EMAS—The Edinburgh Multi-Access System

H. Whitfield* and A. S. Wight

Department of Computer Science, University of Edinburgh, The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland

EMAS is a general-purpose time-sharing system for the ICL System 4-75 computer, with provision for fully interactive and background processing, It is a virtual memory system with a three level storage hierarchy.

An outline description of the system is given together with a more detailed description of the paging and scheduling software, which is based on the working set concept. Detailed performance figures are included.

(Received October 1972)

The Edinburgh Multi-Access System (EMAS) is a general purpose time-sharing system for the ICL System 4-75 Computer.

Work began in August 1966 and for four years was under the control of the Edinburgh Multi-Access Project (EMAP); a joint project between the University of Edinburgh Department of Computer Science and International Computers Limited (ICL). The computer itself was delivered in December 1968 and the disc-file followed in June 1969. The Computer Science Department took over the development of the system in October 1970 and service began in October 1971. The Edinburgh Regional Computing Centre (ERCC) which runs the computing service has taken an increasing role in the development and maintenance of the system since October 1970 and has now (October 1972) taken it over almost completely. The ERCC has also been involved in a major way in the provision of the compilers for IMP (Stephens, 1973), the language in which the system is written, and in the provision of other subsystem software.

About 25 programmers of various levels of experience were involved in the production of the software during the first four years. During the past two years about seven people of very considerable experience have been working on the software much of which has been re-written either totally or in part. EMAP was intended as a research and teaching project as well as a production exercise. It is clear in retrospect that all three functions cannot be reconciled and attempts to do so inevitably lead to delays in production.

The first part of this paper is intended to give a broad picture of the organisation of EMAS sufficient to enable the second part and the other papers to be placed in their proper context. The second part contains a description of the resident supervisor. Where the system follows well understood principles only brief descriptions will be given. We have tried to give actual statistics and performance figures where we have them. We do not believe that these are necessarily better than those of other systems, but we do feel that there is a great lack of hard facts in this area. We must apologise that we do not yet have a coherent performance model of our system and offer the statistics in the hope that others will be encouraged to publish similar figures.

1. Hardware

The ICL System 4-75 computer is the largest of the System 4 range and has paging (address translation) hardware. The non-privileged instruction set of the System 4-75 is the same as that of the IBM 360 series machines. The channel arrangements are also similar but the channel program command codes are not the same. The interrupt and associated context switching arrangements are similar to those on the RCA Spectra 70 series machines, in that System 4-75 has four sets of General Purpose Registers corresponding to four different states called P-states. Normal interrupts, such as channel or program error interrupts, cause entry to the P3 state and the corresponding registers are used by the processor. Power fail and machine check interrupts cause entry to P4 state and the processor then uses the P4 registers. Some of the registers of the P3 and P4 states are aliases for special registers of all four states. Each state has an interrupt mask register, an interrupt status register and a program counter. These fulfil a similar role to the program status words on the IBM 360 series and are amongst the aliased registers. It is not very convenient to run significant amounts of program in P3 or P4 states because the number of registers available for general use is reduced. The programs which run on entry to these states contain instructions which cause an immediate change to P2 state, where further processing is done. P1 and P2 states each have the full set of 16 registers and none of these are aliased or used for special purposes. The P1 and P2 states are completely equivalent. Because most of the supervisor is compiled from a high-level language and uses all of the registers, we have chosen to use P2 state for the resident (unpaged) supervisor and P1 state for all paged programs. A consequence of having the four P-states is that no storing and restoring of registers in main memory is necessary on a switch of context from one state to another.

There is, however, only one set of Floating-Point Registers. Although these are addressable in all four P-states we have chosen to regard them as registers of P1 state. Normally they are not changed by the other three states, but parameters are passed between P1 and P2 states in these registers.

The configuration in use at Edinburgh is as follows:

4-75 CPU†, operator typewriter and console.
768K bytes 1μ sec core store—4 bytes/access and two way interleaved.
2 × 2M byte magnetic drums on one channel, transfer rate of 860K bytes/sec.
Each drum has 128 tracks and there are 4 pages (of 4096 bytes) per track.
Revolution takes 20 mS.
2 × 350M byte non-replaceable disc-file with moving arms, consisting of two devices on one channel. Transfer rate 256K bytes/sec.
Revolution takes 40 mS.
Average arm movement takes 60 mS.
3 × 7·5M byte replaceable disc drives.
4 × 120K bytes/sec 9 track tape drives.
1 7-track tape drive.

*Present Address: Mathematisch Instituut, Rijskuniversiteit te Groningen, Postbus 800, Groningen, The Netherlands.
†Typical instruction execution times are given in Table 1.

Table 1 Sample instruction execution times

MNEMONIC	DESCRIPTION	FORMAT	MICROSECS.
LR	Load	RR	0·80
L	Load	RX	1·65
ST	Store	RX	1·25/1·47
STM	Store Multiple	RS	6·13 for 8 words
A	Add	RX	1·93
AR	Add	RR	1·09
MR	Multiply	RR	5·45
MVC	Move	SS	12·09 for 16 bytes
BALR	Branch and Link	RR	1·79
BC	Branch on Condition	RX	1·44/1·67
BCR	Branch on Condition	RX	1·31
ADR	Add Normalised (Long)	RR	3·22
MDR	Multiply (Long)	RR	11·04

2 Line printers.
2 Card readers.
1 Card punch.
1 Paper tape reader.
1 Paper tape punch.
1 Communications multiplexer with
64 Permanently wired teletypes,
16 Datel 200 ports,
4 1200 baud buffers with character video terminals,
5 2400 baud synchronous buffers.
1 British Standard Interface which connects to a PDP-15 with interactive graphics hardware.

Authorised enhancements are:

1 × 2M byte magnetic drum to go on the existing channel.
256K bytes 1μ sec core store.

The paging hardware

The 24 bit effective address which is generated by the usual process of address computation is regarded as a virtual address and is translated by the paging hardware into a physical address. The method of translation is sketched in **Fig. 1.** Although there is a segment table and a segment field in the virtual address, it is not true (or symbolic) segmentation as the segment and within segment addresses are not computed independently. However, it will be seen later that many of the desirable properties of segmentation can still be realised.

The segment and page tables must be set up by the supervisor so that the virtual to physical address mapping is correct. The page table entries have availability bits to indicate whether a page is in core and an interrupt occurs if access is attempted to a non-available page.

The associative memory has eight cells and remembers the eight distinct most recent accesses avoiding the two extra core cycles needed to access the segment and page tables most of the time.

It can be seen that this paging unit is similar to that on the IBM 360/67*. However the System 4 unit provides 256 segments of 16 pages (each of 4096 bytes), whereas the 24 bit IBM 360/67 provides 16 segments of 256 pages. This larger number of segments allows the segmentation to be used in a more convenient fashion.

*The recently announced dynamic translation unit on the IBM 370 series machines has four models of operation, one of which is essentially the same as the IBM 360/67 and one the same as the ICL 4-75.

System software

EMAS is a virtual memory system. Each foreground or background user runs programs in an independent virtual memory and each such user process has a share of the resources of the system (core, CPU, etc.). At system interface level each user has a virtual memory of maximum size 2^{24} bytes organised as 256 segments each of 2^{16} bytes. Segments 0–31 of each virtual memory are used by the *director* processes and are not available to the user.

Central to EMAS is its File System which contains named files belonging to all users. Each file is a completely unstructured sequence of bytes of arbitrary length (in units of one page or 4096 bytes). Files have two part names as follows

MAC007.FRED

where the first part is the name of a user of the system (the owner of the file) and the second part is his chosen name for the file. All files are held on-line and so are immediately available.

Files are accessed by connecting them, i.e. by having a mapping set up between the whole file and a segment (or several contiguous segments) of virtual memory. This mapping is done by a request to the system and once established access to the file is by direct reference to the appropriate virtual address. The system provides for controlled access to files in read/read-write and shared/unshared modes. When files are used in a shared mode all users have access to the same physical copy whether this is on disc, drum or in core. This facility is used extensively to share the code of programs at various levels in the system and results in marked saving of core, drum and disc space.

Input from cards or paper tape is handled by a system process called *demons* and appears as files in the file system. Output to printers, etc. is also handled by demons which prints files as soon as possible after receiving a request. The foreground consoles are the only devices which communicate directly with running user processes. The user appearance of the system is sketched in **Fig. 2.**

Files reside on the disc-file where they are stored in page (4096 byte) blocks. The system uses the drums and core to buffer parts of files while they are being operated upon.

The system software can be thought of as a set of concentric shells. In the centre we have interrupt analysis, the CPU despatcher, the basic synchronising primitives and message passing software. Then there are the supervisor processes. These are concerned with the following functions:

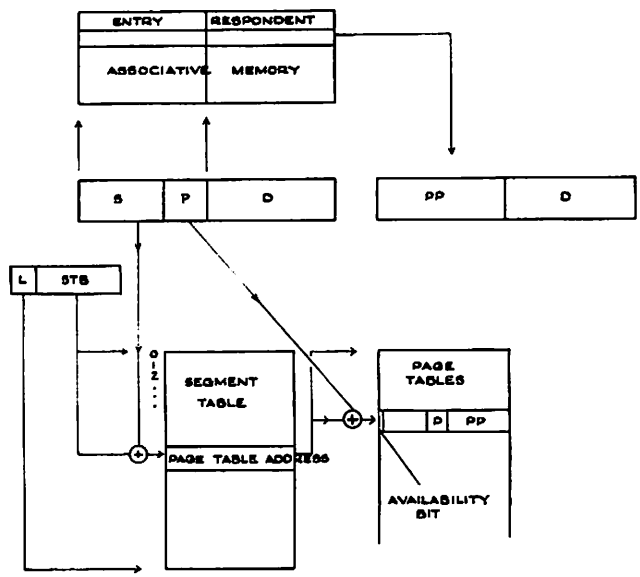


Fig. 1 Paging hardware

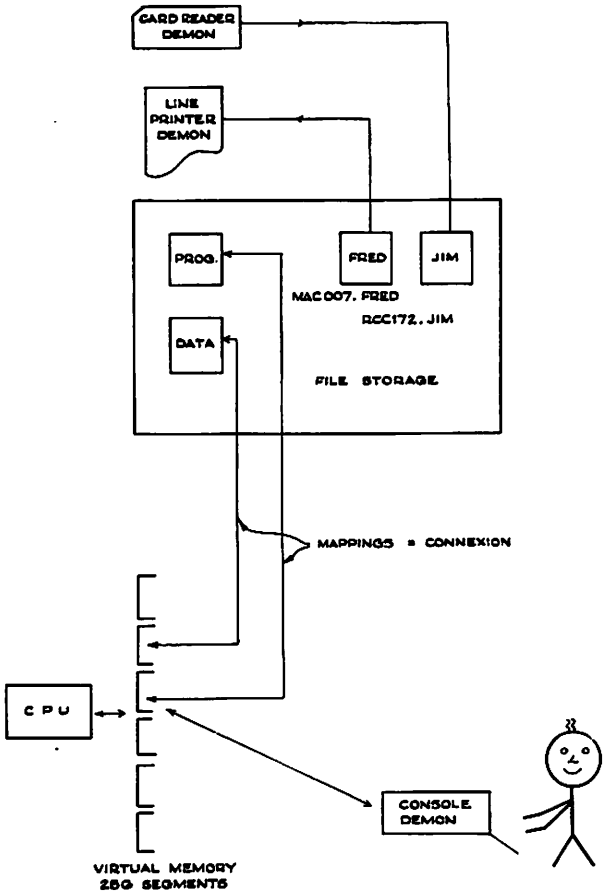


Fig. 2 User appearance of EMAS

(*a*) Driving the fast devices—e.g. drums and disc-file.
(*b*) Driving the communication devices—e.g. typewriter consoles.
(*c*) Providing minimal support for slow devices—e.g. tapes, card readers, line printers.
(*d*) Supporting virtual memories—i.e. the paging software.
(*e*) Scheduling of CPU-time and core for paged processes.
(*f*) Control of CPU errors.

Code for these processes is permanently resident.

The next shell of the set consists of the director processes. Each user process has an associated director process which performs (primarily) file system and console communication services on its behalf. The director process has access to the whole of the virtual memory whereas the user process can access segments 32–255 only. The director process is superior to the user process and can start or stop it and perform various recovery functions. The intention here is to handle all non-time critical system activities in paged processes and it is easier to have such a paged director for each process. Where appropriate (e.g. when accessing file indexes) the directors are interlocked via a semaphore scheme. They all share identical code and use the same physical copy. It should be noticed that supervisor overlays are in this way made unnecessary, as the paging of the directors by the standard paging software provides a similar function in a much more elegant way.

It can be remarked here that it has been a general principle in our design not to place any system function at a more central position than is strictly necessary.

Further system functions such as user vetting on logging in and the input and output functions mentioned above are handled by the demons process (Hayes, 1973).

The detailed organisation of the director is described in an accompanying paper (Rees, 1973), but it is worth looking immediately at the relationship between the paged director and the resident supervisor. **Fig. 3** shows the mapping on the first 32 segments of each virtual memory. Segment 0 is not normally used for somewhat bizarre reasons connected with a design difficulty in the addressing system. Segment 1 is the *buffer segment* which addresses the buffers into which all console and slow device input/output takes place. Segment 1 is common to all director processes and the pages which are allocated as buffers are locked in core while transfers are in progress. Segment 2 is the *master segment*; it is in read-write unshared mode and contains the working variables for this incarnation of director. These consist of a static-storage area and a stack; there is also one page (page 0) used as the *master page*. This contains the mapping of segments onto files for this virtual memory and space for copies of the register sets* of the director and user processes. It also contains usage information for pages of files in active use. These tables on the master page are the only ones accessed by both the director and the paging software. It is the responsibility of directors, which control the file system, to write to the master page correct mapping information which relates segment numbers to the physical disc locations of the desired files. The paging software operates entirely from the master page and has no knowledge of files as such or to whom they belong. When a process is loaded to core the master page must always be fetched—all other pages can if necessary be demand-paged using the information on the master page.

Segment 3 addresses the file containing the director code. This pure-procedure program file is accessed in read shared mode by all directors.

Segments 4–31 address the file indexes of all accredited users and are accessed in read-write shared mode by all directors.

Subsystem software

The items of software mentioned above, viz. the resident supervisor, the directors and the demons process, constitute the *system*, i.e. that part of the whole software complex which the ordinary user cannot change. This basic system provides the virtual memory support and file protection mechanisms.

A user at a console or a background user requires a great deal more than this, e.g. command interpretation, job control language analysis, loading operations, use of commands or catalogued procedures and use of utilities for the creation,

*A register set consists of the program counter, the general purpose registers and the floating point registers.

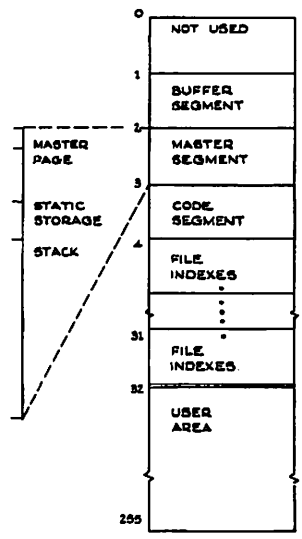


Fig. 3 Mapping on segments 0-31

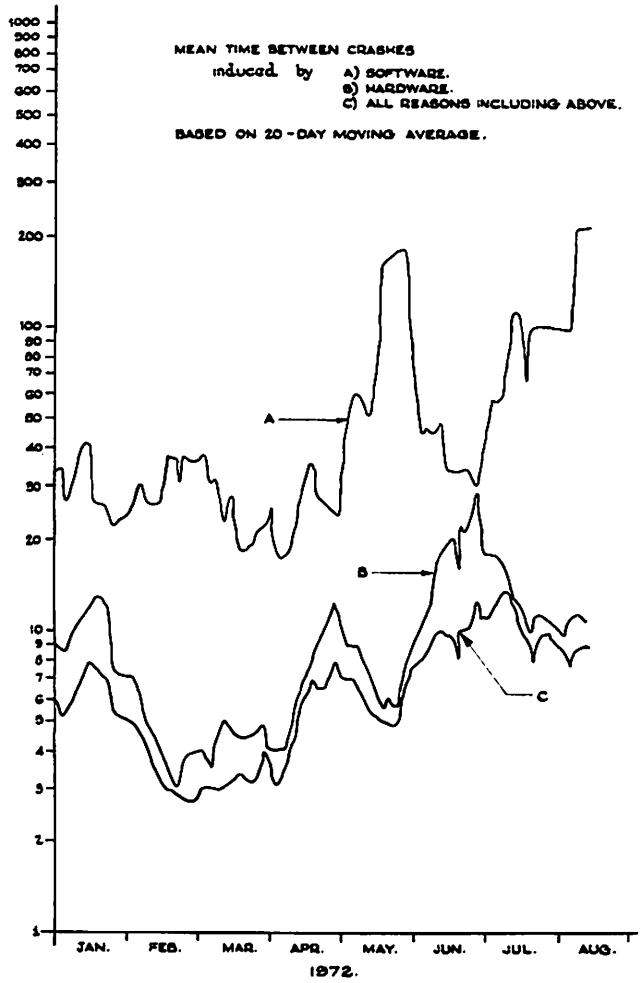


Fig. 4

management, compilation and execution of programs. In EMAS these facilities are provided by ordinary programs which are protected and shared by the normal file control and file access mechanism. A powerful subsystem (or collection of such facilities) with a good range of commands has been provided for the user, but there is nothing to prevent a user adding his own commands (merely by compiling a program) or indeed from choosing to use a completely different subsystem. This is useful to the subsystem programmers who can test new subsystems without affecting other users of the system.

The standard subsystem is described in a separate paper (Millard, Rees and Whitfield, 1973). The compilers in the standard subsystem produce pure-procedure code in a standard layout and observe parameter passing and linkage conventions which permit cross-calling between routines in different languages. The files of the standard subsystem belong to a user of the system called *manager* who also owns the password lists and is responsible for accrediting new users.

History and current status

When work on EMAS began in October 1966 we naturally turned to the paper on Multics (Corbató and Vyssotsky, 1965) as a starting point. This greatly influenced our thinking as did the paper by Arden, Galler, O'Brien and Westerveld (1966). Work on Multics began in 1964, two years earlier than on EMAS, but intermediate results from the Multics Project did not come out soon enough to have much influence on the subsequent course of our work. It is therefore interesting to compare the present state of EMAS with that of Multics (Corbató, Saltzer and Clingen, 1972).

We have not been able to explore those areas which require a multi-processor configuration because of our hardware provision and we deliberately chose not to have hierarchically structured file directories and system administration (our standard subsystem does however provide a hierarchical structure of libraries of entry points of programs). Apart from this we have set ourselves essentially the same goals as Multics and have attained them in all essential respects.

The ICL 4–75 computer on which EMAS works is a slightly modified 4–70 computer. The only additions are the paging unit and 'the use and change' markers on the core keys. The drums do not have hardware queueing facilities but a similar effect is organised in the drum software.

The hardware became available for software development over the first half of 1969, again two years later than in the case of Multics. Until then we used the KDF9 computer with an IMP compiler which produced code which simulated the effect of the same program on System 4. By September 1970 we had put a good deal of the system together. However, it was apparent that a major re-design was necessary in the file system area.

Here again the similarity to the Multics experience is remarkable. The cause of necessary re-design was not so much bad coding as a failure to keep the software sufficiently simple in concept to minimise the amount of code required. In particular we had not realised that interacting paged processes, competing for the same core, would take so long in elapsed time to complete complex sequences of operations.

We very much simplified our specification to include only the minimum features logically necessary, simplified the management of the file system, and improved the interfaces between certain components. However, it was not necessary to change the design of the resident supervisor. We have since re-written the file system twice more to provide improved facilities, to use file disc space efficiently, and to make the director more efficient in its paging characteristics. The supervisor has also been improved in many significant ways although it has not been necessary to re-write it. The iterative technique of program design suggested by Corbató *et al.* (1972) is entirely in accord with our experience.

By early 1971 (again two years later than Multics) EMAS was sufficiently effective for us to be able to use it for all our development. At this point we began to make rapid progress. By October 1971 the system was made available to general users. EMAS currently supports 150 accredited users, and is operated 18 hours per day Monday to Friday and 10 hours on alternate Saturdays. The system currently supports about 30 fairly demanding users using one CPU and 768K bytes (equivalent to 192K 32 bit words). Most of the user consoles are within two miles of the computer building and are used by staff and research students of the University of Edinburgh and by Government Research Council Institutes in the Edinburgh area.

The system performs a great deal of error detection and recovery, and is reasonably proof against all but errors in the CPU. Fig. 4 shows the crashes for the first half of 1972. The software time between crashes is large despite the fact that the system is under continuous development. We would expect the software error rate to fall to zero if we ceased development. After a crash we take a dump to magnetic tape and re-IPL. This takes less than a minute.

The EMAS system, i.e. resident supervisor, director and demons, consists of 11 modules, totalling about 20,000 source statements in all. These compile into about 180K bytes of code. The basic subsystem, including compilers, commands and basic libraries, consists of another 33,000 statements and 500K bytes of code. Details are given in Tables 2 and 3. The compiler

Table 2 Sizes of system components

COMPONENT	FUNCTION	LOCATION	SOURCE TEXT (STATEMENTS)	CODE SIZE (BYTES†)	STATIC DATA (BYTES†)
SPAM	PAGING	RESIDENT	5,200	A,370	1,9C0
CEDRIC	COMMUNICATIONS	RESIDENT	1,425	2,3C8	1,C18
PART	PARAMETER PASSING AND OPERATOR CONTROL	RESIDENT	1,705	3,F88	3,510
FAST	DISC DRUM LOADUP	RESIDENT	1,525	2,6E0	1,C68
TAPE	MAGNETIC TAPE	RESIDENT	1,308	3,2A8	A98
SLOW	SLOW DEVICES	RESIDENT	741	1,520	8D0
PERM*		RESIDENT		618	140
DIRECTOR	FILE SYSTEM COMMUNICATIONS	PAGED	3,500	9,600	1,C98
DEMONS	ACCOUNTING SPOOLING BACKGROUND JOB INITIATION LOGGING ON	PAGED	3,849	C,978	1,089

†In hexadecimal
*In Assembly Language
Compiled code run-time routines

Table 3 Sizes of selected subsystem components

COMPONENT	FUNCTION	SOURCE TEXT (STATEMENTS)	CODE SIZE (BYTES†)	STATIC DATA (BYTES†)
ACP & BCI	ERROR CONTROL and COMMAND INTERPRETATION	818	2,680	5B0
EDCP	STREAM-FILE MAPPING	1,203	3,FC0	1,928
FPD	VM LAYOUT and PROGRAM LOADING	1,412	4,880	1,410
BCLL	BASIC COMMANDS	1,308	6,298	9E0
EDIT	EDITOR	977	2,CA8	308
FORTE	FORTRAN COMPILER	7,749	10,058	1,4D0
IMPS	IMP COMPILER	9,800	16,A78	D10

†In Hexadecimal

operates at about 60 statements a second. Linkage of supervisor takes less than one second. A complete system generation takes less than 10 minutes and a new component can be introduced in less than two minutes. The size of the resident supervisor is 95K bytes of code and for a 30 user load about 128K bytes of data and buffer areas. We make new supervisors and directors about three times a week. New subsystems are introduced about twice a month, but new versions can be tested at any time.

About 120 man years of effort have gone into the system to date, a figure which is similar to the Multics experience. The cost of the software was about £450,000 including about £200,000 paid to the ERCC for some 3,000 hours of computing time during the development period. In the past two years the design and implementation has been in the hands of six people who understand the system and subsystem between them in complete detail. It is estimated that this team, using the existing system as a tool, could move the whole system to a suitable new machine with a different order code in one to two years or to a very similar machine such as an IBM 370 series machine with paging in about six months.

Like Multics, EMAS allows each user to choose the program file which is mapped onto the virtual memory before his process is initiated. The appearance of the system is determined partly by the routines in this sub-system *base file* and partly by the other files and services to which the user has access. EMAS too has a stack-oriented, pure procedure environment with libraries mainly in IMP and FORTRAN IV. **Appendix 1**

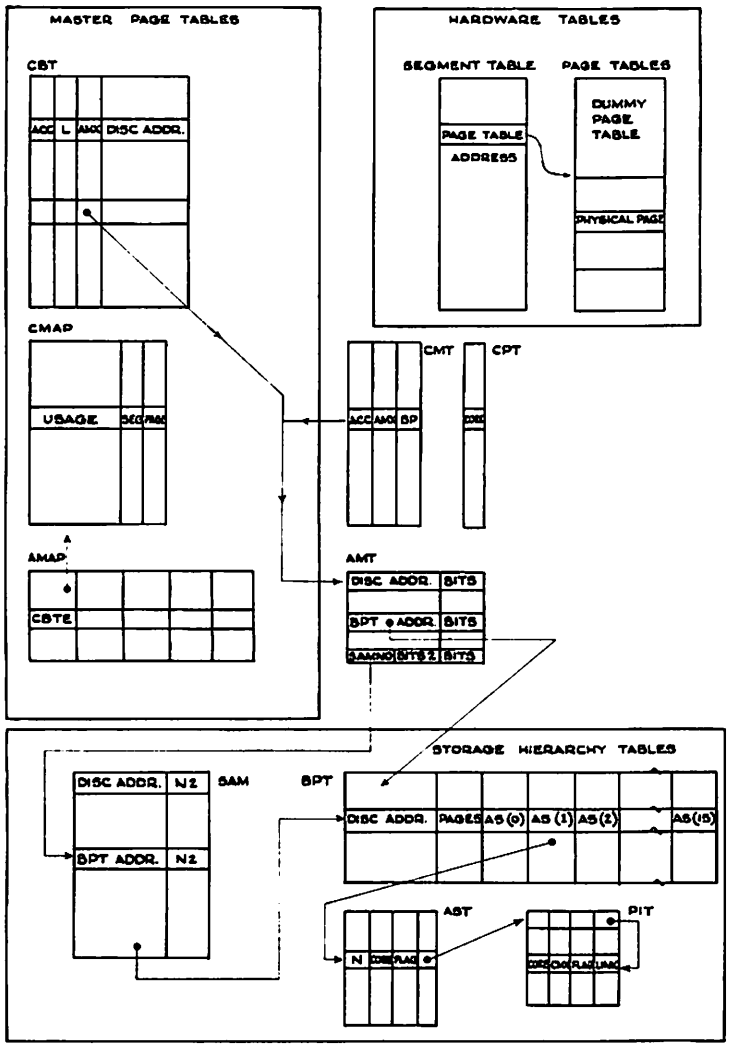


Fig. 5

lists some of the facilities available.

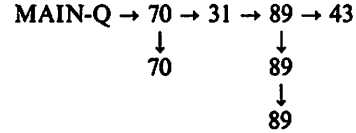
It cannot be stressed too strongly how important the use of IMP has been to the whole exercise. It is only by use of a high-level language that the size of the text of the system can be kept down to a level where a very few people can understand all of it. Why system programmers should continue to deprive themselves of facilities which they feel obliged to provide for others is beyond our comprehension. Good compile time and run time diagnostics make for rapid program development and an optimising mode in the compiler for validated programs solves the efficiency problems. In systems programming it is important that the language make the relationship between the source text and the object code efficiency fairly obvious so that the programmers avoid awkward constructions. In cases where the ultimate in code efficiency is required we have used in-line machine code rather than separate code subroutines which always suffer from routine calling and return overheads. The IMP compiler has 90K bytes of code and uses 25K bytes of data.

2. The EMAS supervisor

The kernel

At the heart of the EMAS supervisor is the CPU despatcher. It is not possible to have more than one CPU in a System 4–75 configuration so the despatching problem is slightly simplified.

The supervisor consists of a number of servicing routines*. Each of these provides a set of associated *services*. For every service there is a service number which can be regarded as identifying a private semaphore (Dijkstra, 1968). A supervisor service is activated (kicked) by sending a request message. This can be regarded as a *V*-operation on its semaphore with the queuing of the associated message. There is a queue (the MAIN-Q) of services awaiting attention by the CPU. They are queued initially in order of arrival, but second or subsequent requests for the same service are held on side chains so that all requests on a particular service can, where appropriate, be serviced at one time.



The despatcher, which runs in P2 state, selects the first item on the MAIN-Q and calls the appropriate routine. The routine is then executed until it returns to the despatcher, all interrupts other than machine checks and program errors being masked off in P2 state. In the course of execution the routine may kick other servicing routines causing further items to be placed on the MAIN-Q. When it is called the servicing routine is passed the request message. It may ask for any other messages on the MAIN-Q side chain if it wishes. When the routine has no more to do it returns to the despatcher. This can be regarded

*Each servicing routine may be regarded as the code of a supervisor process.

Table 4 The category table

CATY	PRIORITY	CORE (PAGES)	RESTIME (SECONDS)	LOOKTIME	NCY1	NCY2	NCY3	NCY4
1	1	44	1	0·125	14	13	9	11
2	1	13	1	0·5	3	2	0	2
3	1	26	1	0·5	4	3	2	3
4	2	39	1	0·5	4	4	3	4
5	1	13	1	0·5	8	6	0	5
6	1	13	2	0·5	9	7	0	5
7	2	13	5	0·5	10	7	0	5
8	1	26	1	0·5	11	9	5	8
9	2	26	2	0·5	12	10	6	8
10	3	26	5	0·5	13	10	7	8
11	2	39	1	0·5	14	12	8	11
12	2	39	3	0·5	14	13	9	11
13	4	39	7·5	0·75	15	13	10	11
14	3	52	2	0·5	14	15	12	11
15	4	52	7·5	0·75	15	15	13	11

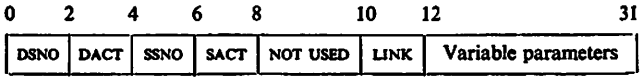
Demons starts in category 2 and all other paged processes in category 1.

as a *wait* or P-operation on its private semaphore.

Routine calls are also provided to enable a servicing routine to inhibit or uninhibit the activation of particular services. These are used, for example, by routines which maintain internal queues of their own for optimisation purposes. When their queues are full they inhibit further requests until servicing of some of the existing requests is complete.

The despatcher normally works through all the uninhibited requests on the MAIN-Q causing execution of the servicing routines in P2 state before switching to the execution of the current P1 state paged process. However, it does examine the interrupt flag register before calling a new servicing routine and if high priority channel interrupts (drum or disc-file) are present it exits to P1-state immediately, to enable the interrupt to be taken (P1-state is run with all interrupts enabled). When interrupts occur a request is placed on the MAIN-Q in the normal way for the servicing routine. However, the high priority interrupts cause immediate execution of the servicing routine which effectively comes to the head of the MAIN-Q.

A request message or reply is always a 32-byte record with the following format



DSNO is the service number being kicked.

DACT is an activity number often used to indicate a sub-service where more than one type of request is handled on the same service number.

SSNO is the service number of the requesting servicing routine in case a reply is required.

SACT is its subservice number which is always returned (in DACT) in replies.

LINK is a chain link (see below).

All service request messages and replies are stored in a common list-processed area. At present there is space for 192 messages. The system maintains a free list and the MAIN-Q consists of a list of head cells to chains in this parameter area.

The restriction to 32 bytes was made to allow a consistent scheme for paged and non-paged processes. Requests from paged processes are made by placing the 32-byte record in the 4 × 64 bit floating-point registers and issuing the supervisor call (SVC) instruction. The floating point registers were chosen because they are the only registers common to all four P-states and it is not convenient to place SVC parameters in memory in a paged system. It has been found that this restriction to 32 bytes has rarely been an embarrassment.

Each director and each user process also has a service number. Messages for paged processes are held in the same area but the head cells are not on the MAIN-Q and are held separately.

A useful feature of the parameter passing scheme is the *service exchange*. This has a table which relates service numbers to the servicing routine or paged process. By changing this table servicing routines can be moved around the system, e.g. between supervisor and demons, without the problems of changing all occurrences of the service number. Closely connected to this is a facility which enables selected services to be monitored. A printout is produced showing all messages entering and leaving the associated servicing routines. This is a most useful diagnostic facility as it enables software errors to be located very precisely.

Servicing routines normally deal with groups of associated services. For example there is a DRUM routine which handles requests for transfers and termination interrupt messages.

The IMP language in which the system is written allows the use of *own* variables and this feature is used to store information between one call of DRUM and the next.

Because all servicing routines return to the despatcher to wait, it is necessary to have one stack only for all the routines of supervisor—indeed the organisation of the routines has been made with this in mind.

It is perhaps worth remarking that servicing routines often have at their head a jump to a switch label (vector label) controlled by an *own* variable, set on the previous entry.

EMAS also has a conventional semaphore scheme for the

Table 5 Priority ratio table

PRIORITY	FREQUENCY
1	21/32
2	8/32
3	2/32
4	1/32

Table 6 Time used by supervisor

EMAS VERSION 729 DATE: 24/08/72 TIME: 08.02.00 UNTIL 16.01.46

METERING INFORMATION
IDLE TIME(SECS)= 14686

SERVICE	COUNT	TIME	MUSECS	FUNCTION
3	1046	1	706	OPERATOR CONSOLE INTERRUPT
4	14352	15	1019	SLOW DEVICE INTERRUPTS
6	343010	553	1611	DRUM INTERRUPTS
7	35058	41	1159	REPLACEABLE DISC INTERRUPTS
8	413388	341	824	DISC-FILE INTERRUPTS
9	73443	288	3924	COMMUNICATIONS MULTIPLEXOR INTERRUPTS
26	2936	3	911	SLOW DEVICE REQUESTS
27	2885	7	2381	DEVICE LOAD-UP REQUESTS
28	23513	16	666	ALTERNATE NUMBER FOR REPLIES
29	981823	420	427	DRUM REQUESTS
32	14558	18	1244	
33	498	1	1305	REPLACEABLE DISC REQUESTS
34	2876	4	1327	
38	139990	132	941	DISC-FILE REQUESTS
39	3144	3	1073	
43	632	1	1575	MESSAGE TO OPERATOR CONSOLE
44	816	2	2213	PROCESS STARTUP ALTERNATE NUMBER
50	917637	637	693	CPU DESPATCHER REQUEST
53	265	1	4452	SCHEDULER REQUEST
54	50950	30	583	SEMAPHORE REQUEST P OR V.
55	133921	481	3588	COREGIVE
57	655654	320	487	PROGRAM ERROR OR SVC REQUEST
58	684757	353	515	ALTERNATE NUMBER COREGIVE
59	390707	468	1198	PAGETURN INTERRUPT
60	1338	1	449	
61	13535	9	651	
63	46819	386	8248	CORETAKE
64	374406	164	438	ALTERNATE NUMBER CORETAKE
67	8423	5	639	CORE ALLOCATION FOR BUFFERS
69	37315	22	592	SCHEDULER REQUEST
70	127525	92	724	SCHEDULER REQUEST
72	11497	10	832	MAGNETIC TAPE CONTROL FUNCTION REQUEST
73	2566	2	795	ACTIVEGIVE
74	7689	4	580	ALTERNATE NUMBER FOR REPLIES
77	1954	7	3635	ACTIVETAKE
78	34356	13	391	ALTERNATE NUMBER FOR REPLIES
79	2559	2	706	FROMACTIVE
80	55405	69	1241	ALTERNATE NUMBER FOR REPLIES
89	3168	2	636	TIME OF DAY REQUEST
97	277	1	2346	PROCESS STARTUP REQUEST
101	485	1	2340	
102	434	1	2504	
103	274	1	4069	
104	255	1	3683	
105	253	1	3664	
106	11489	34	2994	USER CONSOLE INPUT/OUTPUT REQUESTS
108	38014	73	1909	
109	248	1	3189	
110	248	1	3358	
111	268	1	3669	
119	956	1	607	CONSOLE BUFFER ALLOCATION
120	1184	1	548	
126	1440	1	779	SLOWDEVICE REQUESTS
127	9751	8	795	
128	2865	1	219	SUPERVISOR MONITOR OUTPUT

TOTAL TIME 5052

Table 7 Use of CPU time

EMAS VERSION 729 DATE: 24/05/72 TIME: 12.02.00		
TIME IN USER PROCESSES	6048	31.43
SUPERVISOR TIME CHARGED	2613	09.07
SVC'S	1311	04.55
PAGETURNS	1302	04.52
UNCHARGED SUPERVISOR TIME	2439	08.47
IDLE TIME	14686	51.01
TOTAL TIME	28796	100.00
ANALYSIS OF SUPERVISOR TIME		
VIRTUAL MEMORY SUPPORT		
DRUM TRANSFERS (6,29)	973	03.38
DISC TRANSFERS (7,6,32-41)	539	01.87
CORE LOADING (55-6,55-9,63-4)	1852	06.43
DRUM LOADING (73-58)	98	00.34
SCHEDULING OF PAGED PROCESSES	115	00.39
TIME SLICING (50)	637	02.21
FILE SYSTEM SUPPORT (54,35-6,60-1)	40	00.13
SVC PARAMETER PASSING (57)	320	01.11
COMMUNICATIONS SUPPORT (7,100-15)	403	01.39
DEVICE POLLING (27-8)	23	00.07
MAGTAPES (5,45-8,72)	10	00.03
MISC.	42	00.14
EMAS VERSION 729 DATE: 24/05/72 TIME: 16.01.46		

synchronisation of directors requiring common access to file indexes. The P and V operations on the semaphores associated with the indexes are implemented by a servicing routine which forms queues (if necessary) when it receives P-operation requests and sends replies when appropriate, e.g. on receiving a V-operation message.

Process synchronisation in EMAS is based on the parameter passing scheme which can be thought of in terms of P and V operations on private semaphores. P and V operations on resource semaphores can be implemented trivially in terms of the parameter passing scheme. In a system where the P and V operations are basic, a parameter passing scheme could be implemented trivially using private semaphores. The two schemes are dual.

Device handling routines

This section gives a brief description of the supervisor servicing routines which handle fast devices—drums and discs. The communications device routine is described in the paper by Rees (1973) and the slow device routine in the paper by Hayes (1973).

Drums and discs are currently handled by two separate routines. Each is activated in two ways, either as a result of a request for a transfer to be done or as a result of the occurrence of an appropriate channel termination interrupt. The normal request is for a page to be transferred between core and the device. Provision is made for read, write and write-check* transfers. If the device is idle a command chain is created and initiated immediately on receipt of a transfer request. If the device is busy the requests are queued by the routine which inhibits its request service number when its queue becomes full. When the termination interrupt occurs and the routine is

*The write and check read facility is used when there is doubt about the condition of the drum or disc hardware.

Table 8

```
EMAS VERSION 729 DATE: 24/08/72 TIME: 13.35.18

QUEUE SAMPLING INFORMATION


NO. OF TIMES QSAMPLE KICKED WAS      865

ITEM          TOTAL     MAX     MIN
RUNQ            646       5       0      RUN QUEUE
ACT STRQ          1       1       0      AWAITING DRUM SPACE
PROCTF         3688       7       1      FREE SEGMENT TABLES (OUT OF 8)
CORE Q1         921       3       0
CORE Q2         542       6       0
CORE Q3         481       5       0
CORE Q4         275       5       0
COREL         39894     149     -33      UNALLOCATED CORE PAGES
COREF         61693     131       5      UNUSED CORE PAGES
AS FREE      405842.     904     116      UNUSED DRUM PAGES
ASUNUSED     396580      951     102      UNALLOCATED DRUM PAGES
BPT UNUS     132337      307      36      UNALLOCATED BLOCK PAGE TABLES
BPT FREE     133313      287      65      UNUSED BLOCK PAGE TABLES
ACTT  Q         239       5       0      AWAITING MOVEMENT TO DISC
DRUM INH        217       1       0      DRUM BUSY
PT FREE       38494      61      10      FREE PAGE TABLES( OUT OF 64).
MAX PARM SPACE     126                   MAXIMUM NUMBER OF PARAMETES QUEUED

EMAS VERSION 729 DATE: 24/08/72 TIME: 16.01.46
```

called by the despatcher using its alternate service number, it sends replies in respect of completed transfers, sets up a new command chain and uninhibits its request service number to allow queueing of further requests.

The drum routine currently has space to queue up to twenty requests in total and sets up command chains of up to eleven page transfers. There are four sectors round each track and there are separate queues for each sector.

For each disc a single queue ordered by cylinder number is kept. It is serviced first in the forward direction and then in the reverse direction to reduce movement of the disc arms. An analysis of the above routines and of new ones which will shortly replace them will be reported on in due course.

The paging system

Files reside permanently on the disc-file and pages of files are moved between the disc-file, core and drum during the execution of user processes.

User processes do not make file-access requests but instead refer to virtual memory addresses onto which files have been mapped. The paging software moves the relevant pages around the storage hierarchy as required.

The director connects files to virtual memory by writing information on the master page. Small files are laid out on the disc-file in contiguous pages but a file of more than 16 pages in length is broken into 16 page blocks plus a shorter block of pages for any remainder. One 16 page block is not necessarily next to its successor block. On the master-page there is a table which relates segment numbers in virtual memory to physical disc-file addresses.

For each segment there is specified:

ACCESS MODE: no access, read/read-write, shareable/non-shareable.

LENGTH: length of block, i.e. number of valid pages in this segment.

DISC ADDRESS: disc address of first page of the block.

AMX: index into the active memory table.

The paging routines operate from this table (the claimed block table—CBT) and have no knowledge of files as such or to whom they belong. Because this table and other necessary tables (see Fig. 5) are quite large (about 3,000 bytes), the master page is paged with the process. However, it must always be in core while the process is receiving service from the CPU.

When a process which has its pages entirely resident on the disc-file becomes active, it must first wait until it can be allocated an initial amount of drum space. Its master page is then copied to the drum and the process waits for an allocation of core. When it is allocated core its master page is moved to core and execution begins. This causes page demands and further pages are brought in. As pages come in entries are added to the core memory (CMT), core map (CMAP) and core position (CPT) tables. Every so often the read-write markers on the core pages belonging to a process are examined and cleared and entries made in the USAGE field of the core map. If it appears that pages are no longer being used they are removed from core to the drum. In this way the size of the core working set is reduced. If a process goes to sleep (e.g. waits for console input) or is removed from core for other reasons, the core working set can, if appropriate, be retained and the pages in it pre-loaded the next time.

The segment and page tables which drive the hardware have to be set up for processes in core from the other tables. It is worth noting that the core map and the core position tables contain all the information needed to do this. In fact alternative paging hardware operating directly from these tables is easy to

Table 9 Category table transitions

EMAS VERSION 729 DATE: 24/08/72 TIME: 02.02.09 UNTIL 16.01.46

CATEGORY TABLE MOVEMENT

From \ To	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	286	0	0	0	6	0	2	232	0	0	8	0	5	2	0
2	0	301	223	0	0	0	0	0	0	0	0	0	0	0	0
3	0	203	3297	201	0	0	0	0	0	0	0	0	0	0	0
4	0	21	162	176	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	3732	0	0	369	0	0	0	0	0	0	0
6	0	0	0	0	0	0	4	0	8	0	0	0	0	0	0
7	0	0	0	0	0	0	5	0	0	12	0	0	0	0	0
8	0	0	0	0	559	0	0	22613	41	0	2819	0	0	0	0
9	0	0	0	0	2	0	0	16	517	57	0	53	0	0	0
10	0	0	0	0	1	0	1	36	0	521	0	0	190	0	0
11	0	0	0	0	226	0	0	1962	79	0	4637	69	0	1198	0
12	0	0	0	0	9	0	0	19	0	10	38	19	27	19	0
13	0	0	0	0	0	0	1	8	0	41	117	0	152	0	149
14	0	0	0	0	54	0	2	291	0	81	606	0	74	738	109
15	0	0	0	0	1	0	0	8	0	27	202	0	20	0	527

envisage. An associative memory large enough to contain extracts from the core map and core position tables of the current process would be needed. At present we restrict these tables to 63 entries.

To enable the paging routines to perform preloading operations where appropriate, extracts of the claimed block table are kept permanently in core. This table (the active memory table—AMT) has space for extracts concerning 32 blocks but is not always full. The entry specifies the block disc address and has a 16 bit mask to specify which pages of the block are in the working set and therefore eligible for space on the drum. When the pages of the working set are actually allocated space on the drum the disc address is changed to a pointer to a block page table (BPT) which has further pointers to the active store table (AST). Here there is one entry for each drum page. In this entry there is a core address if the page is allocated core. There are also flags which specify whether the page is in core or on the drum, and whether a page on the drum has been changed since it was last on the disc-file.

In the case of files connected in a shareable mode there is an extra level of indirection from the active memory table of each user process through a common shared active memory table (SAM) and then to the block page table entry. In this way all references to a shared file use the same disc, drum or core page as appropriate. There is also a table (the page in transit table—PIT) which has chains of processes awaiting the arrival of a shared page.

When a page fault interrupt occurs, i.e. reference is made to a virtual address where the corresponding page is not in core or the page table is not up to date, the supervisor accesses the claimed block table to check that access is valid. If the AMX field is non-zero there is an extract in the active memory table, and supervisor works down the chain to discover where the latest version of a page is situated. It will then demand-page it from the disc or drum, or, if it is in core already, complete the page table. If the page is not in core but already on its way (for some other process) an entry is made in the page in transit table.

The paging is handled by a number of servicing routines.

ACTIVEGIVE	which moves the master page from disc to drum,
COREGIVE	which pre-loads the current core working set from drum to core,
PAGETURN	which services demand-page requests,
CORETAKE	which removes unused pages or complete working sets, and re-computes the working set, and
ACTIVETAKE	which moves pages from drum to disc.

In general pages leaving core go to the drum. However a process which is designated for removal to disc can have its unshared pages moved directly to disc.

Scheduling

There are two levels of scheduling decisions in EMAS, firstly decisions as to whether processes should be allowed to start and secondly decisions on the allocation of drum space, core space and CPU time to active processes. The first level is fairly simple. The machine operator determines the number of users who are allowed to log on at any one time and the number of background processes permitted.

This description concerns itself with the second level. Initially we will ignore the question of allocation of drum space and look at the core and CPU allocation. The basic aim of the scheduling scheme is to keep the hardware acceptably busy whilst at the same time providing adequate response for interactive activities. This is achieved by having a suitable mix of processes in core at any time. Some will be allowed to stay

Table 10

Q LENGTH DISTRIBUTIONS

	USERQ	ASQ	CQ1	CQ2	CQ3	CQ4	SLPQ
0	1	2557	14880	5454	1825	915	14
1	1	5	3036	1232	360	160	22
2	1	1	2117	559	190	88	494
3	1	0	1369	303	102	51	9420
4	1	0	824	168	67	27	388
5	1	0	512	79	29	11	840
6	3	0	286	27	11	1	1451
7	6	0	136	11	6	0	1355
8	6	0	55	5	0	0	1296
9	4	0	23	2	0	0	1448
10	13	0	19	0	0	0	1590
11	8	0	9	0	0	0	2178
12	6	0	4	0	0	0	2528
13	5	0	3	0	0	0	2986
14	10	0	2	0	0	0	2954
15	6	0	2	0	0	0	2638
16	16	0	1	0	0	0	2053
17	21	0	1	0	0	0	1505
18	32	0	0	0	0	0	959
19	23	0	0	0	0	0	523
20	13	0	0	0	0	0	284
21	13	0	0	0	0	0	155
22	13	0	0	0	0	0	89
23	19	0	0	0	0	0	61
24	16	0	0	0	0	0	55
25	12	0	0	0	0	0	29
26	11	0	0	0	0	0	6
27	8	0	0	0	0	0	0
28	2	0	0	0	0	0	0
	272	2563	23279	7840	2590	1253	37321

	RUNQ	LMBQ
0	292932	102575
1	161151	103387
2	93698	78107
3	24204	18126
4	2238	1224
5	93	58
6	3	10
7	0	0
8	0	0
9	0	0
10	0	0
	574319	303487

JUDYX 1831
EMAS VERSION 729 DATE: 24/08/72 TIME: 16.01.46

for several seconds to keep the CPU busy, others will finish quickly so that their space can be re-used by other interactive processes. As processes change between interactive and non-interactive phases the system must react accordingly.

The core must not be overloaded with processes or thrashing will occur. We have already described in outline the way in which a core working set is established. The scheduling system uses the size of the working set to make an allocation of core, and it also makes an allocation of CPU time. At any instant every process is categorised according to these allocations. Its priority for loading to core depends on its category.

The system unloads from core processes which exhaust their allocations. Whenever a process is unloaded new allocations are made depending on the last allocation, the reason for unloading and the actual amounts of core and time used in the last period in core.

As there are invariably more jobs awaiting execution than will fit into core we have to have queues of such jobs. At present we

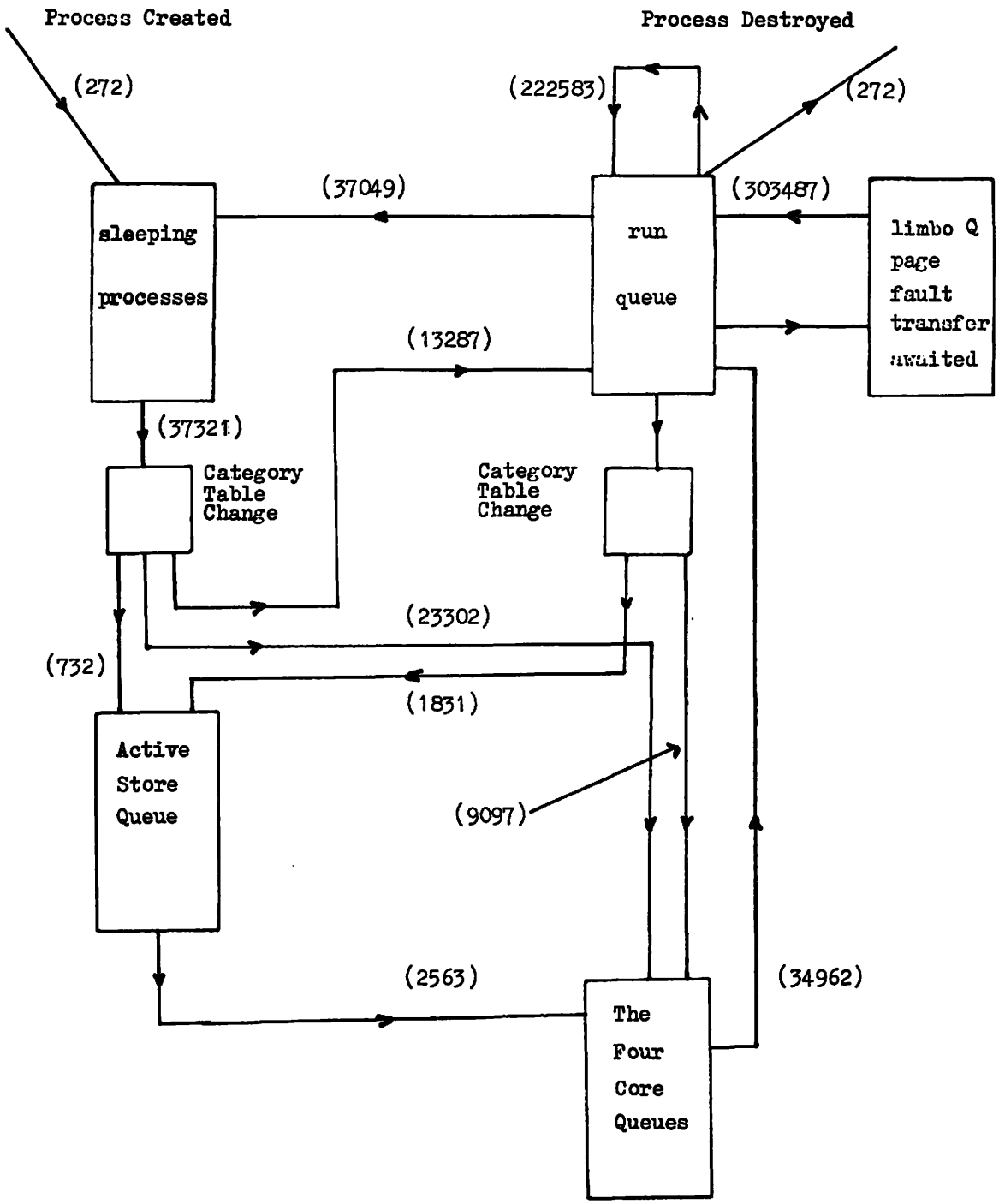


Fig. 6 Traffic Model 08.02.00 until 16.01.46

have four such core-queues, one for each of four priority levels. The priority level of a process is a property of its current category.

We use a simple algorithm to choose one of the four queues. The algorithm is such that the higher priority queues are selected more frequently than the lower priority queues in some defined ratio. Once a queue is selected (and provided it has a process waiting on it) the first process on that queue is chosen as the next process to be loaded. This is done as soon as enough core pages are free for its core allocation.

The whole working set is loaded at this time and the process then goes onto the run-queue where it takes its turn on the CPU. It may acquire extra pages on demand, and pages may be removed by periodic examination of its use of core pages. It is allowed to acquire pages providing its working set size does not exceed its core allocation. If this happens it is totally unloaded and takes its turn on the core-queues again.

Processes on the run-queue, i.e. those in core which are not awaiting the completion of a page-fault transfer, take turns on the CPU in a simple round-robin fashion. They are allowed no more than 30 milli-seconds in any one time-slice. Note, however, that they are not unloaded from core at the end of a time-slice but when they use their CPU allocation. In this way processes in core have a fairly equal share of the CPU. A short time slice is necessary to ensure that processes get back on the CPU quickly after a page-fault transfer is completed, so that those which are building up a working set by demand paging do so in a reasonably short elapsed time. When a process

exhausts its CPU allocation it is removed and put on one of the core-queues again. Otherwise a process leaves core only if it stops, or more commonly, if it is waiting for a slow device such as the user console.

Whenever a process is removed from core its working set is recomputed and it is given a new category dependent upon the reason for removal. For example, if it uses its predicted CPU-time then it is usually moved to a category with a larger time. This will mean it will get more time when it next comes to core but it may have to wait longer to enter core as the priority level associated with its new category is likely to be lower. When it next gets a turn its (full) working set is pre-loaded.

If the process exceeds its core allocation it is normally moved (next time it is loaded) to a category with a higher allocation but its working set is reduced to the master page. It is assumed that it is a well behaved process moving to a different area of virtual memory and that pre-loading of the existing working set would not be helpful.

The system also adjusts the core allocation downwards if a process has not used enough core by the time that it is unloaded to justify its present category.

Processes which 'go to sleep', waiting for a console response for example, are unloaded and moved to a category with the same core allocation but less time and therefore higher priority. When they wake up they go into the appropriate queue and are loaded quickly with a short residence time. Here it is assumed that interaction with the console could have changed the whole course of the computation so that good response is the current requirement. However the (full) working set is still pre-loaded to facilitate rapid execution of repetitive console actions like text editing.

The scheduling scheme has in practice turned out to be very effective and the demand-page rate is low.

In earlier versions of the system we also had similar arrangements for pre-loading pages from disc to drum and for re-computing the *drum working set*. In this case pages would be moved back to disc. This did not appear to justify the complication involved and was removed. However the tables are still set up for this more general situation, in case it is ever desired to revert to the former system.

The whole of the above scheme is driven from the category table*. For each category the following information is held:

PRIORITY	determines which queue the process goes on when waiting to be loaded.
CORE	the core allocation (number of pages).
RESTIME	the CPU-time allocation.
LOOKTIME	interval between recomputation of working set.
NCY 1	Category to move to if process runs out of core.
NCY 2	Category to move to if process runs out of time.
NCY 3	Category with next lower core size—zero if there is no such category.
NCY 4	Category to move to if process goes to sleep.

The actual values in use at present are shown in **Table 4.**

Scheduling of the drum space is simpler. Active processes accumulate pages on the drum and keep them unless they exceed a fairly generous limit or go to sleep (i.e. into a wait state). When a process goes to sleep it may be allowed to keep its drum space if it is in an interactive category or if the drum is not congested. Otherwise the pages of the process which are on the drum and have been changed since last on the disc are copied to disc and the drum space is released.

In congested conditions processes wholly on the disc have to wait for a drum allocation before they can be placed on the core-queues. However it is not efficient to run the system with too little drum space. At present we have 1016 drum pages, which is enough for 25–30 processes. Shortly we shall have another 508 pages added.

The paging and scheduling routines have diagnostic facilities built in to them which can be turned on from the operator's typewriter. Tracing can be performed in varying degrees of detail for a selected process or for all paged processes. The trace shows the elapsed time to the nearest milli-second of every paging and scheduling event and the composition of the core working set whenever it is pre-loaded. The trace gives the virtual addresses of the pages in use providing useful feedback when we are trying to optimise the paging behaviour of standard programs such as compilers, editors, etc.

Accounting

The system keeps full records of the CPU time used by each process, of the number of supervisor calls it makes and of the page-transfers performed on its behalf. These, together with records of input/output and of disc space in use, are used for resource accounting and charging purposes.

The charging formula used by ERCC at present is:

$$C = KR\,(T + S/500 + P/256) + U/128$$

where C is the charge in new pence,

K is a constant (currently K = 2),

R is a rate depending on the time of day (current values are R = 1, R = 0·8, R = 0·6),

T is CPU-time (in seconds) used by both user and director,

S is number of SVCs issued, by both user and director,

P is number of page transfers (including director pages),

and U is number of records read or printed.

File storage is charged at the following rates:

0·15p per page day for files with backup status.

0·075p per page day for semi-permanent files.

Measurement of performance

A full analysis of the figures we have collected must be the subject of another paper. However, we present some basic measurements, to indicate the types of information we collect. As we collect these figures whenever the system is running, the information must be very cheap to collect and there must not be too much of it to be examined afterwards.

We present in **Tables 6 to 10** a set of figures of the type produced at the end of every session. This particular session from 0800 to 1600 is regarded as a typical day-time session for the time of year. The maximum number of users during the session was 32. The system tends to be lightly used before 1000 and over the lunch period, which accounts for the high idle time. During term-time there would be more batch and detached work to be run in slack periods.

Table 6 shows the number of entries to servicing routines on each service number, the time used in total, and the average time per entry. For example, service 29 shows that 981,823 page transfers were made during the 8 hour period. The drum channel can handle 200 transfers per second so the channel is occupied for only 17 per cent of the time. Furthermore service 6 (the drum interrupt) has only 343,010 entries so the average number of transfers on a single chain is 2·86.

Table 7 which is derived from Table 6 gives an analysis of the use of CPU-time.

Table 8 gives the averages, maxima and minima of various system variables sampled every ten seconds during the last 146 minutes of the session. The figure of −33 is not an error, but a recent refinement in which the scheduler allows for pages being shared in core.

Table 9 is the transition matrix for category changes.

Table 10 shows the distributions of various queue lengths.

Fig. 6 shows the movement of processes around the system.

*This scheme is a development from a table driven scheduler for a somewhat simpler situation described by Livermore (1966).

Appendix 1

```
COMMAND :help

          THE FORMAT OF A COMMAND IS :
                <COMMAND> ( <PARAMETER> ) <CR-LF>
OR              <COMMAND> <CR-LF>
WHEN TYPING COMMANDS SPACES ARE INGNORED AND DOUBLE QUOTE CHARACTERS
MAY BE USED TO DELETE INCORRECT CHARACTERS AS FAR BACK AS THE
BEGINNING OF THE CURRENT LINE.
COMMANDS AVAILABLE ARE :
COMMAND             INFORMATION
ALERT         :   PRINT SYSTEM ALERT INFORMATION
APPENDLIB     :   ADD LIBRARY
BATCH         :   RUN BATCH JOB
CHERISH       :   MARK FILE FOR ARCHIVING
CLEAR         :   CLEARS DD DEFINITIONS
CONCAT        :   CONCATENATE SOURCE FILES
DDLIST        :   PRINTS OUT CURRENT DD DEFINITIONS
DEFINE        :   ASSOCIATE FILE WITH LOGICAL I/O CHANNEL
DELIVER       :   CHANGE DELIVERY INFORMATION
DESTROY       :   DESTROY A FILE
DETACH        :   SEND JOB TO BATCH QUEUE
EDIT          :   CONTEXT EDITOR
ENTRIES       :   PRINTS THE NAMES OF ENTRIES IN AN OBJECT FILE
FLIST         :   PRINTS OUT LIST OF FILES
FORTE         :   COMPILE FORTRAN SOURCE FILE
HAZARD        :   CEASE ARCHIVING FILE
HELP          :   USAGE INFORMATION
IMP           :   COMPILE IMP SOURCE FILE USING IMP(AA) COMPILER
INSERT FILE   :   INSERT OBJECT FILE INTO LIBRARY
LIBINFO       :   PRINTS LIBRARY CONTENTS
LINK          :   LINK OBJECT FILES
LIST          :   LIST A SOURCE FILE
LUNAR         :   DEMONSTRATION PROGRAM
METER         :   PRINTS METERING INFORMATION
PARM          :   SET COMPILING PARAMETERS
PASSWORD      :   RESET PASSWORDS
PERMIT FILE   :   PERMIT ACCESS TO OTHER USERS
REMOVE FILE   :   REMOVE OBJECT FILE FROM LIBRARY
REMOVELIB     :   REMOVE LIBRARY
RENAME        :   RENAME A FILE
RUN           :   RUN OBJECT FILE
SEND          :   LISTS AND DESTROYS SOURCE FILE
SET STREAMS   :   DEFINES MULTIPLE STREAMS
STOP          :   STOP SUBSYSTEM AND LOG OUT
USERS         :   NUMBER OF USERS ON SYSTEM
MORE INFORMATION ABOUT INDIVIDUAL COMMANDS MAY BE OBTAINED BY TYPING
THE NAME OF THE COMMAND AS THE PARAMETER TO THE HELP COMMAND
EXAMPLE : HELP(EDIT)
MORE INFORMATION IS AVAILABLE CONCERNING SYSTEM FACILITIES
UNDER THE FOLLOWING HEADS : -
ARCHIVE
FILES
INPUT
INTERRUPT
LIBRARIES
NEW JCL
PROMPT
SCHEDULE
EXAMPLE:  HELP(FILES)
A COMPLETE LINE PRINTER LISTING OF THE CURRENT 'HELP'
INFORMATION CAN BE OBTAINED BY TYPING:  HELP(.LP).
```

The numbers in brackets are counts of the number of movements on each path. For example, it can be seen that the mean interval between demand page requests is 94·9 ms. Demand paging accounts for only 26·7 per cent of all page movements.

During periods when the system is processing mainly batch and detached work and there is a continuous supply of work, the CPU-utilisation rises to about 85 per cent, of which about 10 per cent is supervisor time.

In comparing these figures with other systems it should be remembered that in a virtual memory system, all file accessing operations are performed on the user's behalf by supervisor.

Acknowledgements

In a project of this magnitude and duration many people and organisations have made contributions which we are unable to acknowledge individually. Particular acknowledgement is, however, due to D. J. Rees and S. T. Hayes who shared with us the design and implementation of the system and to G. E. Millard and P. D. Stephens for the principal subsystem work. Their contributions can be judged from the accompanying papers. R. P. Poole made a major contribution to the design and implementation of the paging software and to the early integration of the system. P. Bratley was co-designer of the scheduling system. C. Adams provided Tables 6 to 10 and Fig. 6.

Thanks are also due to J. G. Walker*, S. Michaelson and J. G. Burns for their constant help, encouragement and support.

ICL and the Ministry of Technology (now Department of Trade and Industry) shared the cost of the EMAP Project.

*Manager of the ICL staff of the EMAP Project.

References

ARDEN, B. W., GALLER, B. A., O'BRIEN, T. C., and WESTERVELT, F. H. (1966). Program and addressing structure in a time-sharing environment, *JACM*, Vol. 13, No. 1, pp. 1-16.

CORBATÓ, F. J., SALTZER, J. H., and CLINGEN, C. T. (1972). Multics—The first seven years. *AFIPS Conference Proc.*, Vol. 40, pp. 571-583.

CORBATÓ, F. J., and VYSSOTSKY, V. A. (1965). Introduction and overview of the Multics system. *AFIPS Conference Proc.*, Vol. 27, pp. 185-196.

DIJKSTRA, E. W. (1968). Co-operating sequential processes. In *Programming Languages*, F. Genuys (Ed.).

HAYES, S. T. (1973). The EMAS Demons, to be published.

LIVERMORE, F. G. (1966). A general approach to time-sharing algorithms for scheduling and control of computer resources. General Motors Corporation research publication.

MILLARD, G. E., REES, D. J., and WHITFIELD, H. (1973). The Standard EMAS Subsystem, to be published.

REES, D. J. (1973). The EMAS Director, to be published.

STEPHENS, P. D. (1973). The IMP language and compiler, to be published.