The EMAS Director

D. J. Rees

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

The EMAS (Edinburgh Multi-Access System) Director is the paged part of the EMAS operating system program. A description of its position in the EMAS system, its functions and its implementation is given with particular reference to its main components, the file system and the console input/output system.

(Received January 1974)

The Edinburgh Multi-Access System (EMAS) is a general purpose time-sharing system for the ICL 4-75 computer. An overview of the system is presented in 'EMAS—The Edinburgh Multi-Access System' (Whitfield and Wight, 1973).

The EMAS operating system software consists of several distinct parts. There is a central part, known simply as the *supervisor*, which is permanently resident in core and which performs the time-critical functions of the system. This is also described by Whitfield and Wight (1973). The other parts are not permanently resident in core and are intended to implement the non-time-critical functions of the system. These parts are a number of system-owned processes which provide services for the system in general and the *director* which provides services for individual processes. The director is the subject of this paper.

Instead of having the usual type of supervisor overlay for non-core-resident parts, the director is made part of paged virtual memory and so is brought into core as and when required by the standard paging mechanisms. This also relieves the director of the problem of organising an overlay structure itself. Each EMAS user process has a director process of its own which occupies the first thirty-two segments of its virtual memory and which is paged in and out of core in the ordinary way. All but one of these segments are, however, shared among all the director processes. The remaining segments, 32 to 255, are available to the user process. Whereas when the director is active it may access the whole of that process's virtual memory, when the user process is active it may only access segments 32-255. This protection of the system is achieved by invalidating the relevant entries in the segment table for the process.

The effect of this organisation is for the director to provide a first level of processing of requests for system services from the user level. Such a request takes the form of a supervisor call (SVC) instruction issued by the user process together with a set of parameters which define the service required. When a request is made, the director of that process is activated and attempts to satisfy the request. For many of the services it is able to do so completely, but for others further requests for service to the supervisor or a system-owned process are necessary. These are also invoked by means of SVC but issued by the director.

The most important services provided for users by the system are those for file handling, input/output, (console input/output in particular) and for dealing with contingencies such as program failure, etc. Other services are concerned with entering jobs onto a batch queue, setting local time-limits, getting systems status information and so on. Normally, when a user process issues a service request, it is suspended until a reply is received. In the case of console input/output, however, it may sometimes be desirable to initiate a transfer, say, and to continue processing whilst the transfer is in progress. Facilities which allow this are also available. The file-handling services are implemented entirely by the director and the console input/output services by the director in co-operation with a supervisor servicing routine. Input/output service requests for devices such as card-readers, line printers, etc. are passed by the director to the demons process (one of the system-owned processes).

As a safety precaution, the EMAS system software resides on a replaceable disc unit (RDU). The supervisor is loaded into core from the RDU on start up but the code of the system-owned processes remains and is paged from the RDU as required. The director code which is shared by all the paged processes is that originally associated with the demons process and hence it is also paged from the RDU when any process requires it. In practice, of course, the director is in almost constant use overall and therefore is either in core or on drum storage all the time.

In common with the rest of the EMAS system software, the director was implemented using the IMP language, a language largely developed with this purpose in mind though also widely used by users of the system (see Stephens, 1974). The benefit of using a high-level language such as IMP cannot be overestimated. Only rare use of in-line machine code proved necessary, for example to issue SVCs. Since the system has been on the air, the system itself has been used to further the development of the director. This has also proved invaluable. For instance, it has been possible to decide on a change, make the change, recompile the director and try it out within the space of ten minutes.

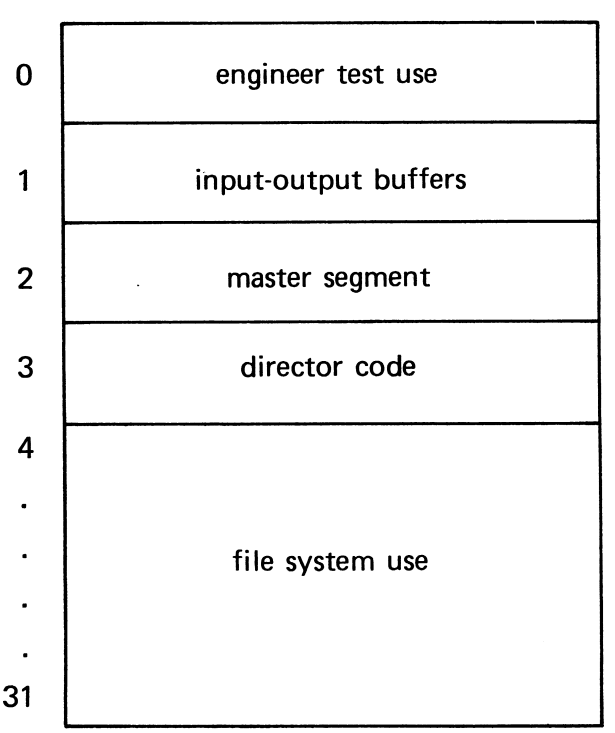


Fig. 1

Structure of the director

Fig. 1 shows the contents of the segments of virtual memory which are the concern of the director. Segment 0 is not used during the normal running of the system but can be made available to run engineers' test programs many of which originated from the non-paged ICL 4-70 and which therefore require low addresses rather than possibly high virtual addresses. Segment 1 is accessible by all directors in read-write shared mode and is used for input/output buffers. Input/output transfers take place from core using physical rather than virtual addresses and hence the pages involved have to be locked in core for the duration of the transfer. In practice, pages are allocated and locked in core for longer periods, for the total period the device is in use. The two equivalent addresses, physical and virtual, provide a convenient method of communication between the supervisor and the director. In the case of console input, for example, the routine in core initiates a transfer from a console into a buffer within the segment using its physical address and on completion, the director uses the equivalent virtual address to pass the input from there to the user area of virtual memory. This latter communication between the director and user area is a straightforward copy since they both occupy the same virtual memory. Since the *buffer segment* is shared among all the directors there is no protection from a director accessing an inappropriate buffer should it go astray. This has never caused any problem, however.

Segment 2, the *master segment* (see Whitfield and Wight, 1973), is the only unshared segment in the director and contains information relevant to the particular process alone. Page 0 is the *master page* which contains all the virtual address and physical disc storage address mappings used by the supervisor to perform the paging, together with other supervisor-used tables appropriate only to that process such as metering information tables. The master segment is also, by virtue of its read-write unshared mode, an appropriate place to locate the data areas of that process's director. The language IMP, in which the director was programmed, uses a static storage area and a dynamic stack at run-time. These areas are located from page four onwards. The gap between the master page and the data areas was created to allow for a possible extension in size of the master page. This has not, in fact, proved necessary. As an indication of the size of the director's data areas, pages four, five and six are the only ones commonly accessed during normal running. In physical terms, space is set aside for master segments, for all but the system-owned processes, on the large disc file and paged from there.

Segment 3 contains the code of the director. This consists of a loader, a module of code needed for running IMP programs, both quite short, and a main module linked together in the system standard way (see Millard, Rees and Whitfield, 1975). On initial entry, the loader satisfies external references between the two other modules and jumps to the main module. This main module is organised in such a way as to minimise page-faulting by grouping together the commonly accessed parts as far as possible whilst bearing the positions of page boundaries in mind. The IMP compiler allows the relative code address of each statement compiled to be monitored and this facilitates such an organisation, though improved versions of the compiler result in the need for minor reshuffling occasionally. The code of the director occupies approximately 11 pages of which only two or three are used at one time during the most common reasons for entry.

Segments 4 to 31 are used by the file system for disc file usage tables and file ownership indexes. This is described further below.

The initial entry to the main director module is at its beginning and thereafter, in IMP terms, the program consists of an indefinite loop which is never left. However, the loop contains an *exit* SVC to the supervisor which suspends its operations while waiting for a reply or when there is nothing for it to process. This is the normal exit point. What can be regarded as the normal entry point is the instruction following this SVC. The program there consists of an examination of the parameters provided by the new entry which indicate what function is required and a switch to the appropriate section of coding. To accomplish the required function the director may have to call upon a supervisor service such as manipulation of semaphores one or more times. For services which need no reply to the director, for example releasing a semaphore, the director may execute an SVC in line. In this case, the director is not suspended. For those services where the director needs a reply, after the SVC requesting the service has been issued the standard exit SVC is jumped to. This mechanism allows services which cannot immediately be completed to take place before a reply to reawaken the director is sent. Upon re-entry the director returns to the appropriate section of coding by making use of the reply parameters and status variables within the director. The reason for always using the standard exit point is that the supervisor queues requests and replies to the director in the order in which they appear. It is therefore possible for an entry to occur other than for an expected reply. Care has to be taken that no interference results.

The file system

File storage plays a central part in the use of EMAS and an important group of user services are concerned with the manipulation of files. The implementation of these services is contained entirely within the director apart from some subsidiary functions such as semaphore operations and file clearing and copying which are done by the supervisor. A user operates on files at a logical level referencing them by names of his own choosing and it is the function of the director to decide on the physical positioning of them on the disc used for their storage. It is then the function of the supervisor to access the disc areas when the user's virtual memory references dictate, using the virtual memory to disc address mappings set up by the director in the master page of the process when the files are *connected*. A file cannot be accessed through the paging mechanism until it has been connected, i.e. the mapping set up. Likewise, the file cannot be accessed after the file has been *disconnected*, i.e. the mapping removed.

Files are stored on a 700 million character disc which logically consists of two devices each with 350 million characters. During the initial development and usage of EMAS, one of these was sufficient to store user's and system files. This allowed a convenient method of development of the file system software by allowing each new version to be tested on the alternate device without the risk of inadvertent destruction of files in actual use. From the hardware reliability point of view it also proved invaluable. Increased availability and use of the system has now, however, dictated that the whole disc be used.

A file name consists of two parts joined by a '.' character. The first part is the user name on the system of the owner of the file and the second an arbitrary string of up to eight characters (excluding '.') to identify the file amongst his own. The total name is therefore unique on the system. For example, the user ERCC24 might have a file DIRFILE2. The full name would be ERCC24 DIRFILE2. Given permission, any user on the system can access the file by using that name. For those operations which only the owner can perform the second part of the name is sufficient, the owner name being implied.

Files have lengths which may be any multiple of a page, i.e. 4,096 bytes, up to a maximum of 1,024 pages. They are regarded from the system point of view as just collections of bytes which may contain any information the user desires. In particular, no distinction is made by the system between files containing binary machine code, source text, etc. It is the function of the

subsystem within the user's own virtual memory area to make any distinction that is required by means of header information in the file or by any other means. Many other file systems have made such distinctions and allowed such things as previous versions of updated files to be retained. The philosophy adopted on EMAS has been one of providing basic facilities which can be built upon by subsystems to provide more diverse facilities. An example of this might be the connect service, described below, which requires as a parameter the segment number in virtual memory to which the file is to be connected. A subsystem could remember which segments are in use and so be able to provide a connect service which found a free segment itself to relieve the user of the burden. The simplicity of the approach to the design also resulted in a certain orthogonality between the services making their functions as distinct as possible. For instance, a file cannot be destroyed if it is still connected in some virtual memory. There is no implicit disconnection defined in this case and it must be disconnected beforehand by a separate SVC. This gives a certain amount of protection from inadvertent destruction.

EMAS provides four possible modes of access to files when they are connected. These are 'read-only' and 'read-or-write' each of which may be 'shared' or 'unshared'. In a shared mode, any number of users may connect the file in their virtual memories (the same shared-read or shared-read-write in all, but at any virtual memory segment positions) whilst in unshared mode only one user, not necessarily the owner of the file, may connect it at one time. To be allowed to connect a file and hence to access it, the user must have been granted permission to do so by the owner of the file. The owner can set permissions for himself, for selected other individual users and for all users and can specify the level of access each should be permitted. The level specification takes the form of a four-bit value, the bits being: S_r S_w U_r U_w where S_r is shared-read, S_w is shared-read-write, U_r is unshared-read and U_w is unshared-read-write. Thus the value '1111' allows any mode of access, '1010' allows either of the read-only modes and so on. When a file is created, permission is set for the owner alone to have all modes. He can change this or permit other users by issuing an appropriate SVC. The access checking procedure operates in the following way. When a user issues a connect SVC his director may go through three possible steps. If the user is the owner of the file his permission is noted. If not, a list containing the names of users granted individual permissions and their access modes is searched and the appropriate information noted if the user's name is found. If neither condition holds, a temporary permission of zero is noted. In all cases, the permitted values noted are 'OR'ed with that set up for all users and the final permission results. At one stage during the development of the file system, an individual user could be singled out and given a lower access level than all other users but it was pointed out that this might be rather vindictive.

The owner of a file may request that it should be archived as a precaution against loss for any reason. Usage flags are maintained which indicate whether the file has been altered, or, at least, connected in a 'write' mode since it was last archived. The archiving system is described in detail in Wight (1975).

The ownership of files can be transferred between two users by making use of two services. The original owner of the file can *offer* his file to another user by using one service and then this user can *accept* the transfer. This two stage method was preferred to a single unconditional *transfer* service to avoid a user being given files by others without any control. File transferring is also the method by which users perform input/output to all devices but consoles. The demons process drives the physical devices. In the case of input, a file is created by demons, filled with the incoming data and then transferred to the appropriate user (indicated by the job document). In the case of output, the user creates a file of output and transfers it to demons which then outputs it to the requested device.

The file services available to a user are the following.

1. Create file. The user supplies a name for the file which must be distinct from those of his other files and the number of pages it is to be in length. The contents of the new file are cleared to zero both to maintain the privacy of the owner of the previous file to use that disc area and to check that that area of disc is undamaged.
2. Destroy file. A user can only destroy his own files and then only if not connected in any virtual memory.
3. Rename file. A user can rename one of his files if the new name is distinct and if it is not connected anywhere.
4. Change file size. A file can be increased or decreased in size by its owner as long as the resulting size stays within the 1 to 1,024 pages range. The file must either not be connected anywhere or at most connected in the owner's own virtual memory. In the latter case, either the additional pages are also connected (and cleared to zero) or the deleted pages (truncated from the high-address end of the file) disconnected. It may only be connected in the owner's memory because a director cannot change or modify the connect status of a file in a different virtual memory.
5. Set archiving status of file.
6. Get access permission. The file owner can ascertain what access permission he has granted to himself or to others.
7. Set access permission. Set the modes of access to be allowed to himself or to others.
8. Connect file. Connect a file to the user's own virtual memory from the specified segment position onwards in the specified access mode. If the file is more than sixteen pages long, i.e. a segment, extra consecutive segments will also be used in the connection as necessary. None of the segments must have a file already connected to them and the file must not already be connected somewhere else in the same virtual memory. The specified access mode must be allowed to this user for the file. If not, a fault flag is returned, but it does not distinguish between this fault and the non-existence of the file. In other words, the existence of a file with a given name is also regarded as private information in addition to its contents. The service call will also be rejected if the file is already connected in a different virtual memory and the required mode conflicts. That is, either the file is already connected in an unshared mode or it is connected in a shared mode which is not the one required or the required mode is unshared. No calls on the supervisor are involved in making the connection. The director simply writes the mapping information into the appropriate place in the master page from which the supervisor accesses it when the file is first referenced.
9. Disconnect file. The connection mapping in the master page is removed. In this case, a call on the supervisor is made to remove any pages written to in the file back from core or drum to disc. In fact, for reasons of consistency, the supervisor removes all the pages belonging to the process which have been written to back to the disc.
10. Change access mode of file. In an unshared mode, the mode can be changed from read to read-write or vice versa without disconnection. For a shared mode, the file must be disconnected and reconnected in the required mode (if it is possible).
11. Get file information. A user can get status information about any file to which he has been granted an access permission. The information corresponds to the contents of the descriptor of the file (described below). This contains such things as the length of the file, his permitted access

modes, the current connect mode, the number of shared connections and its archiving status.

12. Get file names. The user is given a list of the files he owns.
13. Get virtual memory map. A list of the names of all the files connected in the user's virtual memory and their positions is supplied.
14. Offer file for transfer. This is the first part of the transfer. The owner specifies the name of the user ('SYSTEM' for the demons process) to which he wishes to transfer the file. The file must be disconnected at the time and an effect of the offering is to inhibit the owner from using it further. The owner can revoke the offer before the prospective new owner claims it.
15. Transfer offered file. A file offered to this user is transferred to his ownership. The new owner can rename the file in the process of transferring it if he desires to avoid a clash with an existing name of his own.

Implementation of the file system

All information relating to file usage is stored on the same disc as the files themselves, though not in formally named files. Rather, an area at the start of the disc is reserved for the purpose. This area is mapped onto the virtual memory segments 4 to 31 of each process as it starts up in shared read-write mode by writing the appropriate information into the master page in the same way that a file is connected. Three kinds of table are contained in this area. They are the following.

(*a*) 'Page-in-use' bit tables. Storage on the disc is divided into page-sized units and each of these has a corresponding bit in a bit table. These are used to indicate whether that page is in use, i.e. is part of some file, or not. The appropriate bits are set whenever a file is created or extended and cleared whenever a file is destroyed or shortened.

(*b*) User name versus user number tables. For convenience of implementation, each user name has a corresponding unique user number. The advantage is that the number occupies a field of smaller width than the character string of the name and also that it can be used for indexing. These tables, which are hash-coded on the names, hold the correspondence between the names and numbers for retrieval when required. When a process starts up, the entry parameters contain both the name of the process user and the number. The table is consulted and if an entry is not found for this name, implying that it is a new user who is starting a process for the first time, his name and number are inserted. Clearly, the director need not and does not consult the table for its own user's number since it can be saved from the start-up information. The table also is not consulted for the demon process name 'SYSTEM' since this is always assumed to be user number 2. Thus, the majority of file service calls commonly do not involve consulting the table and thereby avoid a probable page fault. Only those calls involving other user's files cause the access.

(*c*) User file indexes. Each accredited user of EMAS has a file *index* which contains all the information, e.g. location on disc, size, etc, relating to his files. It is a page in length and has a virtual memory position given by his user number relative to the start of segment 4.

To maintain this consistent form of addressing, certain user numbers are not assigned so that the position can be used for the other types of tables. This structure imposes a limit on the number of users that may be accredited to the system, which is the number of pages in segments 4 to 31 less the number of pages used for the other tables, eight, i.e. 440. This number is regarded as sufficient for the immediate future. The restriction could be removed either by moving the boundary between director and user area or by mapping onto virtual memory only those indexes belonging to users who are either currently running on the system or who have files currently in use by others.

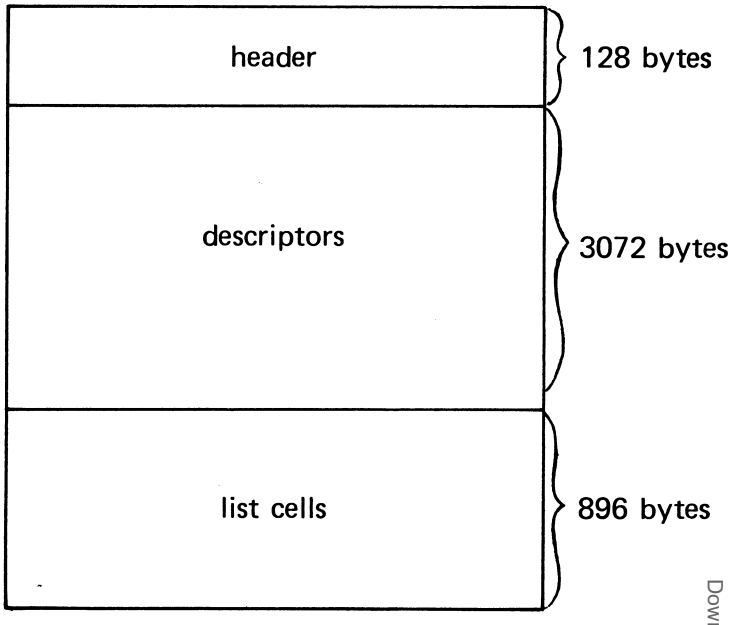


Fig. 2

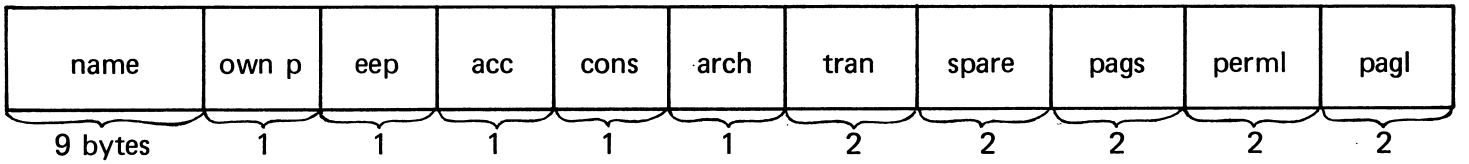
Fig. 2 shows the layout of each file index. The header section contains:

1. the name of the index owner
2. the number of free cells on the free list in the list cells sections
3. the head of the free list of cells
4. a count of the file-pages this user owns for accounting purposes
5. a set of *subsystem file identifiers.*

The use of the list cells is described below. The subsystem file identifiers are used by the director when the process starts. When a process starts, the only virtual memory areas connected are the director code and the master segment. The director makes further connections itself before any processing can be done. Firstly, the segments 4 to 31 are connected and then two files in the user's area starting from segment 32. These are intended to be a subsystem code file in shared read-only mode and a file for it to use as a data area in read-write unshared mode. Their names are the subsystem identifiers stored in the header. When the director has completed these connections, it transfers control to the start of the code file and indicates where the data file is connected (since the code file may vary in length). The identifiers are initialised to the names of a standard subsystem and a data file when a user is first accredited to the system and thereafter he can change them to whatever files he chooses by using a service provided for the purpose, so that next time he starts up the new files are used. In the main, this feature has only been used in the development of new versions of the standard subsystem though it is generally available.

The descriptors section of the index contains a 24-byte descriptor for each file owned by the user. The contents of each descriptor are shown in **Fig. 3.** The position of the descriptor within the descriptor area is given by a hash on the file name. The information in the descriptor area is referenced and updated as services are performed on the file. The layout of the index imposes a limit of 128 files per user. This limitation was accepted to retain the convenience of a page-sized index.

The list cells area contains 224 cells each of four bytes. These are initially formed into a free list from which cells can be



name: character string containing name of file

own p: owner's own access permission to file

eep: everyone else's access permission to file

acc: mode of connection of file

cons: number of connections of file

arch: archiving status and usage bits

tran: user no. of user to whom file offered for transfer

spare: reserved for future use

pags: length of file in pages

perml: link to list of cells for individual access permissions

pagl: either, link to list of cells for disc positions of sections of file, or, position of single sections of file on disc

Fig. 3

taken when required and to which they can be returned when no longer needed. The links are page relative so as to fit in a two-byte field. The cells are used to contain one of two sorts of information, either starting page numbers on the disc of the various sections of files (**Fig. 4**(*a*)) or access permissions for individual users (**Fig. 4**(*b*)). The boundary between the descriptors and bit cells sections was fixed at what was found to be a suitable value for an average user. The position could have been made dynamic to suit more diverse needs but the cost of more complication was not felt to be justified.

The file storage disc consists of disc surfaces on two separate rotating spindles. One of the two devices occupies the two top halves of the spindles and the other device the bottom halves. Cylinder addressing alternates between the spindles with even cylinder numbers on one and odd cylinder numbers on the other. There are 1,024 cylinders on each device, i.e. 521 per device spindle or quadrant, and each cylinder can hold 80 pages. To guard against failures of the disc hardware it was decided to make use of this separation. Each user on the system is assigned to one of the quadrants (from the range in which his user number lies) and all the files belonging to him reside there. The files on each quadrant are also archived separately so that in the event of a partial disc failure involving only one quadrant, such as a head crash, only those files need be restored. The system can also be restored to full use in one quarter of the time. Each quadrant has its own bit table and user name versus number table. With 40,960 pages potentially available for allocation to files on each quadrant, each bit table occupies a page and a quarter. The remainder of the second page is used for the name versus number table. The user file indexes are also organised so as to reside on the appropriate quadrant. In addition to these tables and indexes in the reserved area at the start of the disc, space is also reserved for the master segments of processes (other than the system-owned processes). These are dynamically allocated to processes as they start and are not related to the user number. Allowing for a possible 63 processes, this reduces the total number of pages available in each quadrant by 720. The corresponding positions in the bit table are made use of by the storage allocation algorithm.

The director has the facility to use the disc only from a given cylinder onwards. During the development of EMAS this was made use of to allow the manufacturers' J-level operating system some disc space. It is also a safety feature should any cylinder reserved for indexes, etc become permanently unusable.

As files can be accessed by users other than their owners, precautions have to be taken to guard against mutual interference when file operations take place. To this end, each file index has a corresponding semaphore (Dijkstra, 1968), upon which *P* and *V* operations are performed by calling on a supervisor routine. Whenever a director wishes to access a file index, it first claims the associated semaphore with a *P* operation. If the semaphore is not already claimed, control is returned immediately and the director will proceed to access the index and to release the semaphore with a *V* operation on completion. If the semaphore is claimed, the supervisor queues the request and suspends the director, i.e. does not send a reply, until its turn in the queue for the semaphore comes round.

The bit tables and name versus number tables also each have a semaphore which must be claimed before accessing them. An exception is made in the case of the name versus number tables in that the semaphore need only be claimed when updating a table, i.e. when a new user comes on since any concurrent look-ups will not mutually interfere and the updating is done in one instruction.

A simple list-processing scheme is all that is required to implement the semaphore routine in the supervisor. 'Deadly embrace' situations are avoided by allowing the director to claim only one semaphore at once. This is a more stringent rule than is actually necessary but it does not cause any difficulties since most of the services only involve one semaphore. For those that involve two, it is occasionally necessary to release a semaphore and reclaim it later after a different semaphore has been used to follow the rule. The simplicity of the rule, however, makes for easy checking.

The amount of space in the file indexes and in the master page for mapping tables precludes a completely general storage

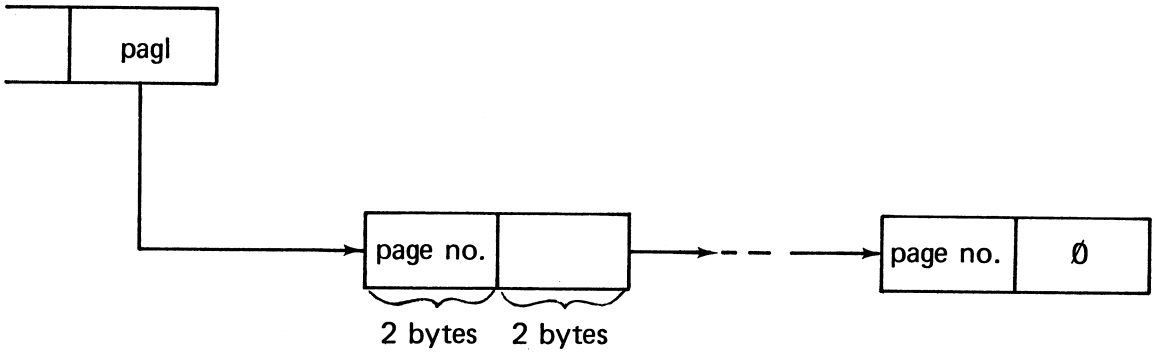


Fig. (4a)

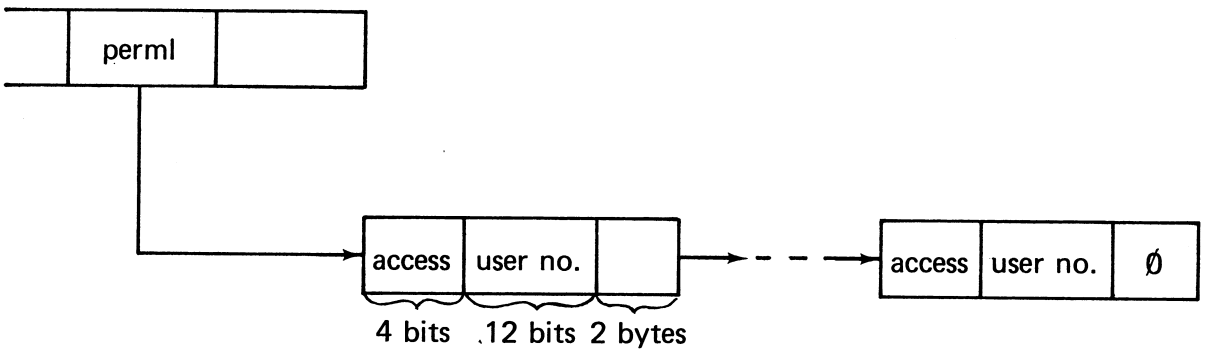


Fig. (4b)

allocation system in which any page of a file can be anywhere on the disc. The space is kept to manageable proportions by insisting that the pages of a file connected to a segment of virtual memory should be located in consecutive disc page positions. The only mapping required therefore is that for the first page of each segment. The remaining positions are found simply by adding the page in segment number. Overlap from the end of one cylinder to the beginning of the next but one (to maintain storage within a quadrant) is allowed for in the disc accessing routines. This scheme will tend to improve disc accessing for the many files which will be accessed sequentially, though the multi-programming will dilute this attribute. No consecutive areas of disc storage greater than sixteen pages are required. A file of more than sixteen pages will be divided into sections, all sixteen pages in length excepting possibly the last which will be whatever remaining number is required. Likewise, a file of less than sixteen pages will be all in one section.

These groups of pages are allocated by a search of the bit table for a long enough sequence of zero bits. The efficiency of the search algorithm depends to a large extent on the way the user or the subsystem he is using creates, extends and shortens files. In particular, when a file is extended, if its length is not a multiple of sixteen, the last incomplete segment will require more consecutive disc pages. An attempt to do this in situ is made, but if the required pages are already allocated, a complete new set of pages of the right length must be found, the existing part copied to the new area, the remainder cleared to zero and the old space deallocated. When a file is shortened, if the last segment is incomplete, this remains in situ and the truncated pages are deallocated. The more time-consuming and complex possibility of finding a group of pages in a hole of the right or more nearly the right size (a 'best-fit' algorithm), and copying into it, was not attempted. The result is that it is possible to make inefficient use of the disc, for instance by creating files of sizes in multiples of sixteen pages and then truncating to an exact length later. Small unusable holes will proliferate. The preferred strategy would either be to create files of the right length immediately or to create small files and extend them when necessary. The latter may result in excessive copying.

Simulation studies were carried out to find a suitable algorithm for the expected usage. Speed is a primary requirement in addition to a solution to the fragmentation problem. The algorithm this resulted in operates in the following way. Rather than searching the bit table for a hole of the right length from every bit position onwards, which will be very slow when large holes are required, the positions from which comparisons start are related to the length of the hole required. For a sixteen page hole, the positions are chosen every sixteen bits and so on for hole sizes which are a power of two. For intermediate sized holes, the positions relating to that of the next higher power of two are used. The least fragmentation was found to occur if the searches always start from the beginning of the bit table but the loss of speed of this method was unacceptable. Therefore, a compromise was adopted in which the search always starts from the next position beyond where the last hole of this same size was allocated and continues cyclically from there. Although the turnover in file-page usage is quite high, the benefit of compaction of disc use toward the beginning in terms of head movement is helped by initialising the search positions to the start of the bit table whenever EMAS is started up. The resulting algorithm is a variation of the 'Buddy' allocation system which is not directly applicable because of the large amount of space involved.

Whenever EMAS is started up, the bit tables are recreated and a consistency check performed on the file indexes. Initially, the bit tables are cleared to zero. Each file index is then inspected in turn. The user name in the header of the index is checked against the entry in the name versus number table and then each file descriptor is examined. The list containing the disc positions of the file pages is inspected and checked to be consistent with the file length in the descriptor. In the process of doing this, the bits corresponding to the pages of the file are set in the bit table. If any of the bits are already set, this implies that two files claim to have been allocated pages in common and a fault is signalled. This is conceivably possible when a crash has terminated the previous session and the latest

copies of the file indexes may not have been written back to the disc. Also allowing for possible previous crashes, the connect mode and number of connections fields in the descriptor are cleared to zero.

This initialisation can be inhibited, for example if access to the disc is temporarily to be avoided or if a complete reload of files from backup tapes is to be performed (Wight, 1975). In the latter case, the preliminary clearing down of a quadrant of the file system prior to recreating the files consists of clearing the bit table to zero and setting each file index to an empty status. The name versus number table is also recreated from archived information.

The console input/output system

Console input/output differs from other input/output in that communication with the user process is direct rather than through the file system. The user process controls it by means of SVCs to the director which co-operates with the supervisor to perform the required services. The equivalent of the various device controllers in the demons process is therefore part in the director and part in the supervisor in this case, The director part, being entered first when a service call is made, performs all the validity checking of the parameters and deals with the input/output in terms of streams. The supervisor part deals with physical input/output and initiates transfers through the multiplexor.

The console system also provides interrupt and prompt facilities. Up to sixteen consoles may be coupled to a single process for concurrent usage. Consoles are operated in an echo-plex mode when possible. That is, input characters are echoed to the printer mechanism from the multiplexor rather than direct from the keyboard. This allows confirmation of the successful receipt of the input by the multiplexor. A non-echo mode is also available.

A user gains access to EMAS either by dialling up on a modem telephone connection or simply by switching on if he is using a dedicated telegraph circuit. Pressing any key on the keyboard produces a hardware interrupt from the multiplexor when the line is in its dormant state. This is directed to the appropriate part of the supervisor to initiate the login procedure. The user is requested to type his system name and his password, which is not echoed. The demons process also has the function of validating names and passwords. If it succeeds the supervisor is requested to start up a process for the user. Since the same mechanism is used to log in additional consoles to a process, the supervisor first checks that no process is already in existence for this user. If one is in existence, no further process is started up. In both cases, the presence of a console is made known to the process by passing suitable parameters to the user level via the director. During the passage through the director, the console identification number which is one of the parameters is noted so that any future requests to use consoles can be verified as relating only to those consoles which have been logged in to that process. Since the input/output services use stream numbers rather than console numbers (which may vary from run to run) for convenience, service calls to associate console numbers and stream numbers must be issued before any other calls. The services available are:

1. Couple input. Associate a console with an input stream number
2. Couple output. Associate a console with an output stream number
3. Get input. Get a block (defined below) of input on a specified stream and place it in virtual memory from a given address onwards
4. Set input request message. The prompt facility described below
5. Put output. Output on a specified stream the specified text which is in virtual memory from a given address onwards
6. Input available? Query whether any input is ready for a 'get input' call. If a 'get input' call is issued when no input is available the process will be suspended until there is some
7. Output possible? Determine how much output can be requested without having the process suspended, i.e. how much buffer space is available?
8. Kill input. Cancel the previous 'get input' request
9. Kill output. Terminate the 'put output' transfer
10. Decouple input. Dissociate an input stream from a console
11. Decouple output. Dissociate an output stream from a console
12. Logout console. Detach a console from the process.

The logout service does not stop the process. A separate service is provided for that purpose. This allows consoles to be logged onto and off from a running process as required. It follows that there is no distinction between what are sometimes known as 'foreground' and 'background' jobs since either can become the other at any time though they may have different running characteristics.

Input/output operations use segment 1, shared among all virtual memories, as a communication region. Pages are allocated for this segment and locked in core as required by the demands of console usage. The supervisor accesses them by physical addresses and the director by virtual addresses. Each console has fixed buffers of 128 characters for input and output and they are used cyclically.

For input, the supervisor initiates a transfer from the console to the appropriate buffer and the characters are read in as they are typed. When a 'get input' request is issued, the director requests the position and extent of any waiting input in the buffer from the supervisor by means of an SVC. If input is available, a reply containing this information is sent back to the director immediately. If not, a reply is withheld and this effectively suspends the director (and hence the user level) from further activity. Only when a *block* of characters has been typed is a reply sent. This is clearly necessary to avoid unnecessary paging in and out of the process just to process single characters at a time. A block of input is intended to contain sufficient for waking the process to be worthwhile. It is defined to be a sequence of characters up to either a line feed, an end message (EM) or an end of text (ETX) character. The multiplexor has a very convenient feature which allows it to pass a hardware interrupt to the supervisor whenever one of these characters is typed without, except for ETX, terminating the transfer. The supervisor can therefore recognise when an input block is complete while the transfer is still in progress. Unfortunately, the interrupt does not indicate where within the transfer one of the characters has occurred so that a scan of the characters still has to be made to determine the extent of the block. In the case of ETX, the transfer then also has to be reinitiated. Unfortunately the multiplexor gives in addition interrupts for characters which have no significance for EMAS and these have to be ignored. If there is more than one complete block in the buffer when a 'get input' is issued, the total extent of all the blocks is indicated to the director. On receiving the information the director copies it into the position specified by the request and returns control to the user level.

The prompt facility is used with the 'get input' service. A string of up to fifteen characters can be set as a prompt message. This message is output to the console whenever a get input is issued and there is no input yet in the buffer, i.e. none at all rather than no complete blocks. Its use is therefore to indicate to the user what input the program expects from him. If the user does type ahead of the 'get input' request, the prompt

message is not output. This avoids the malordering possible when the standard output stream is used for prompting. The same message remains and is used for subsequent 'get input' requests until it is changed by a further 'set input request message' service. It can be set to a null string if no prompting is required. The basic command interpreter of the standard subsystem makes use of the facility. By setting the message to COMMAND: it can indicate that the subsystem is at command level and that it expects a command as the next input. If the user has typed ahead, it can only be assumed that he knows what will be expected and therefore the prompt is redundant. Any user program can similarly specify what input it wants by setting the message appropriately. The mechanism is convenient where input may come to a program either from the console or from a file since the prompts can be regarded as an additional output stream. In the case of input from a file, ordinary prompts would be quite spurious, whereas this mechanism allows them to be suppressed without having to change the program.

For output requests, the director transfers into the buffer from the virtual address specified and calls on the supervisor to output the buffer to the console. The process is allowed to continue if all the requested output will fit into the buffer. If this is not so the director divides the output into parts which fit into the buffer and these are transferred successively. Meanwhile the process is suspended until the last part has been transferred into the buffer when it is then allowed to proceed. Gaps in the transfer to the console which might occur through having to wait for the next part to be transferred into the buffer by the director are for the most part avoided by double-buffering within the cyclic use of the buffer. Gaps can only occur if the system response time is longer than the time needed to transfer half the buffer to the console.

From the point of view of the supervisor and the user sitting at the console, there are three levels of priority in the use of the console. Lowest priority is regarded as input. When there is no other activity on the console, a read transfer is left on. This allows the user to type his input, possibly ahead of get input requests. If he types too far ahead, however, and fills the buffer, he is told to wait and the read transfer is removed, only allowing him to *interrupt* should he wish to. The next priority is output. As long as the user is not in the middle of a block typing ahead, any 'put output' request will cause the supervisor to halt the read transfer and to initiate the output transfer. If the user is typing ahead, the output is withheld until he completes an input block. Highest priority is the interrupt. When there is either a read or a write operation in progress, the user can interrupt it by typing an escape (ESC) character. ESC is used for the purpose since it is one of the characters which causes a hardware interrupt from the multiplexor when an input transfer is in progress. Due to the design of the multiplexer any character typed causes a similar interrupt when an output transfer is in progress. The break function, i.e. break the transmitted signal, which is often used for interrupting, is not used on EMAS because of the havoc it tends to cause in the multiplexor with the generation of an uncontrollably large number of hardware interrupts. This havoc has, of course, to be accepted when accidental line faults or breakages occur, though hardware modifications to improve the situation are being implemented. When the hardware interrupt generated by ESC is received, the supervisor terminates the operation in progress and queries the user for an *interrupt identifier*. This is a string of characters which after being read in is transferred to the director to take some action on. Three kinds of action are possible. The general mechanism is for the director to store the identifier together with the number of the console from which it came within its own data area so that the user level program can query, when it chooses, whether a particular identifier has appeared either at a particular console or at any console coupled to the process. This is a mechanism the user program can use to redirect itself, for example to terminate excessive printing, while still retaining control. The exceptions to the general rule are to cater for situations where some immediate action is required, e.g. the program is in a loop. Exceptional action is taken when the identifier consists of a single character. One of these, the letter *Z*, is allocated specially. If the director finds this, it immediately terminates the process, i.e. an escape route if all else fails. The remaining single character identifiers are treated as *outer level signals*, described below. The effect is to transfer control to some standard place in the user area. This is intended to be in the subsystem so that it can take some corrective action. In the standard subsystem, for example, the identifier *A*, for abandon, returns control to the basic command interpreter.

Parameter passing for multi-console operation

Particularly for console input/output with more than one console attached to a process, the possibility of certain service calls resulting in the process being suspended may be inconvenient. To circumvent this, an alternate way of issuing a service call is provided, using a call known as *pon*, for parameter on. In this case, the parameters for the required service are laid out in virtual memory instead of in the floating point registers where they are normally put and their address passed as the parameter to the pon. The effect is for the director, which receives the pon call, to issue the required service call recursively, and immediately to reply to the user level. The initial processing of the recursive call will, of course, take place before control returns to user level since the director has higher priority, but the effect of issuing the reply is to allow the user level to proceed whatever the outcome of the service call. The reply to that call will be sent to the director since all replies are sent to the source of the request. The sequence of code at the director entry point recognises that the input parameters represent a reply for a pon service and the information in the reply is stored on a queue.

A *poff*, for parameter off, call retrieves information stored by the director in this way. If there are no replies in the queue, the user level is suspended until one appears. Again, to avoid the possibility of suspension, a *toff*, for test for parameters off, call is provided which tests whether there are replies available to be poffed. Replies are returned to the user level by poffs in the order in which they come to the director. In other words, if the user level has ponned several calls before issuing a poff, the order of the replies may be different. The user level therefore identifies each pon call with an *activity* number as one of the parameters and this number is returned in the corresponding poff so that the reply can be identified.

This mechanism is also used to pass information from within the system or from a console to the user level. Certain activity members are reserved for this purpose. For example, one use is the passing to the user level of the console number of a new console just logged on so that it can be coupled and used for input/output.

The signal facility

The *signal* facility provides a method of recovering from failures in the user level. The user program can specify a position within its area and environment in terms of register contents required so that the program could be restarted at this point. Such a specification is set up by a call on the director which stores it so as to be available when required. If a number of these calls are made, the positions are stacked. For most common failures where a recovery is required, such as overflow, address error, etc. the most recently stacked definition is used. However, for disastrous errors, such as a hardware malfunction affecting this process, the oldest stacked definition, or *outer level*, is used. The standard subsystem always stacks a recovery position first to be used as this back-stop and other programs may stack and unstack definitions as they require. The user program may also

induce an artificial recovery at either the current or outer level of the stacked definitions.

Acknowledgements

Too many people have made contributions to the design of the director for them all to be individually acknowledged. Particular thanks are, however, due to S. T. Hayes, H. Whitfield and A. S. Wight. Thanks are also due to F. Barratt and his staff at the Edinburgh Regional Computing Centre for their valuable help in the elucidation of the multiplexor hardware. S. Michaelson has always provided encouragement and support. H. Whitfield implemented the signal mechanism.

References

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

MILLARD, G. E., REES, D. J., and WHITFIELD, H. (1975). The Standard EMAS Subsystem. *The Computer Journal*, to be published.

STEPHENS, P. D. (1974). The IMP Language and Compiler, *The Computer Journal*, Vol. 17, No. 3, pp. 216-223.

WHITFIELD, H., and WIGHT, A. S. (1973). EMAS—The Edinburgh Multi-Access System, *The Computer Journal*, Vol. 16, No. 4, pp. 331-346.

WIGHT, A. S. (1975). The EMAS archiving program, *The Computer Journal*, Vol. 18, No. 1, pp. 131-134.

Book reviews

Functional Analysis of Information Networks: A Structured Approach to the Data Communications Environment. by H. B. Becker, 1974; 281 pages. (*John Wiley*, $8.40)

In his preface, the author indicates that failures in implementing information networks (IN's) are usually due, not to faulty or inadequate design, nor to misunderstandings of the problem, but to one cause only—the lack of a DEFINITION of the problem.

In his book, Hal Becker has attempted, and achieved, this difficult definition task. Network components and functions are presented in a structured format, independent of reference to any proprietary devices. IN components are reduced to a basic set of simple functions, which are related by a hierarchical tree structure. Examples are then produced which evaluate designs from the top of this tree down to the applications level, rather than the traditional 'bottom-up' approach.

As an introduction to the subject, the evolution and definition of IN's are discussed, and their levels are then defined. Network functions in the form of the physical network are analysed to depth, and this is followed by a description of control functions in the form of the logical network. The main text concludes with an excellent IN example based on the ARPA network.

The appendix is worth mentioning, for it includes a full IN tree, a good but brief section on related reading, and a useful summary of the standardisation work taking place in this field.

Finally, it is appropriate to mention that the book is one of a series of 20-odd volumes intended for systems analysts, programmers, and DP managers.

I. G. DEWIS (Teddington)

Macro Processors and Techniques for Portable Software, by P. J. Brown, 1974; 244 pages. (*John Wiley*, £5·25)

This is one of the most useful books which I have seen for a long time. It is almost two books in one, as the title suggests.

The first 'book within a book' deals with the whole field of macro-processors. Starting from a simple example of writing standard letters to visiting speakers by means of macros, the basic concepts are introduced, and the facilities which may be needed are described. Distinctions such as general purpose versus special purpose, free mode versus warning mode etc. are illustrated with actual macro-processors such as the IBM OS Macro-assembler, PL/I pre-processing facilities, GPM, TRAC, ML/I and STAGE2. These are not examined thoroughly, but provide examples of various features. Several more chapters fill out the ramifications of these features, so that the subject is examined from all angles, always in a lucid readable style, with well-chosen illustrations.

The second 'book within a book' describes the technique of using macro-processors to implement a programming language in a portable fashion. An attempt is made to quantify (roughly) the expenditure of effort needed to write software and to transfer it to another machine by various methods. The method which is recommended is to code the software by means of a DLIMP (Descriptive Language for Implementing Macro-Processors). The DLIMP is low enough in level to be mapped on to most assemblers by means of suitable macros. As an example, almost a hundred pages are devoted to a detailed explanation (and full listings of the programs) of the implementation of ALGEBRA by means of the DLIMP called LOWL.

Throughout, the temptation to mount the hobby horse of ML/I (the author's own macro-processor) is staunchly resisted. The result is a wide-ranging description of the state of the art which never loses sight of practical objectives. Despite the price and a fair sprinkling of misprints, it is highly recommended reading.

A. C. DAY (London)

Graph Theory with Applications to Engineering and Computer Science, by Narsingh Deo, 1974; 478 pages. (*Prentice-Hall*, $17·05)

In his preface the author states that the book grew out of a number of courses. He attempts to recommend the sections relevant to particular subject areas. Even so, it has grown farther than the likely needs of most degree courses. It is rather a reference work on Graph Theory to which the author has confined himself most expertly and comprehensively. For all that, the text is greatly enhanced by constant reference to and example of practical application. The book maintains rigour and gives careful proof of most assertions yet particular attention is paid to the danger of losing contact with the reader. Contact is maintained when the going is heavy by taking time off to summarise progress in orderly and numbered notes and, most refreshingly, by simple numerical illustration. Copious exercises with useful hints are provided and a complete and fairly up-to-date bibliography will be of great benefit to anyone embarking on research in the subject.

The first ten chapters, almost exactly half the book, cover the theory of graphs, though, even here, one meets a mass of illustrative applications from chemistry to feeding dogs. Chapter eleven presents a number of basic algorithms, all flow charted and some programmed in a mixture of APL and FORTRAN. There is a useful list of graph oriented languages and software packages. The remaining four longish chapters are devoted to applications including switching and coding, electrical networks, operations research and a survey of other applications. Two theorems have, rather curiously, been relegated to two short appendices.

The book itself is nicely produced and a pleasure to handle. It should be on the bookshelves of every combinatorial mathematician and referred to by many more.

R. J. ORD-SMITH (Bradford)