would not have been met. This would result in an ACK-2 with error bit (bit 6) being returned. This would result in the transfer being aborted but not very gracefully (ie an error would be returned to the application program rather than the protocol attempting a re-try). It should be stressed however that the scenario presented here is a special case where not only

is the request character corrupted to another valid request character but another error occurs to prevent a checksum error. In MS&A there are currently over 200 PCs connected to Data General machines that use this protocol and no occurrences of this problem have yet been reported.

In general the area of the protocol that created the most problems was syncronisation. Originally only one ACK character was used in the protocol. The main problem that was experienced was that all I/O to the IAC on the DG is buffered. If therefor syncronisation was lost due to a lost REQUEST character or slow response resulting in a time out then problems were experienced with re-establishing syncronisation. The solution to this problem was to introduce the second ACK character to minimise the effect of buffering on syncronisation.

8. **p.92: Presumably the "record of all print queues" kept on all machines just indicates the existence and location of each queue (else how is consistency guaranteed?).**

Correct. Remote print queues merely consist of a pointer to the machine on which they reside. All requests concerning these queues are then re-directed to the relevant machine for processing.

9. **p.104, last paragraph: How is unauthorised access to the queues prevented?**

No access control has been provided for print queues. This means that any user can submit a printout to any queue. The reasons for this are covered in some detail in the thesis in the discussion on forms (see page 91).

If the question refers to the fact that queue data is held in shared memory and a risk exists that a different user can connect to the same shared memory and corrupt the data then the following considerations are relevant. Firstly access to shared memory is achieved by linking to the shared memory file on disk (see page 33) and as this has no ACL only super users can gain access. Secondly AOS/VS allows access to shared memory to be restricted to read only.

10. **How does the accounting system described in chapter 8 compare with those of other systems (that do have reasonable accounting)?**

The normal accounting systems take cognisance of CPU, disk, memory, connect time and pages printed. Rates can be applied to these figures but the charging of shared resources and development costs are done manually. The major differences of the MCP accounting system are as follows (see pg 127 and 128):

a) MCP makes provision for penalties to be charged for inefficient use. This has the effect of stretching the machine capacity as users are encouraged to work efficiently.

b) MCP allows charges for shared resources (such as data base controllers) to be automatically allocated to a number of cost centres based on the usage by each cost centre.

c) Development resources used for systems are charged out automatically to cost centres based on their usage of the systems.

11. **What overhead is introduced by the accounting system? Has any performance measurements been made with accounting turned on and off?**

The accounting system has three modes, namely; OFF, BRIEF and DETAIL (see page 135). When either brief of detail modes are selected the system activity is scanned every 60 seconds. This entails one system call to ascertain the processes running on the system and a system call for each process to ascertain the activity for that process. The results are then accumulated in memory arrays. The task takes between 0.5 and 2 seconds to execute and results in a CPU usage, on average, of 0.1% for that interval.

In brief mode the accounting information and total utilisation for capacity monitoring is written to disk every 30 minutes. This process takes between 10 and 20 seconds (depending on disk utilisation at the time) and uses between 0.5% and 1% of the CPU for this period. The performance figures presented in chapter 10 were measured with accounting in brief mode.

In detail mode capacity information is kept for each program run by each user on the system. This results in a large increase in information that is output to disk every 30 minutes. The process can take up to 2 minutes and uses between 2% and 3% of the CPU for this period. Because of the relatively high overhead this mode is not intended to be used for normal operation but rather for special investigations or research into machine performance.