MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

A Simulation Study of a CPU Priority Task Dispatcher

R. A. Pocock

IPP R/4 January 1972

Die nachstehende Arbeit wurde im Rahmen des Vertrages zwischen dem Max-Planck-Institut für Plasmaphysik und der Europäischen Atomgemeinschaft über die Zusammenarbeit auf dem Gebiete der Plasmaphysik durchgeführt.

IPP R/4 R.A. Pocock

A Simulation Study of a CPU Priority Task Dispatcher

January 1972
(in English)

ABSTRACT:

The central processor priority dispatching program SMOOTH, introduced by IBM to schedule allocation of the 360/91 central processor to separate data processing tasks in a multitask system, is simulated using the simulation system GPSS/ 360. A variety of task mixes is considered, with the computer running with and without SMOOTH. The advantages and limitations of the dispatcher are discussed.

1. Introduction

This study is designed to determine the effect of the priority dispatching program SMOOTH on the allocation of the 360/91 central processor (CPU) to separate data processing tasks in a multitask system. Since the allocation of input/output facilities (i.e. I/O channels) to particular tasks is dependent on their CPU scheduling, I/O channel allocation is also considered. The aim of an efficient dispatcher should be to overlap as much as possible the use of the CPU by one task, with the use of I/O channels by other tasks. The priority dispatcher SMOOTH attempts to do this by using a simple algorithm to reduce the dispatching priority (DP) of CPU-bound tasks (i.e. jobs which spend much more time using the CPU than the I/O channels).

The running of the computer for various job mixes has been simulated, both with and without SMOOTH, by use of the General Purpose Simulation System GPSS/360. Comparison of the CPU/I/O overlap for the two cases shows that in general SMOOTH increases the efficiency of task scheduling. In certain cases, however, the SMOOTH algorithm shows certain limitations and these are also considered.

2. The Operation of the Priority Dispatcher SMOOTH

The Priority Dispatcher SMOOTH is a resident system task of high (system job) dispatching priority (DP=236), running in a region of 6 K bytes, with effective core utilisation of 1500 bytes. SMOOTH, although always present in the system, is usually in the wait state, but takes control of the CPU every 500 msecs. to scan the Task Control Block (TCB) chain for CPU-bound programs. If a CPU-bound task is found, its dispatching priority may be reduced using the OS/360 CHAP macro.

Whilst SMOOTH is scanning the TCB chain I/O and external interrupts must be supressed in order to avoid alteration of the chain. The time of supression is of the order of a few milliseconds. During its scan SMOOTH ignores the TCB's of system jobs and problem subtasks as it does not attempt to alter the DP's of these tasks.

Every problem (job step) main task has a table in core from which SMOOTH can determine its stepname and "remaining" CPU time. This "remaining" CPU time is not the actual CPU time left to run, but the difference between the CPU time used so far and the CPU time requested by the programmer (or by default) for the jobstep. Thus, if after a scan interval the step is still active, the CPU time used during the interval can be calculated from the difference between the "remaining" CPU time at the beginning and end of the interval.

SMOOTH scans the TCB chain from highest to lowest DP, and ends its scan with the penultimate TCB or with the last TCB with dispatching priority DP $>$ 11. It should be noted that dispatching priorities are calculated from selection priorities (the job/step external priority allocated by the programmer or by default) according to the following algorithm:

$$DP = (\text{selection priority}) \times 16 + 11.$$

Thus the program with the lowest DP or with DP $<$ 12 is not subjected to change of priority by means of the CHAP macro, even though it may have monopolised the CPU during an interval.

A simplified flowchart of the SMOOTH scan loop is shown in Fig.1. SMOOTH scans down the TCB chain until it finds a job step TCB which was in existance at the last scan (new names imply new jobsteps and are ignored). During initiation of its scan SMOOTH sets an initial comparison interval (INT_o) of 5oo msecs., and compares the CPU time used by the highest priority task with this time interval. If the task has used 75 % of INT_o it is considered as CPU-bound. Otherwise the interval is reduced by the CPU time used by this task:

$$INT_1 = INT_o - CPU_o$$

and the CPU time used in the next lowest priority job step TCB is compared with INT_1. This interval reduction is repeated until a job using $CPU_n \geq 75$ % of INT_{n-1} is found. If in this case

$$INT_n \leq INT_o/4 \quad \text{(i.e. 125 msecs)}$$

the step is ignored since it shows too little resolution. When a CPU-bound job step is found , a flag is set and SMOOTH does not attempt to find other CPU-bound tasks in that scan.

The aim of a priority dispatcher is to schedule the use of computer resources efficiently by overlapping as far as possible CPU and I/O activity. This is achieved by various dispatchers in different ways, but all work on the basic principle of allowing I/O-bound tasks higher DP than CPU-bound tasks. Since the CPU is needed to initiate I/O activity, I/O-bound tasks must have high priority access to the CPU if they are to run at all. When such a task begins to transmit across the channel the CPU is released to the lower priority tasks. When the data transmission finishes, the task again has high priority access to the CPU to initialise its next I/O operation.

SMOOTH is programmed to reduce the DP of CPU-bound jobs as follows: having found a CPU-bound job step, as described above, SMOOTH compares its DP with the value 12 (A step of DP=12 has already been found to be CPU-bound by SMOOTH and has had its priority reduced using the CHAP macro). If it is not of priority 12, the DP is set to 12 using the CHAP macro. CPU-bound job steps of DP=12 have their "remaining" CPU-time compared with 5 mins. If the remaining CPU time is less than 5 mins. a flag is set telling SMOOTH to wait until the task again becomes CPU-bound before using the CHAP macro to move its TCB to the lowest position in the DP=12 group in the task queue. Otherwise it is moved to this position immediately. If the remaining CPU time is also less than 1 min. the flag is set to indicate that SMOOTH should wait until the task becomes CPU bound twice again before pushing it to the bottom of the chain. Thus a form of self-adjusting time-slicing is achieved for all tasks which have been reduced to DP=12.

The time slice is:

≥ 3 SMOOTH intervals	for job steps of small remaining CPU time (1 min.)
≥ 2 " "	for job steps of medium remaining CPU time (5 min.)
≥ 1 " "	for job steps of large remaining CPU time (5 mins.)

The actual time slice for each task is variable and depends upon the whole job mix.

It should be noted that once a step has had its DP reduced to 12, it has no way of increasing its priority. It may, however, by virtue of no longer being CPU-bound, remain at the head of the DP=12 group, which is equivalent to an increase in priority. DP=12 is the lowest effective priority in the dispatching system, and thus I/O-bound jobs which have never been CPU-bound will run with higher dispatching priorities than this. Those which have also been CPU-bound at any stage, but which are now I/O-bound will remain at the top of the DP=12 group.

3. The Simulation Program

The flowchart of the program used to simulate SMOOTH is shown in Fig. 2. The program is rather complicated due to the fact that GPSS/360 is designed to simulate running systems. It cannot readily be adapted to stop such systems and reorganise them at fixed intervals, as required by SMOOTH. The simulation of SMOOTH is effectively two simulation problems combined:

1) A priority system in operation on a running computer (CPU and I/O priority schemes being different)
2) The computer in a "static" condition with SMOOTH examining priorities at any one instance in time

and altering them as required. The system must then be restored to state 1) without disturbance.

The program is divided effectively into two sections to deal with this.

3.1 Simulation of the computer running between SMOOTH scans (Fig. 2(a))

At the start of the simulation a number of transactions is initiated by GPSS to simulate the jobs running in the system. (n-1) of these jobs represent problem jobs and 1 represents the SMOOTH job which is immediately put into a wait state on a User Chain. For the sake of simplicity it is assumed that each job consists of a single job step, with initial "remaining" CPU time equal to the default system value for the execution step of a job of its selection priority.

The following parameters are assigned to each job as it is initiated:

1) A DP. This is dependent on the job-class. For example a mix might consist of

   a) 2 F jobs of pty. 8o say
   b) 2 A jobs " " 7o say
   c) 1 C job " " 6o say.

   The actual values of DP assigned are unimportant as long as they are consistant with the system priority scheme and greater than the priority 12 of a CHAPed job.

2) Total number of SMOOTH intervals the simulation should run. 3ooo intervals were simulated, representing 25 mins. of real-time running of the computer.

3) An initial "remaining" CPU time as described above e.g. lo secs. for F jobs, 3 mins. for A jobs, lo mins. for C jobs etc.

4) An initial actual remaining CPU time, representing the actual amount of CPU time the job will run, calculated from 3) above by multiplying it by a random factor $o < f < 1$.

5) The time-slice flag (number of SMOOTH intervals to wait before CHAPing CPU-bound jobs) is set to zero.

Each job is then put into a queue waiting to gain access to the CPU. Jobs are allowed to take the CPU according to their DP. 2 jobs of the same effective DP in the real system, e.g. 2 F jobs, are simulated as two jobs of DP=81 and DP=8o in order to retain the FIFO structure of the TCB chain. Thus priorities 8o-89 all represent F-jobs, 7o-79 A jobs etc., the highest priority in each group representing the first job of this class to enter the system. When a job ends and a new job of this class is initiated it takes the lowest DP of its group (e.g. 8o for F jobs) and other jobs in the group have their priorities shuffled upwards as required.

The highest priority job in the queue takes the CPU and holds it for a certain length of time, depending on a random time function assigned to this job. For example, the CPU interval allocated might be random within the range 3oo±5o msecs. The job relinquishes the central processor after its assigned time, goes to queue for the I/O channel, and the next highest priority job in the queue takes the CPU. However, if whilst it is in control of the CPU, the real time elapsed becomes an exact multiple of 5oo msecs., the job relinquishes the CPU and is queued onto a User Chain by DP, together with all the other jobs in the system, so as to simulate the stationary TCB structure at this time. The CPU interval left to run for the jobs is stored, so that the next time it runs it will take the CPU for this time.

A job, on using up its CPU interval, goes to queue for the I/O channel. In the simulation only one channel is considered for the sake of simplicity, although in the real system there may be more than one I/O channel in operation. The channel is allocated

to tasks on a strictly FIFO basis which does not depend on DP. This is the case in the 360/91 system for almost all I/O channels, including the tape channels. The same considerations as specified above for the CPU apply to the I/O channel. Each time SMOOTH runs the job holding the channel releases it, and is queued onto the same User Chain as the job holding the CPU and the jobs in both the I/O and CPU queues. The User Chain then reflects the TCB chain in order of DP. I/O left to do is noted, so that this job can continue its I/O (as if uninterrupted) after the SMOOTH scan. When a job releases the I/O channel it returns to the CPU queue to initiate its next I/O action.

Running totals are kept of the total I/O and total CPU performed by each job in each SMOOTH interval, and totally throughout the job.

3.2 The SMOOTH Scan

Every 500 msecs. the SMOOTH transaction (job) is removed from its special User Chain and allowed to run. It unlinks the problem jobs one by one in order of DP from their User Chain, as shown in Fig. 2 (b). The CPU time used by the job during the previous interval is examined according to the rules specified previously in the description of SMOOTH (Section 2), and the first CPU-bound job found has its DP reduced to 12 unless its time-slice flag is set. Subsequent jobs, and the lowest priority job are always ignored. The jobs are then queued again on another User Chain in the new order of DP.

Fig. 2(c) represents necessary DP manipulation so that the TCB priority structure can readily be handled by GPSS. As previously described two Fjobs must have different DP's in order to reflect their order of initiation. Similar considerations must apply to CPU-bound jobs which have had their DP's reduced to DP=12 by SMOOTH. If each of these jobs is allowed to have priority 12 in the model, GPSS handles the jobs incorrectly, without retaining

the task queue structure. The job last CHAPed must take dispatching priority 12, and other jobs in this task queue group must have their priorities incremented to make room for it at the bottom of the queue. Priorities 12-2o are reserved for this purpose. Thus 3 jobs, all of which would have DP=12 in the real system are simulated as having DP's of 14,13 and 12, reflecting the order in which they were CHAPed and thus their real order of priority. The last part of the program deals with priority shuffling, both for CHAPed and ended jobs, and the reinitialisation of ended jobs.

At the end of the SMOOTH scan, jobs which were either holding or waiting for the CPU are returned to the CPU queue, the job of highest DP (which may not be the same one as was highest before) taking the CPU. The job which was holding the I/O channel is allowed to take it again to finish its I/O, and the I/O queue is restored to its exact condition before SMOOTH ran. In this way the I/O system is undisturbed by the scan.

4. Results

Initially the simulation was run with 3 jobs assumed to be running in the computer. These were defined as follows:

1) F job, DP=8o, initial "remaining" CPU-time = 1o secs.
2) A job, DP=7o, " " " " = 3 mins.
3) C job, DP=6o, " " " " = 1o mins.

Comparisons were made between the system running with and without SMOOTH. The same program was used in both cases, but in the latter case the effect of SMOOTH was supressed. This represents the system running without SMOOTH but with a time-slice every 5oo msecs. The effect of this time-slice is, however, minimal in the system simulated, as CPU intervals chosen for the jobs were always substantially less than 5oo msecs.

Table 1 shows the results of a series of simulations of the computer running with and without SMOOTH. The number of jobs

passed of each type is shown, together with the percentage CPU and I/O utilisation as a function of real-time for each job class. The figures for the number of jobs passed are only given as a guide, since job lengths in terms of CPU-time are randomly assigned. Thus when only a few jobs have been passed (A and C jobs) the values may be misleading. The CPU and I/O figures are an accurate guide to the use of the CPU and channel by a particular job class.

Also tabulated are the average queues at the CPU and I/O channel over the whole real-time running of the simulation. Similarly the total average CPU and I/O utilisation as a fraction of total real-time is shown. The CPU/I/O overlap is calculated for each simulation, followed by the increase in overlap produced by SMOOTH as a percentage of the overlap without SMOOTH.

Initially all the jobs were run with the same CPU and I/O functions:

e.g. CPU intervals random within the range 3oo-48o msecs.
I/O " " " " " 1 -1oo msecs.

These functions were varied to simulate different cases:

simulation number	
1	all jobs CPU-bound but not I/O-bound
2	" " I/O-bound " " CPU-bound
3	" " CPU-bound and I/O-bound
4	" " neither CPU-bound nor I/O-bound

None of these cases shows any increase in I/O and CPU overlap as a result of the use of SMOOTH, since I/O and CPU utilisation is quite efficient without it. However, it should be noted that in cases 1 and 3, SMOOTH does have a considerable "smoothing" effect on the allocation of the CPU to different jobs. The lower priority jobs are allowed more access to the CPU than without SMOOTH, and the higher priorities less. As would be expected, in cases 2 and 4, SMOOTH has negligible effect, as there are no CPU-bound jobs in the system.

Secondly comparisons were made with each of the jobs having different I/O and CPU functions, each job being either CPU- or I/O-bound but not both or neither. The results are tabulated in simulations 5-1o. "CPU" and "I/O" written in the second column imply:

a) CPU - CPU-bound job, where the CPU function is random between the limits 3oo-48o msecs. and the I/O function is random between the limits 1-1oo nsecs.

b) I/O - I/O-bound job, where the I/O function is random between the limits 3oo-48o msecs. and the CPU function is random between the limits 1-1oo msecs.

Amongst simulations 5-1o, numbers 6 and 7 are the most interesting. Simulation 6 shows that SMOOTH fulfils its task of allowing a low-priority I/O-bound job access to the CPU to initiate its I/O even though CPU-bound jobs of higher initial DP exist in the system. The C-job, which has about 8o % of the total I/O to do, is allowed to take the channel when required, whereas without SMOOTH it had very little access. The CPU/I/O overlap is increased by 3oo %. Simulation 7, shows the same effect for the A-job. In this case the increase in overlap is less, since the I/O-bound job had much better access without SMOOTH than in simulation 6, due to its higher initial DP. The other simulations show very little effect due to SMOOTH, and on careful consideration of each individual case this is what would be expected. In case 5 the I/O job already had high DP and in cases 8-1o the CPU is not loaded so that the I/O-bound jobs do not have to queue long for it.

Simulations 11-2o show an additional feature. Here jobs may be both CPU- and I/O-bound. (Jobs which are neither CPU- nor I/O-bound have not been considered as they represent no load on the system and are rarely affected by SMOOTH). In the second and third columns are given the attributes of each job (F, A, C in order), implying:

a) CPU - CPU function random between the limits 300-480 msecs. (CPU-bound)

b) ¬CPU - CPU function random between the limits 1 -100 msecs. (not CPU-bound)

c) I/O - I/O function random between the limits 300-480 msecs. (I/O-bound)

d) ¬I/O - I/O function random between the limits 1 -100 msecs. (not I/O-bound)

Simulations 11 and 17 show a marked improvement in I/O activity by an I/O-bound C-job when SMOOTH is running. In the first case, however, the highest priority job, an F-job, which is both I/O- and CPU-bound, shows a decrease in CPU and I/O activity. This is to be expected, since SMOOTH reduces the priority of any CPU-bound job, regardless of whether it also has substantial I/O to do. Simulation 19 also shows an improvement in CPU/I/O overlap, this time due to the C-job, (which is both I/O and CPU-bound), gaining more CPU and I/O time. The set of simulations shows that in general a low-priority I/O-bound job improves its I/O activity when SMOOTH is running. This is most noticable when the job is not itself CPU-bound, and the other jobs are. When all the jobs are I/O-bound, the FIFO action of the I/O queue makes any improvement negligible. The improvement is less if the I/O-bound job is also CPU-bound. I/O-bound jobs of higher initial priority show no improvement, as might be expected, since both with and without SMOOTH they have high priority access to the CPU.

Simulations 21-28 show that although SMOOTH often increases I/O/CPU overlap, there are certain cases where it may be a disadvantage rather than an advantage, and other cases where perhaps a more sophisticated algorithm would have a better effect. All these cases involve jobs which SMOOTH considers to be CPU-bound, but which also have a reasonable amount of I/O to do. These jobs are reduced to low priority by SMOOTH, and thus may have less chance of getting the I/O channel than without SMOOTH. It should be noted, however, that a strict comparison of the system with and without SMOOTH does not necessarily show the full picture. The system without SMOOTH represents the worst possible case. The real aim of SMOOTH must be

not only to make an improvement in I/O/CPU overlap relative to this, or at least not to make things worse, but to distribute the CPU and I/O activity relative to the percentage needs of the different jobs, so as to maximise both I/O and CPU utilisation.

Simulations 21-28 show the limitations imposed by SMOOTH on jobs which it considers to be both CPU- and I/O-bound. In these simulations I/O in column 3 implies that the I/O function is random between the limits 45o-48o msecs., except in simulation 22, where it has been increased to 6oo-9oo msecs. All these simulations include two CPU-bound jobs, and one which is both CPU- and I/O-bound, or one of the former and two of the latter type. Several of the simulations show that the CPU/I/O overlap is actually decreased by SMOOTH when the jobs which are both I/O- and CPU-bound start in the higher priority F-job and A-job positions. This is because they are CHAPed by SMOOTH without consideration of the I/O activity they have to do. Case 23, which shows an improvement is still not as efficient as it might be. The A-job has 8o % of the total I/O to do. This total is 5o % of the total CP to do. Thus in the case where all jobs were given equal CPU access, the A-job should hold the I/O channel 4o % of the time for maximum efficiency. If it were given highest priority, by an improved SMOOTH algorithm, this utilisation would be improved. In fact, SMOOTH only allows the A-job to hold the channel 23 % of the time. Similarity in case 25, the A-job and C-job should each obtain the I/O channel for at least 4o % of the time. In fact they obtain only 35 % and 3 % respectively. In both these simulations the I/O channel is consistently underutilised, and the other simulations in this group show the same effect to a lesser degree.

It should also be noted from simulations 26-28 that when all the jobs are A-jobs which run for a long time (DP always in the DP=12 group) I/O utilisation is even worse. Jobs with long "remaining" CPU time do worse than, say, F-jobs, from this point of view, as they are CHAPed every time they are found to be CPU-bound due to the time-slice effect described in Section 2. They also

always have long remaining CPU time.

Thus simulations where the I/O channel utilisation is low relative to that possible (depending on the ratio of CPU to I/O required) show that the SMOOTH algorithm is not optimum in certain cases. Simulation 23, for example, shows only 3o % use of the I/O channel even though for maximum efficiency the channel should be used at least 5o % of the time (more if the I/O-bound job were given highest priority). If the A-job were using a tape channel, the tape would appear to move in jerks, an occurence which has been noticed at IPP.

Simulations 29 and 3 show that a job which changes from being CPU- to I/O -bound may not be treated any worse than if it had always been I/O-bound. In these simulations the A-job and F-job, respectively, started as CPU-bound, but became I/O-bound as soon as they were CHAPed. However, they soon rose to the highest priority position in the DP=12 group of the task queue, and so managed nevertheless to do their I/O. The CPU-bound jobs in the mix were continually being CHAPed and so stayed in the lower positions of the DP=12 group.

Finally, for the sake of completeness, simulations were run with 5 jobs, representing 2xF-, 2xA- and 1xC-job. These results showed exactly the same trends as for the 3-job mix, and so have not been included.

5. Conclusion

This study of SMOOTH shows that in general SMOOTH performs its functions well by:

1) Sharing out the available CPU-time for CPU-bound jobs
2) Discriminating slightly against the longer jobs (time-slice feature)

3) Allowing low initial DP I/O-bound jobs access to the CPU to initiate their I/O even though there may be CPU-bound jobs of higher initial DP in the system.

The algorithm does, however, have a disadvantage when jobs which it considers to be CPU-bound are also relatively I/O-bound. Such jobs have bursts of CP greater than about 3oo msecs., and bursts of I/O of greater than 4oo msecs. If the CPU is busy processing other CPU-bound jobs in the system, the job which is also I/O bound finds it difficult to obtain the CPU to initiate its I/O, since it has low DP (it is CHAPed when SMOOTH finds that it is CPU-bound). Such a job may obtain the I/O channel much less often than it requires it, and this effect has been noted on the tape channel at the IPP. SMOOTH might be improved by reducing jobs which are both CPU- and I/O-bound to a higher priority than those which are only CPU-bound.

Although SMOOTH represents a considerable improvement in some cases over a system running without it, a more sophisticated algorithm which considers both the I/O- and CPU-requirements of a job would be preferable to optimise the priority dispatching on the 36o/91.

FIG. 1

FLOWCHART OF SMOOTH ANALYSIS LOOP (slightly simplified)

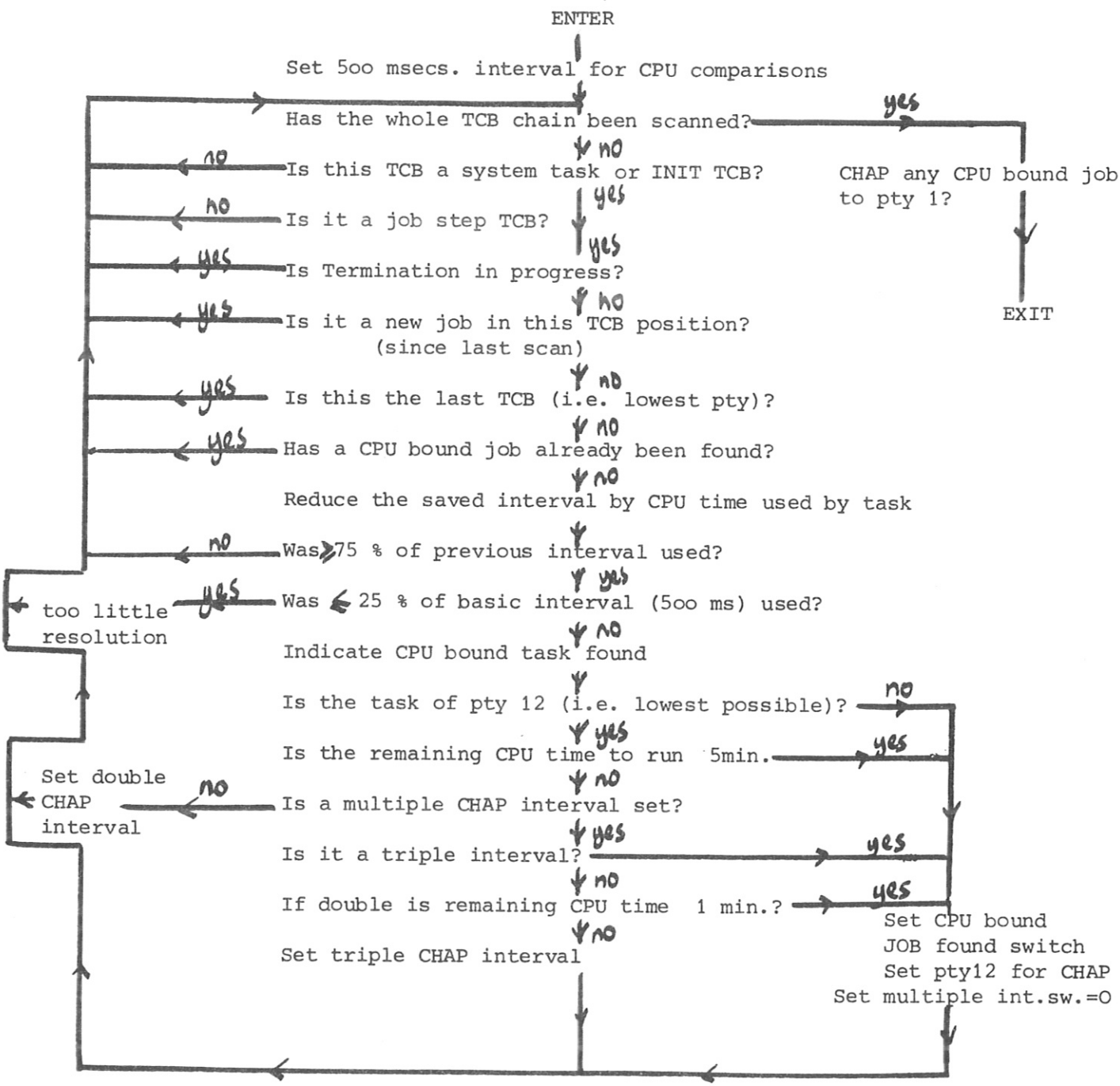
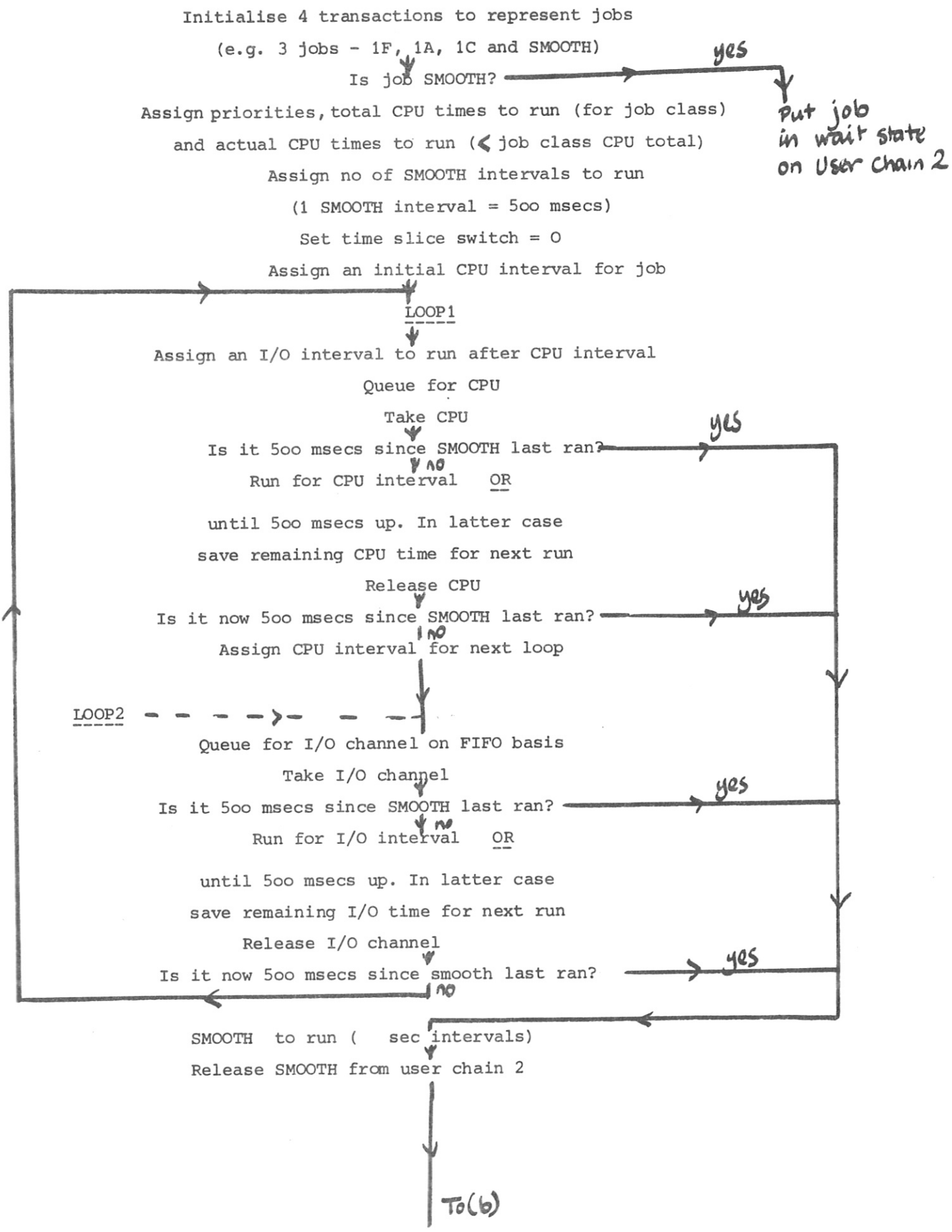
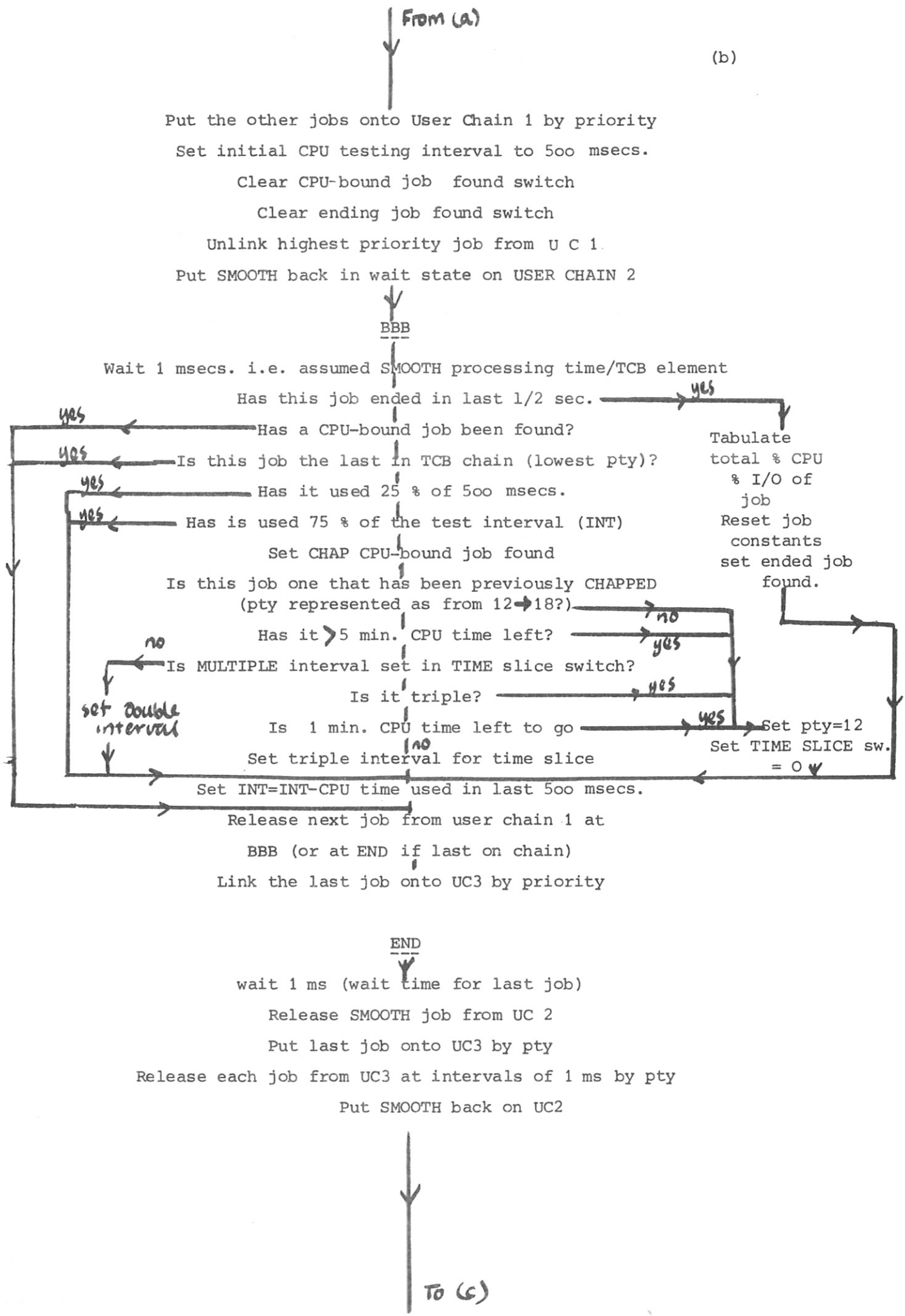
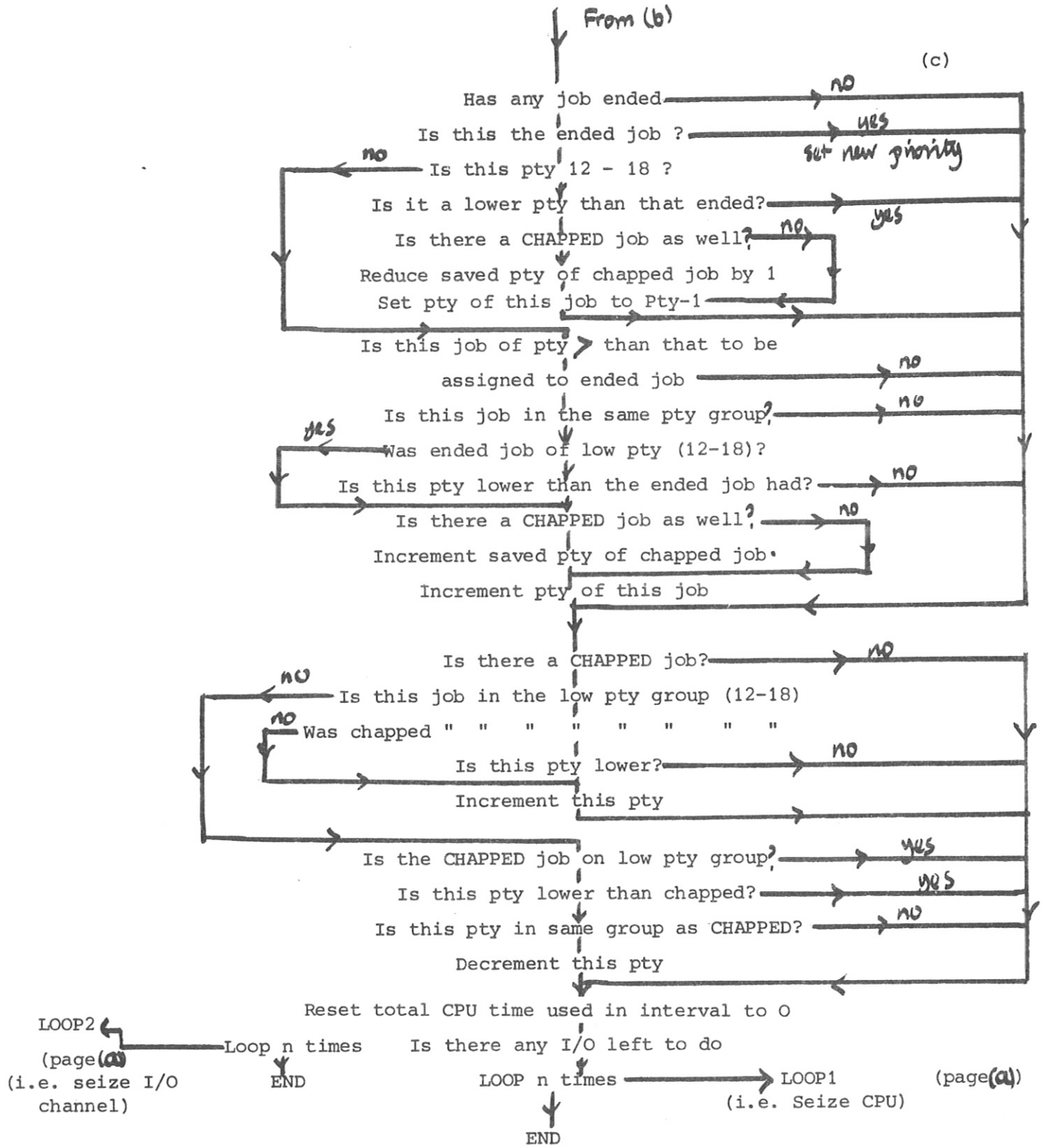


FIG. 2 (a)

FLOWCHART OF SIMULATION OF SMOOTH




From (a)
(b)
Put the other jobs onto User Chain 1 by priority
Set initial CPU testing interval to 5oo msecs.
Clear CPU-bound job found switch
Clear ending job found switch
Unlink highest priority job from U C 1
Put SMOOTH back in wait state on USER CHAIN 2
BBB
Wait 1 msecs. i.e. assumed SMOOTH processing time/TCB element
Has this job ended in last 1/2 sec.
yes
Tabulate total % CPU % I/O of job
Reset job constants
set ended job found.
yes
Has a CPU-bound job been found?
yes
Is this job the last in TCB chain (lowest pty)?
yes
Has it used 25 % of 5oo msecs.
yes
Has is used 75 % of the test interval (INT)
Set CHAP CPU-bound job found
Is this job one that has been previously CHAPPED
(pty represented as from 12→18?)
no
Has it > 5 min. CPU time left?
yes
no
Is MULTIPLE interval set in TIME slice switch?
set double interval
Is it triple?
yes
Is 1 min. CPU time left to go
yes
Set pty=12
Set TIME SLICE sw. = 0
no
Set triple interval for time slice
Set INT=INT-CPU time used in last 5oo msecs.
Release next job from user chain 1 at
BBB (or at END if last on chain)
Link the last job onto UC3 by priority
END
wait 1 ms (wait time for last job)
Release SMOOTH job from UC 2
Put last job onto UC3 by pty
Release each job from UC3 at intervals of 1 ms by pty
Put SMOOTH back on UC2
To (c)


From (b)
(c)
Has any job ended
no
Is this the ended job ?
yes
set new priority
no
Is this pty 12 - 18 ?
Is it a lower pty than that ended?
yes
Is there a CHAPPED job as well?
no
Reduce saved pty of chapped job by 1
Set pty of this job to Pty-1
Is this job of pty > than that to be
assigned to ended job
no
Is this job in the same pty group?
no
yes
Was ended job of low pty (12-18)?
Is this pty lower than the ended job had?
no
Is there a CHAPPED job as well?
no
Increment saved pty of chapped job
Increment pty of this job
Is there a CHAPPED job?
no
no
Is this job in the low pty group (12-18)
no
Was chapped " " " " " " " "
Is this pty lower?
no
Increment this pty
Is the CHAPPED job on low pty group?
yes
Is this pty lower than chapped?
yes
Is this pty in same group as CHAPPED?
no
Decrement this pty
Reset total CPU time used in interval to 0
LOOP2
(page a)
(i.e. seize I/O channel)
Loop n times
END
Is there any I/O left to do
LOOP n times
LOOP1
(page a)
(i.e. Seize CPU)
END

Simulation #	CPU time m secs	I/O time m secs	FJOB						AJOB						CJOB						Average Queue at CPU		Average Queue at I/O chan.		CPU Utilisation		I/O Utilisation		Overlap		% increase in Overlap
			No of jobs passed		% CPU use		% I/O use		# jobs		% CPU use		% I/O use		# jobs		% CPU use		% I/O use												
			With Sm.	Without Sm.	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	
1	300-480	1-100	122	199	43	7o	5	9	8	3	36	27	4	3	1	1	2o	2	2	o	1.85	1.84	.oo4	.oo6	.991	.991	.127	.126	.122	.121	1
2	1-100	300-480	14	14	4	4	33	33	1	1	4	4	32	32	1	1	4	4	33	33	o	o	1.85	1.85	.128	.128	.991	.991	.123	.123	o
3	300-480	300-480	99	1o6	35	37	32	34	3	4	32	34	32	34	1	1	29	24	29	24	.43	.44	.63	.62	.963	.96o	.948	.944	.93o	.924	1
4	1-100	1-100	89	85	31	32	31	31	3	7	27	29	27	29	1	1	25	23	25	23	.64	.64	.64	.64	.844	.844	.842	.844	.764	.766	o
5	F=I/O A=CPU C=CPU		25	26	9	9	7o	7o	8	1o	4o	28	6	7	1	1	38	25	45	3	.96	.97	.23	.22	.966	.966	.819	.816	.8o2	.799	o
6	CPU CPU I/O		134	194	48	7o	6	8	8	4	9	7	5	3	1	1	9	o	69	6	.98	1.8	.22	.o2	.968	.99o	.8o7	.184	.72o	.174	3oo
7	CPU I/O CPU		146	199	51	71	6	8	1	1	6	6	69	55	1	1	37	19	5	2	.97	1.2	.23	.15	.967	.982	.811	.667	.794	.658	23
8	CPU I/O I/O		133	142	47	5o	5	6	1	1	44	46	46	46	1	1	6	5	46	45	.o8	.1o	1.3	1.3	.579	.6o4	.99o	.986	.574	.597	-4
9	I/O CPU I/O		18	16	6	6	46	46	7	9	6	6	5	5	1	1	6	6	46	46	.o7	.o8	1.3	1.3	.566	.588	.99o	.989	.561	.583	-4
1o	I/O I/O CPU		18	18	6	6	46	46	1	1	42	43	46	46	1	1	44	44	5	5	.o7	.o7	1.3	1.3	.568	.568	.99o	.99o	.563	.563	0
11	CPU CPU CPU	I/O I/O I/O	126	136	42	46	4o	44	4	5	42	43	5	5	1	1	6	3	47	28	.29	1.o3	.83	.21	.9o8	.977	.941	.787	.878	.775	12
12	CPU CPU CPU	I/O I/O I/O	95	94	34	34	31	32	3	7	32	33	31	32	1	1	4	4	34	33	.1o	.1o	1.2	1.2	.7o	.71	.99	.99	.697	.71o	-2

Table 1 (a)

Simulation run	CPU time in secs	I/O time in secs	FJOB						AJOB						CJOB						Average Queue at CPU		Average Queue at I/O		CPU utilisation		I/O utilisation		Overlap		% increase in Overlap
			No of jobs passed		% CPU use		% I/O use		# jobs		% CPU use		% I/O use		# jobs		% CPU use		% I/O use												
			S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	
13	¬CPU CPU CPU	I/O I/O ¬I/O	18	17	6	6	48	47	6	7	41	42	41	42	2	1	42	39	5	5	.28	.26	.84	.86	.9o	.9o	.95	.96	.875	.868	1
14	¬CPU CPU CPU	I/O I/O I/O	11	11	4	4	34	33	6	6	32	33	32	32	1	1	32	31	32	31	.o8	.o8	1.2	1.2	.69	.69	.99	.94	.686	.684	o
15	CPU ¬CPU CPU	I/O I/O ¬I/O	126	132	42	45	4o	43	1	1	6	5	47	45	1	1	4o	4o	5	5	.29	.29	.83	.83	.92	.91	.91	.94	.891	.882	1
16	CPU ¬CPU CPU	I/O I/O I/O	92	94	34	34	31	32	1	1	4	4	34	33	1	1	32	31	32	31	.o9	.o9	1.1	1.2	.7o	.7o	.99	.99	.694	.697	o
17	CPU CPU ¬CPU	¬I/O I/O I/O	128	181	45	63	5	7	7	4	4o	3o	39	3o	1	1	6	2	48	2o	.35	1.18	.77	.23	.9o	.97	.94	.93	.892	.574	5o
18	¬CPU CPU CPU	I/O ¬I/O I/O	19	18	6	6	48	5o	6	9	43	48	5	6	1	1	4o	38	4o	38	.28	.32	.83	.79	.9o6	.92o	.949	.931	.98o	.987	-1
19	CPU ¬CPU CPU	¬I/O I/O I/O	132	18o	46	63	5	7	1	1	6	6	49	51	1	1	38	22	38	22	.37	.86	.76	.38	.9o9	.92o	.935	.815	.876	.773	12
2o	¬CPU ¬CPU ¬CPU	I/O ¬I/O ¬I/O	29	29	1o	1o	77	77	1	1	1o	1o	1o	1o	1	1	1o	1o	1o	1o	.o9	.o9	1.6	1.6	.3o5	.3o5	.982	.982	.297	.297	o
21	CPU CPU CPU	I/O ¬I/O ¬I/O	95	116	33	4o	37	43	5	4	36	39	4	5	1	1	28	19	3	2	1.o4	1.4	1.3	.9o	.985	.983	.476	.521	.468	.513	-1o
22	as but =6oo	21 I/O -9oo	84	93	3o	33	55	6o	7	3	33	33	4	4	1	1	3o	27	3	3	1.o	1.o	.34	.37	.946	.939	.64o	.691	.613	.661	-7
23	CPU CPU CPU	¬I/O ¬I/O I/O	126	195	44	7o	5	2	4	8	36	27	4	3	1	1	19	2	23	2	1.6	1.8	.o7	.o1	.988	.991	.335	.152	.329	.148	12o

Table 1 (b)

Simulation #	CPU time	I/O time	FJOB						AJOB						CJOB						Average Queue at CPU		Average Queue at I/O		CPU Utilization		I/O Utilization		Overlap		% increase in overlap
			No of jobs passed		% CPU use		% I/O use		No of jobs		% CPU use		% I/O use		No of jobs		% CPU use		% I/O use												
			S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	S	¬S	
24	CPU CPU CPU	I/O I/O I/O	1o3	116	35	39	39	42	3	4	33	39	39	46	1	1	28	17	3	2	.84	.76	.33	.33	.97o	.961	.834	.926	.82o	.9o6	-1o
25	CPU CPU CPU	I/O I/O I/O	121	187	42	67	5	8	2	3	29	21	35	26	1	1	27	1o	3	1	1.4	1.6	.11	.o6	.985	.987	.444	.356	.437	.349	3o
26	as 21 but all A JOBS		1	1	26	39	31	46	1	1	33	39	4	5	1	1	35	19	4	2	1.4	1.4	.12	.1o	.986	.981	.419	.34o	.412	.53o	-29
27	as 24 but all A JOBS		1	1	3o	39	35	46	1	1	31	39	42	46	1	1	34	17	4	2	.87	.73	.34	.33	.973	.958	.787	.951	.773	.93o	-18
28	as 25 but all A JOBS		1	1	31	67	4	8	1	1	3o	21	35	25	1	3	34	9	4	1	1.4	1.6	.14	.o6	.984	.987	.447	.357	.437	.349	3o
29	as 6 but with AJOB CPU→I/O		14o	194	49	7o	5	8	4	4	39	28	5	3	1	1	8	o	69	6	.98	1.8	.22	.o2	.968	.99o	.8o7	.181	.32o	.171	3oo
3o	as 2o but with FJOB CPU→I/O		32	29	14	1o	72	77	1	1	1o	1o	1o	1o	1	1	1o	1o	1o	1o	.13	.o9	1.6	1.6	.322	.3o5	.967	.982	.3o5	.297	3

Table 1 (c)