University of Edinburgh

Department of Computer Science

Performance Measurement on the Edinburgh Multi-Access System

J.C. Adams & G.E. Millard

EMAS Report 7

James Clerk Maxwell Building
The King's Buildings
Edinburgh EH9 3JZ
031-667-1081

February 1975

Performance Measurement On The Edinburgh Multi Access System

J.C. ADAMS Department of Computer Science, Edinburgh University

G.E. MILLARD Edinburgh Regional Computing Centre

Summary

The performance monitoring facilities incorporated in EMAS (Edinburgh Multi Access System) are described and a brief description of the results obtained is presented.

- presented at the International Computing Symposium 1975

June 2 - 5 1975

Antibes, France.

PERFORMANCE MEASUREMENT ON THE EDINBURGH MULTI ACCESS SYSTEM

Introduction

The Edinburgh Multi Access System (W1) is a virtual memory, time sharing operating system developed at Edinburgh over the last eight years. It has been implemented on the International Computers Ltd. System 4/75 which is a paged 3rd generation machine. The entire system is written in the high level, Algol-like language IMP (S1). Up to fifty five or thirty five simultaneous users are supported depending upon hardware configuration and the system supports sharing at all levels. Originally intended as a research and teaching project as well as a production exercise, the system has had incorporated at various levels in its hierarchy a variety of monitoring facilities enabling close measurement of its performance. The structure of the system is shown in figure 1. The monitoring which will be discussed in this paper splits naturally into two main parts:

A) Measurements taken at the level of the resident supervisor. (All monitoring is done by software, we have no hardware monitoring facilities.)

B) Measurements taken at the subsystem level.

The system is implemented at present on two separate System 4/75's at Edinburgh. Normal hardware configurations used are given in table 1. Because of the architecture of the System 4/75 there is no multiprocessing and these systems are distinct, but the capability exists to switch filestores, drums, some tape drives and MCCCU's between machines for experiments or in the event of processor failure.

PERFORMANCE MONITORING WITHIN THE RESIDENT SUPERVISOR

The resident supervisor, in common with the rest of the system, is written entirely in IMP and is thus relatively easy to modify. Special, 'one off' pieces

		LEVEL	INVOKED BY	FUNCTIONS PERFORMED	CODE
RESIDENT SUPERVISOR	(A)	KERNEL	Interrupts, routine calls, returns from MAIN processes.	Inter-process message switching. Privileged hardware instructions. Interrupt message conversion. Process dispatching.	10 k.bytes
		DEVICE HANDLERS	Inter-process message.	I/O device control. Paged I/O scheduling and error recovery. Interactive communications device control.	52 k.bytes
		VIRTUAL PROCESSOR SUPPORT	Inter-process message.	Active and Main storage allocation. Virtual processor scheduling and contingency handling.	32 k.bytes
DIRECTOR		VIRTUAL MEMORY MAPPING	Inter-process message, and Master Page entries.	Association of virtual memory addresses and immediate memory sites. User process Signal mechanism.	4 k.bytes
		SUB-PROCESSES	Inter-process message.	Interactive communication. File system maintainence. File transfer for I/O.	46 k.bytes
USER EXEC		EXECUTIVE USER PROCESSES	Inter-process message.	Unit record device spooling. Batch scheduler. Archive memory control. Engineering test programmes.	69 k.+ 16 k. + 75 k.bytes
USER NORMAL	(B)	STANDARD SUBSYSTEM	Inter-process message and routine calls.	File definition. Object file layout, linking and loading. Logical I/O mapping.	110 k.bytes
		PROGRAMMES	Routine calls.	Editors, Compilers, Utilities and User provided routines and programmes.	450 k.+ user code bytes

VIRTUAL / REAL BOUNDARY

SOFTWARE PRIVILEGED / NON-PRIVILEGED BOUNDARY

figure 1

Table 1

E.M.A.S. HARDWARE CONFIGURATIONS

	MACHINE 'A'	MACHINE 'B'
4/75 CPU, operator typewriter and console	1	1
1μ sec CORE (4 bytes access, 2 way interleaved)	1024 K bytes	768 K bytes
2 M byte DRUMS (128 tracks, 4 pages per track) (transfer rate 860 K bytes/sec revolution 20 m sec)	3†	2†
2 x 350 M byte, non replaceable DISC drives on 1 channel (transfer rate 256 K bytes/sec revolution 40 m sec Average arm movement 60 m sec)	1†	1†
7.5 M byte replaceable DISC drives	2	3
1600 b.p.i. TAPE drives	2†	2†
Hardwired communications multiplexer (MCCCU)	1†	1†
DATEL 100 lines	64	16
DATEL 200 lines	16	8
DATEL 600 lines	4	0
DATEL 2400 lines	5	0
DATEL 4800 lines	0	4
British Standard Interface	1	1
line printer	2	1
card reader	1†	0
card punch	1	0
paper tape reader	1	0
paper tape punch	1	0

† - switchable between machines by means of a DXE switching unit.

The MCCU's will be replaced shortly by a Front End Processor (PDP 11/45) which is connected to the BSI's and currently handles 16 DATEL 100 lines, a synchronous line to another concentrator and a 1200 baud line to a video, usually going into the 'B' machine.

Page size is 4096 bytes (8 bits). On each machine the drums are on one channel.

of monitoring may easily be added for short periods of time, the compilation and linking of a new supervisor taking only about 10 minutes (real time) on average. We shall describe the permanent monitoring features in this section.

The scheduling algorithms employed within the resident supervisor are discussed in detail in the paper by Shelness et al. (S2) and we give a very brief resume here, to ease understanding of the results presented. Figure 2 is a diagrammatic representation of the management of a process by the supervisor showing all the system queues and major states involved. Each process known to the system exists in one of three states.

a) ASLEEP - awaiting user input

b) AWAKE - awaiting allocation of some system resource

c) PROCESSING - on CPU

Each process is also resident at up to a certain level in the storage hierarchy DRUM, DISC or CORE. E.M.A.S. uses a local scheduing algorithm in which each process is assigned to a CATEGORY dependent upon the recent past history of that process. Associated with each category are

i) a CPU, CORE and DRUM allowance

ii) a CORE queue

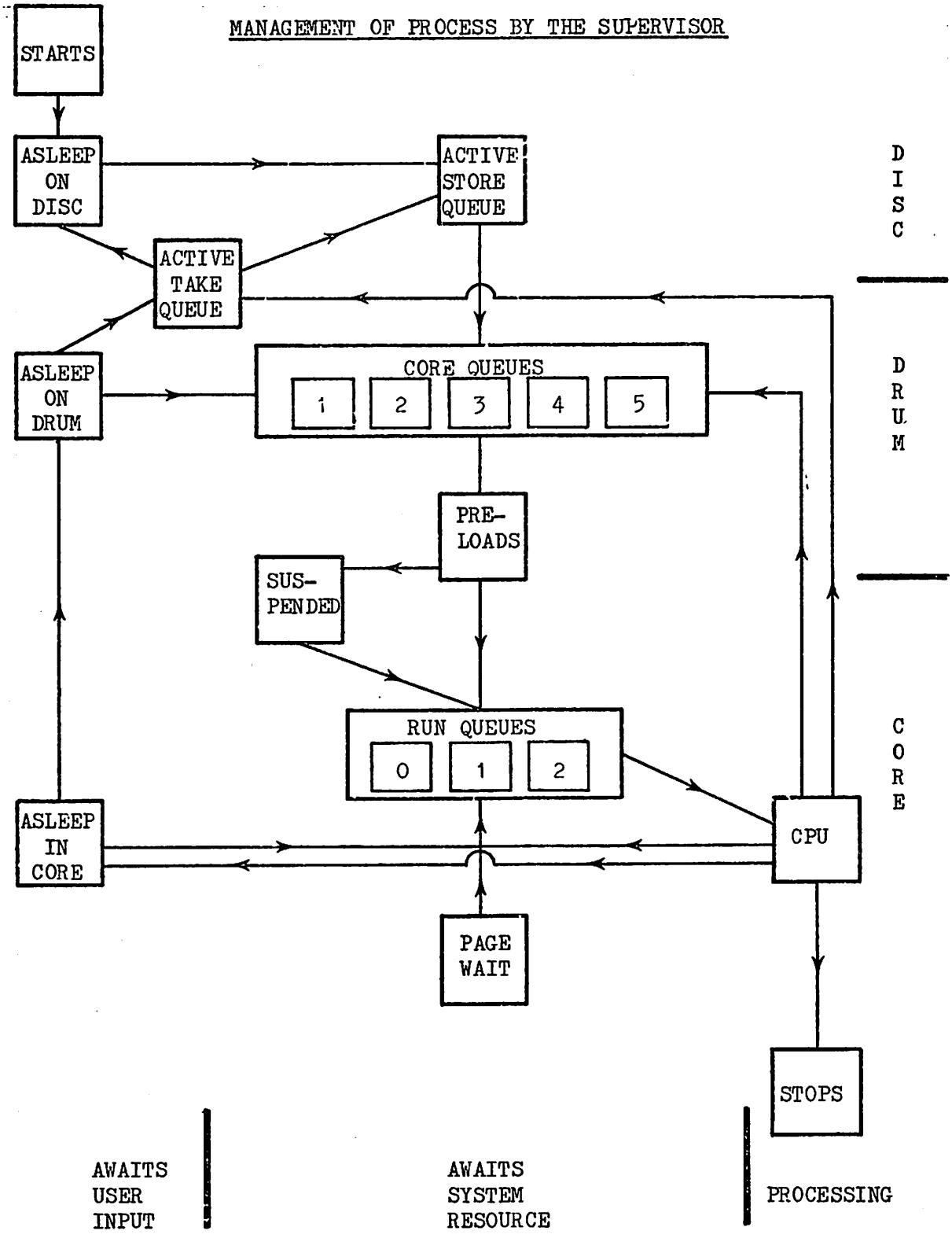
iii) a time period after which an attempt to compact the working set will be made

iv) a set of transitions to other categories dependent upon the future behaviour of the process

There are currently 20 categories. Table 4 shows the current category allowances.

Thus for a process which wakes up resident only on DISC. It is first placed in the Active Storage Queue to await a drum allocation. Then it is placed on a Core Queue dependent upon its current category. These core queues are serviced

MANAGEMENT OF PROCESS BY THE SUPERVISOR



according to a priority scheme and when a process is selected from a core queue it is held until its current working set (D1) is preloaded. When the process is core resident it is placed on Run Q0. The run queues are serviced according to an absolute priority basis. All processes which have just completed a page transfer being placed on Run Q0, all preempted or time-slicing processes on RUN Q1 and all penalised processes (which have not interacted with a console for a set period of time) are placed on RUN Q2. The time slice is currently 100 milliseconds. The process will remain core resident until it either goes to sleep or overruns one of its category allocations, whereupon it is rescheduled (perhaps into a new category) and then removed from core.

As it is highly unlikely that all of the core resident processes will be using their full core allocation at any given instant, core is over allocated by a certain amount to obtain more efficient utilisation. Another modification to the basic algorithm concerns preloading: when attempting to preload a process, if it is found that there is not adequate core free to give the process its full allowance but, enough free core exists to allow the preloading of the working set then this 'partial preload' proceeds. When the partial preload is completed, if enough core is still not available then the process is suspended until core is released. A drum working set is also maintained.

CPU TIME MONITORING

The resident supervisor itself consists of a set of routines (or services) each of which has its own unique service number. Requests for these services are stacked in a system table known as the MAIN Q. One of the functions of the KERNEL is to unstack these requests and call the appropriate service. We keep a simple count of the number of calls on each service and the time spent in each. This gives us a very accurate record of all time spent in the supervisor state and a complete record of the areas within the supervisor in which this time is being spent. This raw data is printed at the end of each session.

When the system is IDLE (i.e. none of its current multiprogramming set is able to run), a dummy process, (process $\neq 0$) is loaded and the system executes an idle loop until work arrives. This idle time is further broken down into time during which there was no work available (no process awake) and blocked time (all of the multiprogramming set awaiting some form of a page transfer).

A full analysis of CPU utilisation is printed at the end of each session (table 2). In a typically heavy user session the breakdown between User state, Supervisor state and Idle seems to be 50%, 30%, 20%. Figure 3 is a pie chart representation of supervisor time and it can easily be seen that the dominant areas are the organisation of drum transfers (41%) followed by core loading, (22%) the core loading figure includes time spent compacting working sets - a costly business on the System 4 as we have to read 4 read/write markers per page.

A facility which has proved useful in system tuning and the gaining of a rough insight into how system variables are behaving is one we call Q SAMPLING. In this, a routine is called every 10 seconds to sample interesting system variables. It accumulates a total and records the maximum and minimum values found. An example of the output from this is given in table 3. The negative figure for the minimum CORE UNALLOCATED value is a result of the deliberate over allocation of core. A 20 x 20 array is also stored in which we accumulate all transitions from any one category to any other. This transition matrix, which is printed at the end of each session gives us some insight into how well the system is tuned and what type of load is currently being placed on it. The matrix usually turns out to be highly diagonal and shows a great tendency to cluster around a small subset of the 20 categories. The current category allowances are shown in table 4. The % transition figure is the % of total category transitions involving this category.

EVENT TRACING

E.M.A.S. also maintains an event tracing facility. A list of events which

TABLE 2

E.M.A.S. CPU TIME ANALYSIS

Configuration 'A' from 17.27.01 on 21.02.74
to 01.40.30 on 22.02.74

	SECONDS	% OF TOTAL TIME
TIME IN USER PROCESSES	15272	50.90
SUPERVISOR TIME	9024	30.07
IDLE TIME	5705	19.01
TOTAL TIME	30001	100.00
ANALYSIS OF SUPERVISOR		
VIRTUAL MEMORY SUPPORT		
DRUM TRANSFERS	3675	12.24
DISC TRANSFERS	568	1.89
CORE LOADING	2942	9.80
DRUM LOADING	247	0.82
PROCESS CONTROL	132	0.43
TIME SLICING	661	2.20
FILE SYSTEM	41	0.13
SVC PARAMETER PASSING	354	1.17
COMMUNICATIONS	177	0.58
DEVICE POLLING	35	0.11
MAGTAPES	8	0.02
MISC.	184	0.61

Supervisor CPU Time Analysis

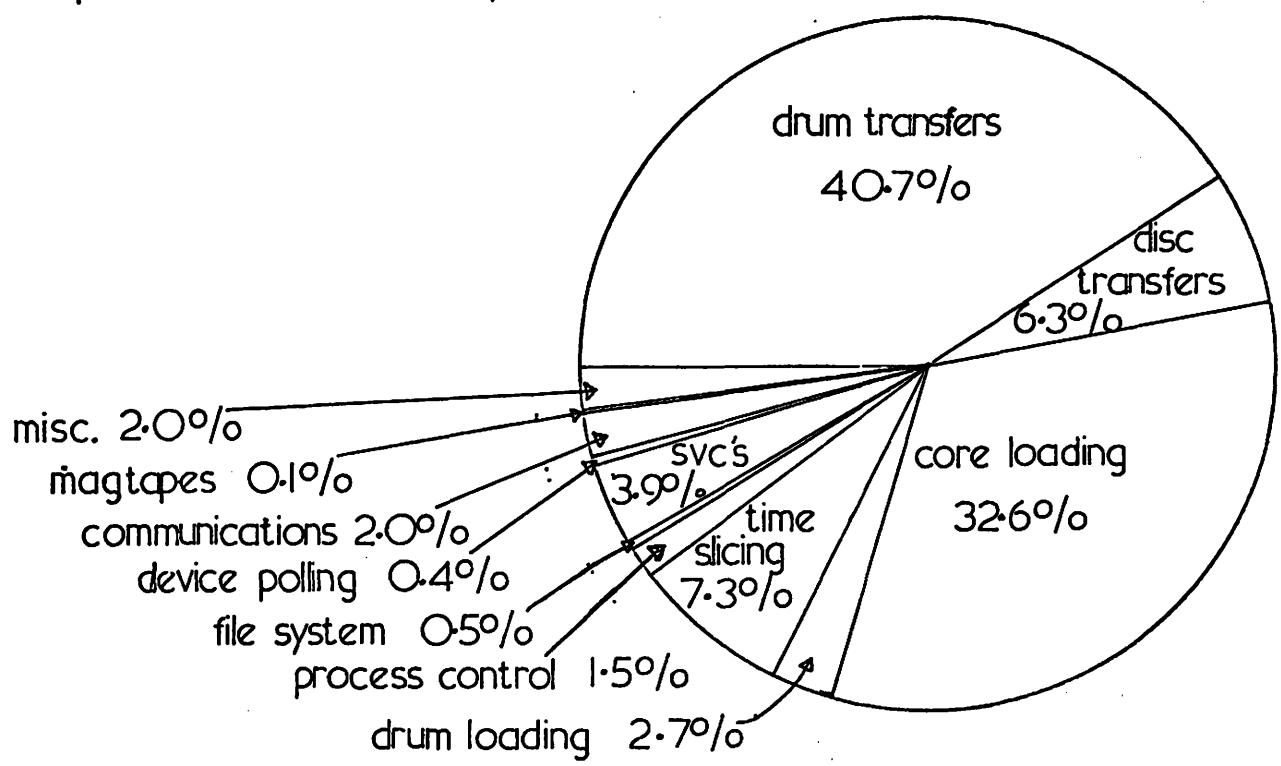


Figure 3

QUEUE SAMPLING INFORMATION — TABLE 3

CONFIGURATION 'A' 12/03/74 10.07.51 - 12.54.50

NO. OF TIMES QSAMPLE KICKED WAS 1000

ITEM	TOTAL	MAX	MIN	
RUNQ1	416	7	0	
RUNQ2	438	4	0	AWAITING CPU
RUNQ3	738	4	0	
ACT TKEQ	1313	20	0	AWAITING REMOVAL FROM DRUM
CORE Q1	1213	19	0	
CORE Q2	1557	13	0	
CORE Q3	815	8	0	AWAITING ALLOCATION OF CORE
CORE Q4	117	3	0	
CORE Q5	1870	9	0	
CORE L	32591	202	-35	UNALLOCATED CORE PAGES
CORE F	57129	185	6	UNUSED CORE PAGES
CORE S	38270	107	0	SHARED CORE PAGES
ASUNUSED	730547	1182	288	UNALLOCATED DRUM PAGES
AS FREE	741322	1165	342	UNUSED DRUM PAGES
BPTUNUSD	153822	285	23	UNALLOCATED BLOCK PAGE TABLES
BPTFREE	158322	261	40	UNUSED BLOCK PAGE TABLES
PT FREE	52920	97	6	UNUSED PAGE TABLES
SAM FREE	42365	71	12	UNUSED SHARED ACTIVE MEMORY TABLES
PARAMTAB	145685	158	101	UNUSED ENTRIES IN PARAM.PASS.TABLE
USERS	37939	50	27	

TABLE 4 — E.M.A.S. Category Allocations

CAT.	COREQ	CORE (pages)	MAX DRUM (pages)	MIN DRUM (pages)	CPU (sec)	W.S.COMPACT. (sec)	%TRANS.
1	1	50	80	64	1.0	0.128	0.8
2	1	20	80	64	0.5	0.5	8.1
3	1	30	80	64	1.0	1.0	7.4
4	1	50	80	64	2.0	0.5	1.7
5	1	20	80	64	0.5	0.5	23.3
6	4	20	80	64	4.0	1.0	0.1
7	5	20	80	64	10.0	1.0	0.2
8	1	30	80	64	1.0	0.5	15.0
9	4	30	80	64	10.0	1.0	0.2
10	4	30	80	64	6.0	1.0	0.3
11	2	40	80	64	1.0	1.0	16.5
12	4	40	80	64	10.0	1.0	0.3
13	5	40	80	64	12.0	1.0	0.5
14	2	50	80	64	1.0	1.0	12.8
15	4	50	80	64	10.0	1.0	1.0
16	5	50	80	64	10.0	1.0	0.9
17	3	60	128	80	2.0	0.5	5.1
18	4	60	128	80	7.0	0.5	0.0
19	5	60	128	80	5.0	1.0	0.8
20	3	62	128	80	2.0	0.25	5.2

may be monitored is given in table 5. The numbers circled in figure 2 show the points in a process existence at which these events are issued. The monitoring is switched on from the operator's console by setting a system test flag to a mask value showing which events are to be traced. The event monitor then claims 4 pages of buffer space and starts up, depositing its data on one of the replaceable disc units. The monitoring automatically switches itself off, if the test flag is reset, if the system closes down or if it fills its data space (800 pages). The data is then transferred by the privileged ENGINEERS process from the RDU into an EMAS file for analysis.

This type of monitoring produces vast quantities of data e.g. on the 'B' machine configuration, with all events being traced and approximately 30 users on the system, the data area (3.2 M bytes) was filled in only 17½ minutes (230,000 records) representing a CPU degradation during this monitoring period of approximately 1% over the normal system overhead. However this is a very good way to get accurate figures on system behaviour at a very deep level. Page traces obtained by this method are used in the fine tuning of system utilities to current scheduling schema

Several analysis programs have been developed. These give results on a by process, by category, or by number of active users on the system, basis. Amongst the data produced by these programs are

total paging traffic of each type

mean transfer times

paging rates and mean transfer times for each rate

total traffic between the stations on figure 2

distributions and means for waiting times and queue lengths for each of the stations shown in this figure

successes of attempts at compaction of working sets.

The following results presented were obtained from a trace monitoring session on configuration 'B' during a session with 30 users during a typical Friday

TABLE 5

E.M.A.S. LIST OF EVENTS WHICH MAY BE TRACED

The first two words of an event record are always in the same format:

word 1, splits into 4 byte fields:
event identifier! length (in words)! current process! CPU process.
word 2, is always the value of the clock register in 4/75 clock ticks.

ID	EVENT	WORDS PER RECORD
1	Process wakes up.	3
2	Process is put onto a system queue.	3
3	Process begins to preload.	4
4	Process completes preloading.	4
5	Process makes a demand page request from disc.	4
6	Process makes a demand page request from drum.	4
7	A demanded page arrives in core.	2
8	Process makes a demand page request for a shared page or a page in core.	3
9	Process overruns a category allocation.	3
10	Attempt made to compact process working set.	3
11	Process goes to sleep (awaits console input).	4
12	All of process' pages removed from core.	4
13	All of process' pages removed from drum.	4
14	Process goes to sleep whilst holding a semaphore.	2
15	Process drum working set is recalculated.	4
16	One of process' pages removed during process' removal from core.	4
17	One of process' pages removed during compaction of working set.	4
18	Process is created.	2
19	Process begins to log out.	2
20	Process logged out.	2
21	Process suspended after partial preload.	4
22	Process resumed after suspension.	2
23	Process removed to disc to ensure consistency of files.	2
24	Attempt made to compact working set when process moved out, before reaching a normal compaction point.	2
25	Address of preloaded page just arrived in core.	4
26	Process has just issued a Supervisor Call.	4
27	Process working set is transferred to a different drum.	4
28	Exit from supervisor state	4
32	Status of system queues (every 10 seconds).	5
33	Start up or gap.	5

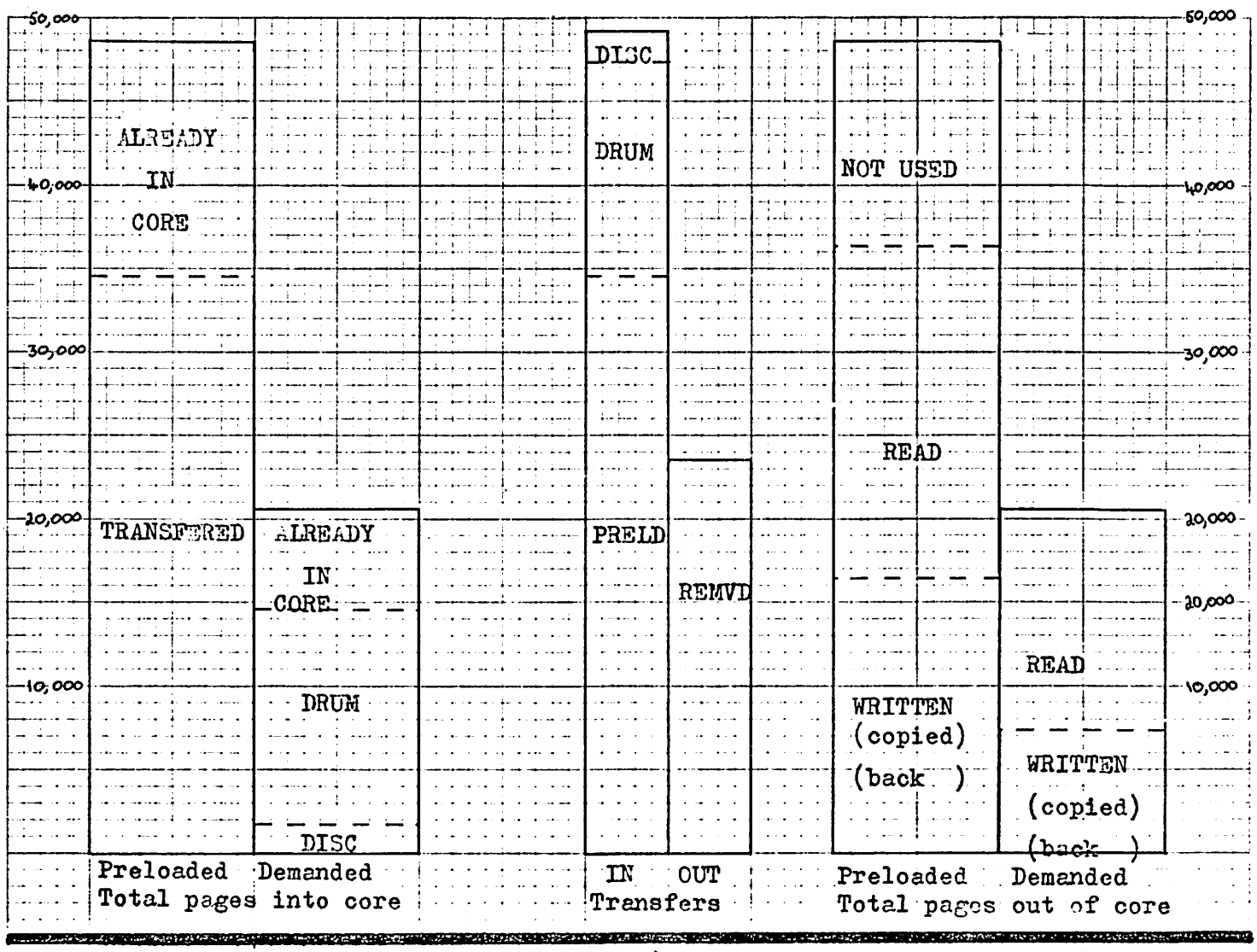
Events 32 and 33 may not be masked out.

afternoon session.

PAGING TRAFFIC THROUGH CORE

As the data from this session shows, approximately twice as many of the paging requests were for prepaged pages rather than demand pages (70%:30%). Only 71% of the prepaging requests resulted in transfers (virtually all from drum), the rest being for shared pages or for pages already in core. The mean number of pages per preload was 20 (14 requiring transfers). The demand paging requests show 8% for pages on disc 30% for SHARED or already in core pages and the rest for pages on the drum. The average page transfer times for the various types are: prepaged 21 milliseconds, demand paged from disc 443 milliseconds, demand paged from drum 63 milliseconds. Inherent in prepaging is a certain waste as it is highly unlikely that all of the pages which are prepaged will be used, on the basis of current figures approximately 25% of prepaged requests are for pages which are not subsequently accessed in any way. However from the data produced by this run, if only demand paging (from drum), for all the 'useful' pages, were to be used it would increase the total page wait time for these pages by a factor of 2, and the wastage figure would have to rise above 37% before demand paging would show any advantage. It would be hoped that the prepaging wastage could be cut if it was possible to make more frequent recalculations of the working set, but this would be too expensive with the current architecture (figure 4).

Removal of pages from core takes place when either a process is removed from core or some pages are found to have been unused for some time during a compaction of the process working set. Pages removed during compactions form by far the smallest factor (7%). 85% of attempts to compact the working set actually succeed in removing pages. Of all pages brought into core approximately 34% are subsequently written to and require writing back out to drum, the average time per page for such a transfer being 34 milliseconds.


50,000
40,000
30,000
20,000
10,000
ALREADY IN CORE
TRANSFERED
ALREADY IN CORE
DRUM
DISC
Preloaded
Demanded
Total pages into core
DISC
DRUM
PRELD
REMVD
IN
OUT
Transfers
NOT USED
READ
WRITTEN (copied) (back)
READ
WRITTEN (copied) (back)
Preloaded
Demanded
Total pages out of core

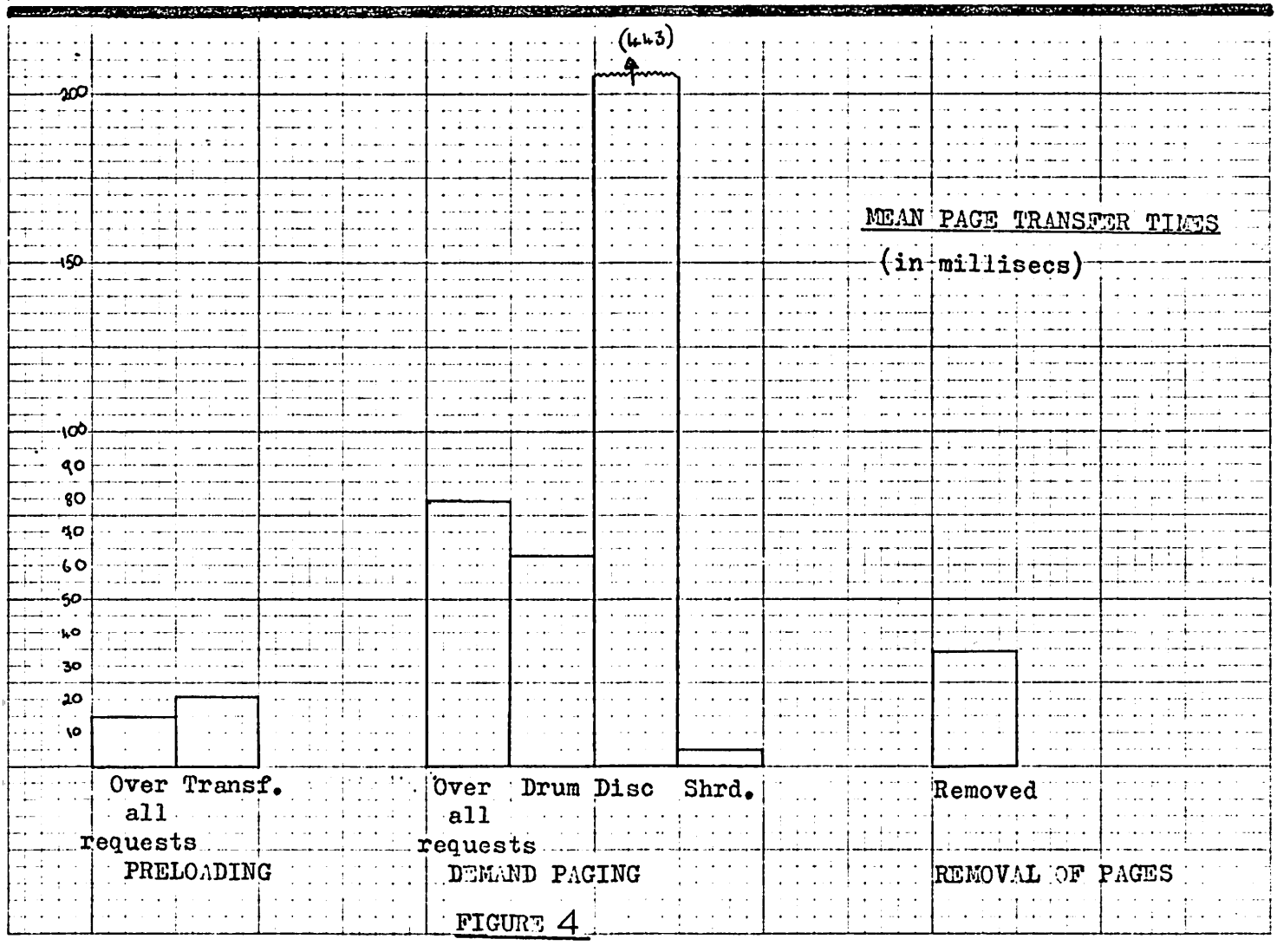

(443)
200
150
100
90
80
70
60
50
40
30
20
10
MEAN PAGE TRANSFER TIMES
(in millisecs)
Over all requests PRELOADING
Transf.
Over all requests DEMAND PAGING
Drum
Disc
Shrd.
Removed
REMOVAL OF PAGES

FIGURE 4

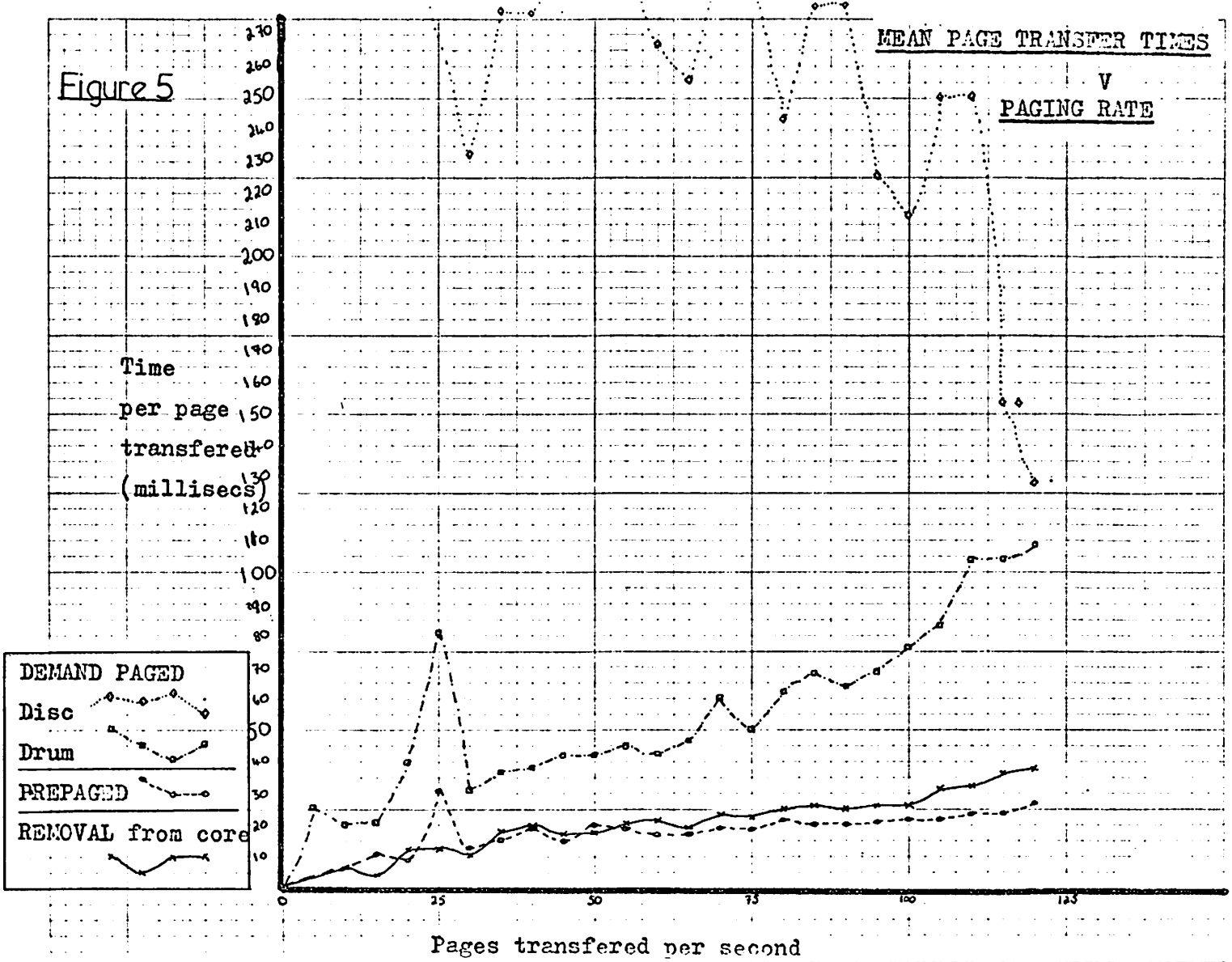
An analysis of paging rates is made by dividing the monitored period into intervals of 2 seconds, calculating the paging rate and the mean page transfer times during this interval. The contribution made to this paging rate by the various types of paging is also recorded. The results of this analysis are shown graphically in figure 5. It is interesting to note that as the paging rate increases this is mainly due to an increasing contribution from prepaging, with the demand paging factor showing a tendency to level off. This demonstrates how the local scheduling algorithm limits the demand paging rate (and hence eliminates thrashing). However the mean page transfer times show an opposite picture and as the rate increases the time for a demand page transfer increases whilst those for prepaging show a very small increase. Thus the differential between prepaging and demand paging increases with prepaging becoming increasingly more efficient as the rate increases.

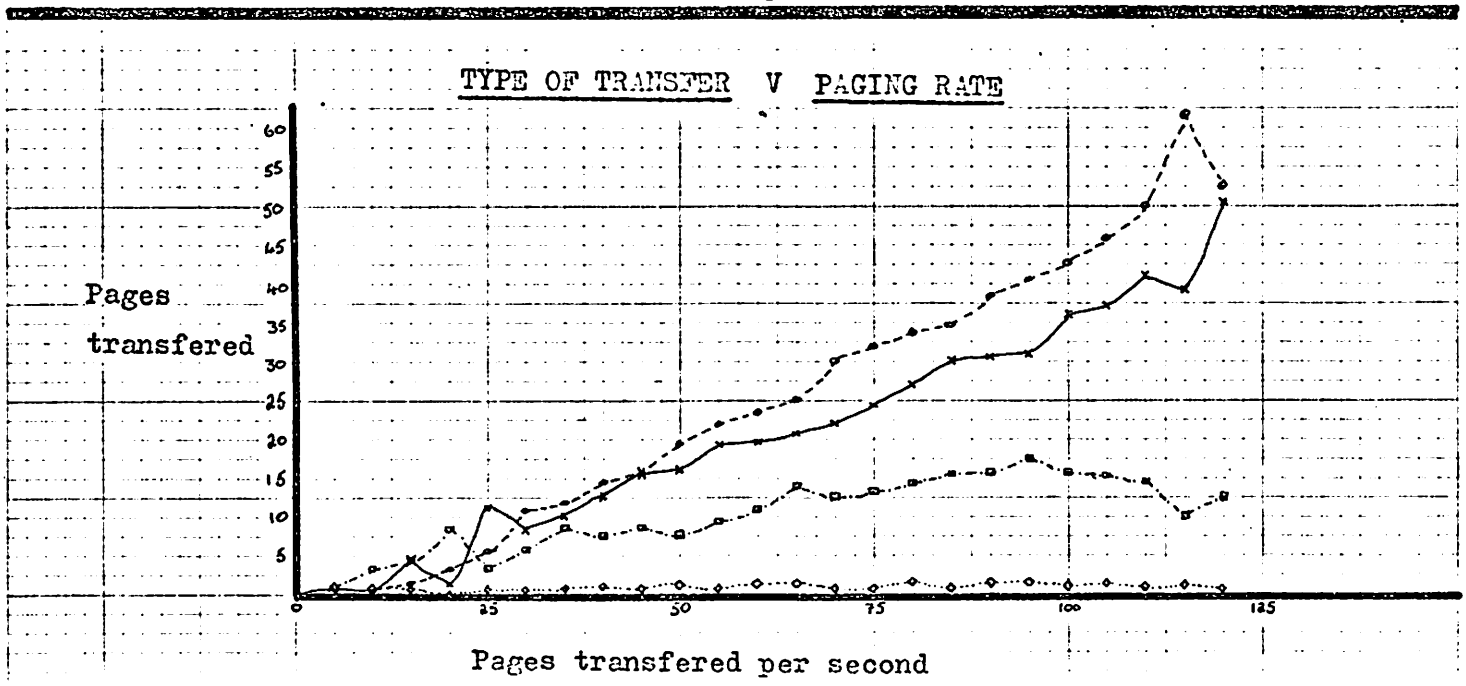
QUEUING TIMES

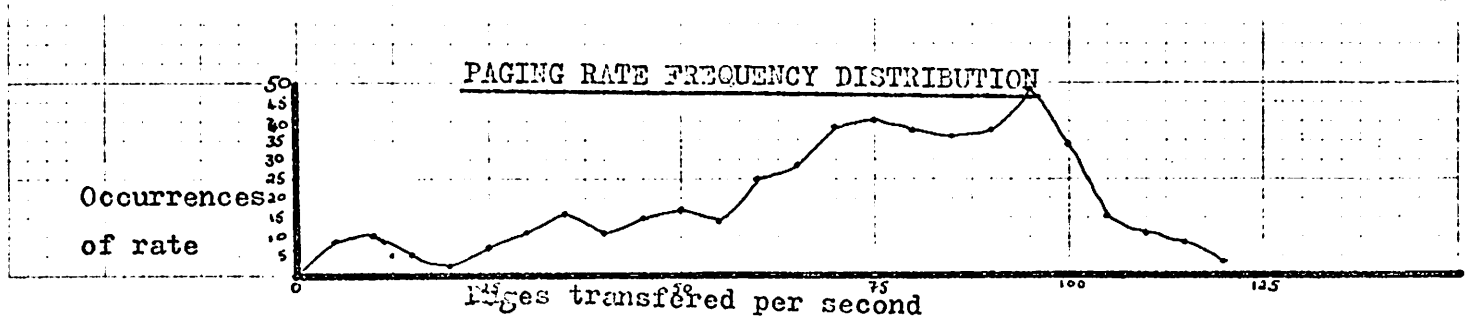
It would be impossible to enter any detailed discussion of queuing times here but figure 6 is presented to give an indication of queuing activity, showing total numbers of entries to each queue and mean wait times per entry, during the monitored period. The mean multiprogramming level during this period was 4 rising on occasions to 8. The mean headway between faults is in the order of 17 milliseconds and the mean CPU time per core residence is 157 milliseconds. A brief check of mean wait times per core residence reveals that 1435 msecs were spent on a core queue, 303 msecs preloading, 199 msecs on a run queue, 677 msecs waiting on a page fault and 157 msecs on the CPU which gives a ratio of 16.6:1 for total wait time v CPU time obtained, during a relatively heavily loaded period.

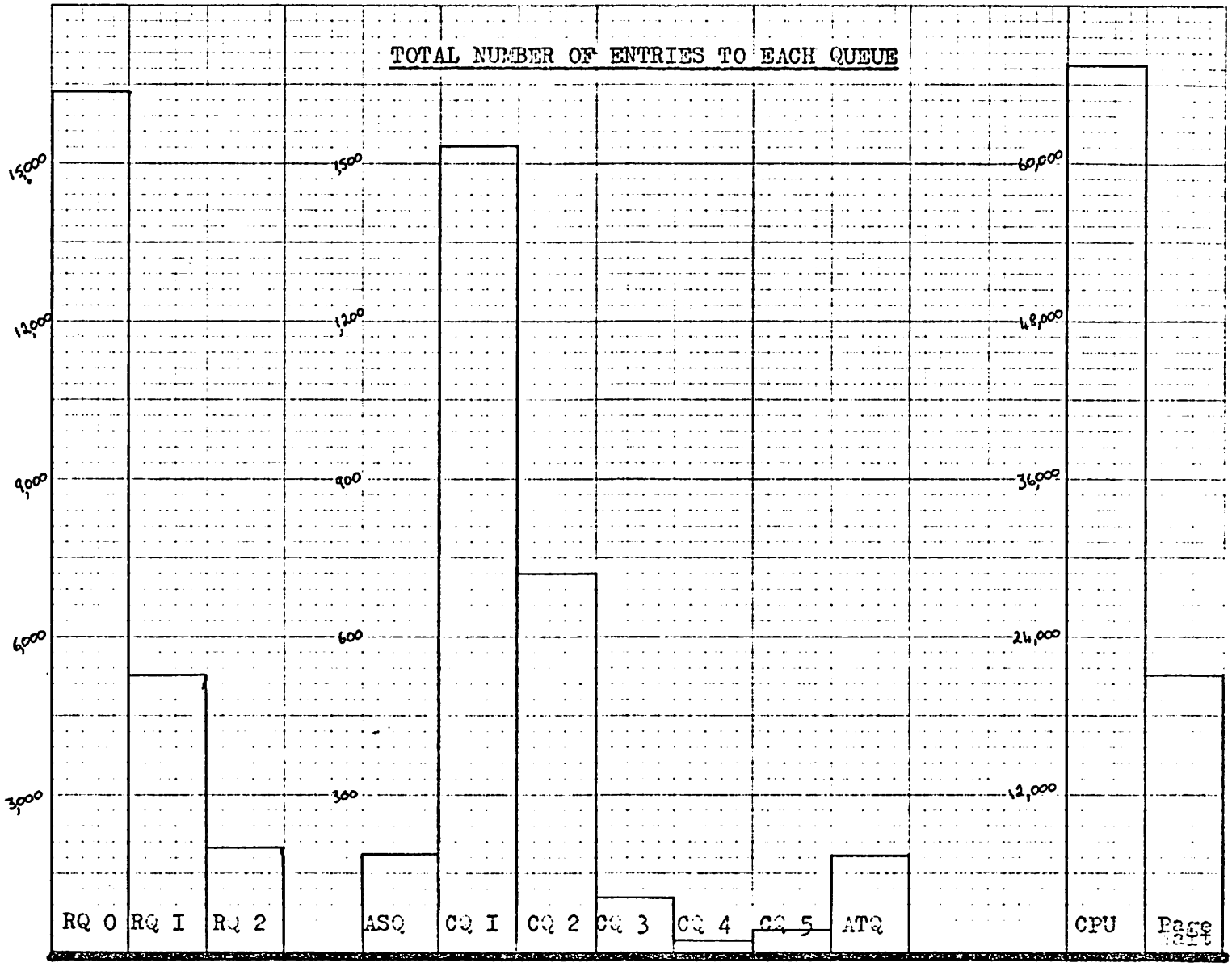
Though the majority of the figures presented are cited as overall averages it would be more meaningful to consider them broken down into individual categories. Indeed great variation is found in the behaviour of the different categories but such analysis must be the subject of another paper.

Figure 5


MEAN PAGE TRANSFER TIMES
V
PAGING RATE
Time per page transfered (millisecs)
DEMAND PAGED
Disc
Drum
PREPAGED
REMOVAL from core
Pages transfered per second


TYPE OF TRANSFER V PAGING RATE
Pages transfered
Pages transfered per second


PAGING RATE FREQUENCY DISTRIBUTION
Occurrences of rate
Pages transfered per second


TOTAL NUMBER OF ENTRIES TO EACH QUEUE
15,000
12,000
9,000
6,000
3,000
1,500
1,200
900
600
300
60,000
48,000
36,000
24,000
12,000
RQ 0
RQ I
RQ 2
ASQ
CQ I
CQ 2
CQ 3
CQ 4
CQ 5
ATQ
CPU
Page Wait

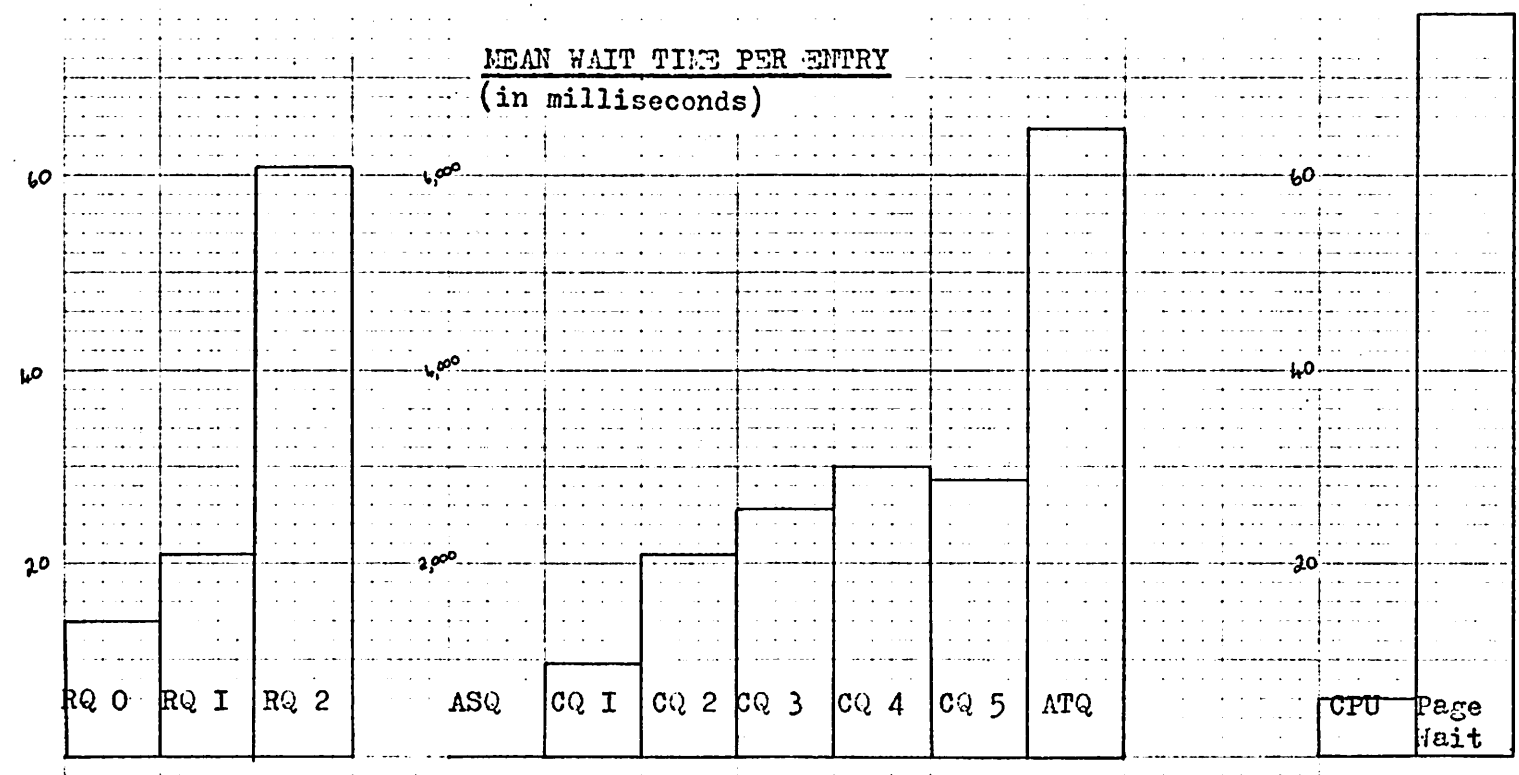

MEAN WAIT TIME PER ENTRY
(in milliseconds)
60
40
20
6,000
4,000
2,000
60
40
20
RQ 0
RQ I
RQ 2
ASQ
CQ I
CQ 2
CQ 3
CQ 4
CQ 5
ATQ
CPU
Page Wait

FIGURE 6

SUBSYSTEM MONITORING

The primary reason for monitoring activity at the subsystem level was our wish to determine which facilities were most extensively used and to obtain quantitative measures of command rate, CPU utilisation, response times and terminal input/output activity. It was hoped that this information would enable us to improve the performance of the subsystem by minimising the number of subsystem pages needed for the most frequently used commands. A secondary objective was to provide the information which would be needed to define and construct a 'benchmark' for comparing the performance of different multi-access operating systems, or indeed the same system on different hardware configurations.

DATA CAPTURE

Each user of the standard EMAS subsystem (M1) has mapped into his virtual memory the same physical copy of the 'basefile'. This is an object program file containing the basic command interpreter (BCI) and all the other routines needed to support the standard facilities. Each command typed by a user is processed by the BCI. It is assumed to be the name of a routine (system or user provided) which is located, loaded if necessary, and entered. Following its execution, which may involve further input/output at the user's terminal, control is returned to the BCI.

If subsystem monitoring is active a monitoring record is written each time control returns to the BCI. Each record includes the following information.

user identifier (a six character name assigned by the system manager)
time of day at which the command was received
command (eight significant characters)
ready time i.e. time of day at which the command has been completed
cpu time used
page turns used
number of lines input/output at the terminal

number of characters input/output at the terminal

file size (relevant for editing and compiling commands)

completion code (relevant for compilations)

Much of the information required is readily available in the subsystem data space at the time the record is written. A call on supervisor is required to note the time when the command is received, and again on completion at which time a metering call is also made to determine the current cpu time and page turn utilisation by the process. All input/output associated with the user's terminal is routed through one routine in the subsystem, enabling line and character counts to be maintained.

On EMAS, files contained in the file system are not accessed by 'conventional' transfers of blocks of information between a storage device and core under the control of a user program. When a particular file is required it is simply mapped into the virtual memory and accessed directly. Pages not in store when they are referenced generate a page fault which is handled by the supervisor. The current size of a file is held in a header record at the start of the first page of a file. Thus the sizes of files input to the editor and compilers are directly available when the files are first connected.

Additional records are generated when a process is started, i.e. after a user logs on, and when the user logs off. Otherwise redundant fields in these records contain the date and total resource utilisation during the session.

Thus the only modifications to the subsystem for monitoring user activity have been the addition of a small routine to the BCI and minor additions to the terminal I/O, editor and compilation control routines. Care was taken to ensure that these changes would not result in references to any pages which would not otherwise be in core store.

DATA COLLECTION

It was obviously necessary to attempt to minimise the overhead in accumulating the recorded data from up to fifty concurrent processes. As mentioned above files are mapped directly into processes' virtual memories and records are accessed by direct reference to virtual memory addresses. It was decided to collect all the data in one file belonging to the system manager's process. When a process is started it determines whether it is permitted write-shared access to the file. If so, it connects the file in write-shared mode, and monitoring is activated. The monitoring file is created with adequate length to contain the records for the period required and has a standard EMAS header containing pointers to the current end of data and physical end of file. Monitoring records are of fixed length and are mapped consecutively onto the file following the header. When a monitoring record is to be written from a particular process the area in the data file is first reserved by updating the current data pointer at the head of the file. If this does not exceed the physical end of the file then the record is then copied to the file. If an end of file condition applies the monitoring activity automatically switches off.

Monitoring is activated by initialising the collection file and permitting it to all users in write-shared mode and terminated by withdrawing the access permission (this does not affect currently active users but no new user will be monitored).

OBSERVATIONS

Subsystem monitoring has been carried out at intervals over two years and samples representing in some instances several days continuous monitoring have been analysed. Comparisons between the distribution of commands measured over a 24 hour period with that determined from an included, continuously busy, period (3 - 5 p.m.) have shown discrepancies of the order of only one per cent. It was

therefore thought adequate to look at analyses of mid-afternoon sessions to determine any trends in the user behaviour pattern or changes in system performance.

As anticipated the monitoring has enabled us to observe the benefits (or otherwise) of different scheduling strategies within the supervisor. We have also been assisted in our attempts to improve the internal organisation of the subsystems such that overall paging activity is reduced.

Over the last two years the only significant change to be observed, apart from improvements in performance, has been a reduction in the average size of source files being compiled. This must in part be due to the increasing number of students using the system, but may also reflect a greater awareness of users to the benefits of editing and recompiling only those parts of their programs which are currently under development.

Since the observations have been so consistent it is relevant to reproduce here some figures relating to a typical two hour period (3-5 p.m. on 22nd April 1974).

Number of concurrent users	25-30
Number of user sessions	136
Total number of user commands	1652
Cpu time spent in user processes	51 mins (42.5% of elapsed time)
Average cpu/command	1.85 secs
User terminal activity	
input average	3.5 lines (43 chars)/command
output average	8.5 lines (327 chars)/command

Table 6 gives the command distributions and the proportion of cpu and elapsed time occupied by the principal commands for editing, compiling and executing programs. The other commands are mainly used for interrogation (e.g. number of users on the system, resources used) and basic file operations (e.g.

listing files, destroying files, defining associations between logical and physical files).

Command type	Count	% total	% cpu time	% elapsed time
Editing source and data files	329	20	8	51
Compiling (IMP and FORTRAN)	153	9	24	8
Executing compiled programs	280	17	61	34
Others	890	54	7	7

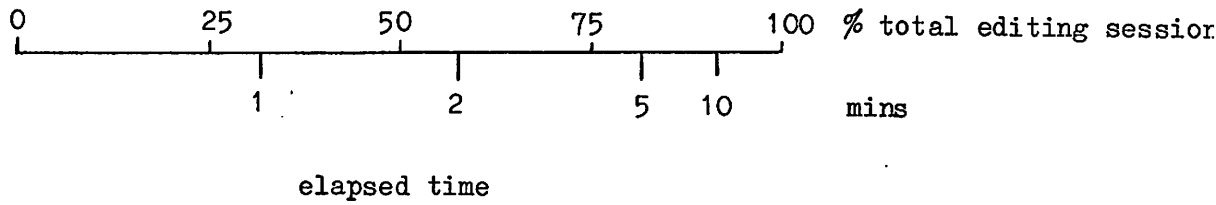
Table 6 Command distribution for a typical two hour session

Figure 7 shows, in percentage terms, the elapsed time distribution for the editing, compiling and program execution commands observed during the above session, together with an indication of the sizes of programs compiled and cpu demands made by programs in execution. The average elapsed time for editing was about 3.5 mins and on average each editing session involved 11 lines (140 chars) input and 12 lines (288 chars) output. The average cpu time requirement for a compilation was 5 secs (250-300 statements), and the average elapsed time 67 secs. As can be seen in figure 7(b) these figures are biased by a small number of large compilations. The elapsed times for programs being executed (figure 7(c)) would have been significantly affected by the amount of terminal I/O activity - on average 3 lines (31 chars) in and 27 lines (1233 chars) out at a maximum rate of 10 chars/sec. The average elapsed time of 158 secs and average cpu time of 6.6 secs are also biased by a number of longer running programs.

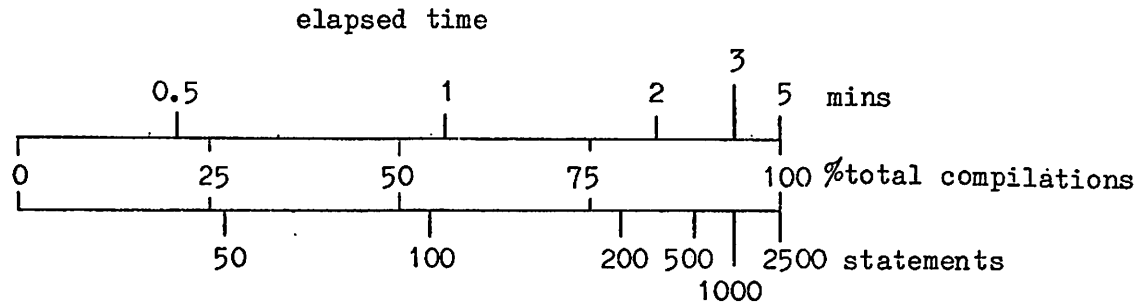
CONSTRUCTION OF A BENCHMARK

Having demonstrated that at a given point in time the profile of user activity is very stable a benchmark has been defined based on the observations over a specific two hour period. Additional information was required about the

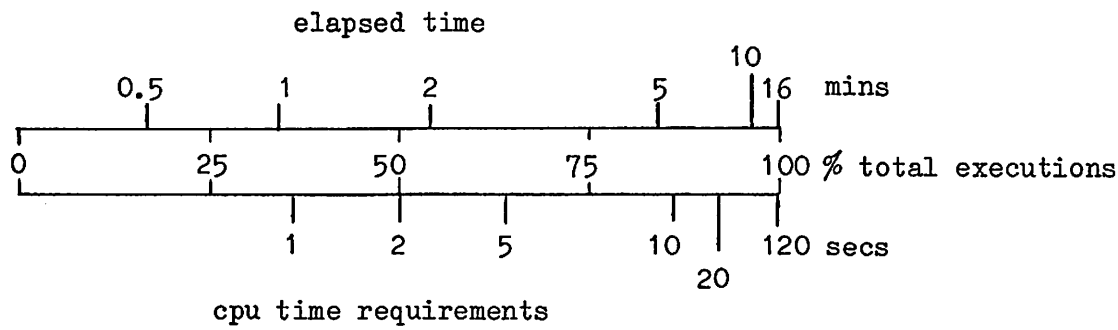
Characteristics observed during a typical 2 hour session



(a) Editing



(b) Compiling



(c) Executing compiled programs

Figure 7

behaviour of a typical user at his terminal, in particular his typing rate and the 'think time' between receipt of an output message or prompt and the input of the next command or data. This has been obtained by external monitoring of the activity over the lines connecting individual terminals to the ICL 4/75s.

A number of 'jobscripts', each containing the command and data input for a complete terminal session, have been constructed. The individual command totals were identical with those in the observed period and the program files, contributed by users, were selected to exhibit the spread of compilation and execution characteristics which had been observed. A PDP 11/45 has been used to simulate the user activity defined by the jobscripts. Each job script includes control data specifying think times and typing rates and appropriate delays are imposed by the PDP 11/45 control program. Delays appropriate to the maximum output rates of the terminals being simulated are also imposed. The PDP 11/45 has the appearance, as far as the system is concerned, of a standard EMAS concentrator.

This benchmark will be used by this and other Universities for assessing the capabilities of new systems. It also provides a standard workload for a series of experiments to be conducted on EMAS using different hardware and software configurations.

ACKNOWLEDGEMENTS

We would like to acknowledge the contribution made by everyone connected with the EMAS project, especially Peter Stephens for invaluable help in the resident supervisor area, and by Ken Dietz and Brian Gilmore in measuring external characteristics and providing the PDP 11/45 software respectively.

REFERENCES

D1 Denning, P.J. 'The working set model for program behaviour' Communications of the ACM Vol 11 No. 5

M1 Millard, G.E., Rees, D.J. and Whitfield, H. 'The Standard EMAS Subsystem' Computer Journal - to be published.

S1 Stephens, P.D. 'The IMP language and compiler' Computer Journal Vol 17 No. 3

S2 Shelness, N.A., Stephens, P.D. and Whitfield, H. 'The Edinburgh Multi-Access System Scheduling and Allocation Procedures in the Resident Supervisor' Proc. International Symposium on Operating Systems, Theory and Practice. April 23rd-25th 1974 I.R.I.A. Paris

W1 Whitfield, H. and Wight, A.S. EMAS - 'The Edinburgh Multi Access System' Computer Journal Vol 16 No. 4