Scheduling in a general purpose operating system*

by V. A. ABELL, S. ROSEN and R. E. WAGNER

Purdue University Computing Center
Lafayette, Indiana

INTRODUCTION

In recent years there has been a great deal written and published about scheduling and storage management in time sharing systems. During the same period there has been a significant trend toward the development of more general purpose operating systems on large computers. Such systems support a high volume batch processing operation and at the same time provide modes of computation usually associated with time sharing systems. They are multiprogramming and multiprocessor systems that execute jobs that enter the job stream from local and remote card readers, and from local and remote on-line consoles. Some jobs are interactive during execution and some are not. Many jobs use interactive file creation and editing and debugging processors even though they are basically batch jobs.

This paper describes some aspects of an operating system of this type that is now running at the Purdue University Computing Center on a CDC 6500 supported by an IBM 7094. The paper deals mostly with the scheduling mechanisms and strategies used in the system. These mechanisms and strategies are probably not new, since all kinds of scheduling disciplines have been proposed and discussed in the literature. However, we believe that this is the first time that scheduling and job movement techniques of the type described here have been implemented and used in a very large system with the high job volume and diversity that characterize a large university computing center.

* The work described in this paper was partially supported by the National Science Foundation under Grant No. GJ-245 for the establishment of the Indiana Regional Computer Network. It was also partially supported by Control Data Corporation through a research grant to the Purdue University Computing Center.

The Purdue MACE operating system

The Purdue MACE operating system is based on the MACE2 operating system which was originally designed by Mr. Greg Mansfield of the Control Data Corporation.

MACE is an outgrowth of the first operating system for the Control Data 6000 series that was developed at CDC's Chippewa Falls Laboratory. The underlying design of that first system, the Chippewa Structure, has formed the basis for several of the most successful operating systems for the CDC 6000 series. These include SCOPE 2.0, SCOPE 3.0–3.4, and MACE.

The Chippewa Structure is successful, to a large degree, because it is closely integrated with the unique hardware organization of the CDC 6000 series. The organization consists of one or two central processors (CPU's), and ten peripheral processors (PPU's), all of which share a large, fast central memory of 60 bit words. The CPU minor cycle time is 100 ns, while for the PPU it is one microsecond.

The peripheral processors each have a full instruction complement, including arithmetic, shift, and input/output instructions, and 4,096 12 bit words of private storage. They share access to twelve, one megacycle 12 bit wide data channels. The PPU's are primarily designed for input/output tasks, communicating through the common central memory with the CPU, which is used mainly to perform computational tasks for executing programs.

CDC markets several variants of the 6000 design, each of the same structure, differing from the others only in CPU configuration. The 6600, the fastest system, has a CPU with parallel arithmetic units. The 6400 has a slower CPU with sequential arithmetic units. The 6500, which is the system in use at Purdue, has two 6400 CPU's. The 6700 has one 6600 and one 6400 CPU.
Central memory in the Purdue 6500 system consists of 65,536* 60 bit words. The memory is organized in phased banks with access time of 100 nsec and cycle time of 1 microsecond.

Central memory organization and the control point

In the Purdue MACE operating system the large central store is divided into a user portion and a central memory resident system area. The system area which now occupies just under 11000 words, contains allocation tables, routine and file directories, a small amount of system central processor code (most of the system executes in the peripheral processors), a number of key peripheral processor routines, and a set of job control blocks, known as control points.

A control point is a pivotal area, occupying 128 words of central memory, through which job execution is controlled, and to which the resources for job execution are allocated. The control point may be thought of as the control element of an individual computer, and the entire set of control points as a division of the hardware machine into a number of separate machines, each of which can execute an independent task.

The number of control points was fixed at eight in the original Chippewa System and was retained at that number in most derivative systems. One control point is allocated to various system overhead functions—storage movement, mass storage space allocation, etc. The remaining control points can be assigned to active jobs, including the control of input-output devices such as card readers, line printers, remote batch stations, and keyboard consoles. While the MACE system retains this control point allocation method, it provides for the optional declaration of as many as 26 control points at system load time.

A job is assigned to an active control point after it has been queued to a system mass storage device (usually a disc storage unit). The resources required for the execution of the job are allocated to the control point. These include central memory space, central processor time, peripheral processor assistance, mountable equipment (tapes, disc packs, etc.), mass storage space, and file pointers.

The resources are allocated to control points through a monitor program which runs in a dedicated peripheral processor. A second dedicated peripheral processor runs a display program, DSD, that provides operator-system communication via a twin screen, display-keyboard console.

The remaining peripheral processors are pooled for input-output and job sequencing functions. Each contains a small resident executive containing communication, overlay loading, and mass storage driver subroutines. The pool peripheral processors constitute one of the resources assigned to control points by the monitor, and execute programs which communicate with the monitor through central memory registers.

The control point area, which occupies a fixed portion of central memory, contains pointers relating to job status, and the resources assigned to the job. Included in the control point area are a 72 word buffer, used to contain the control statements supplied for the processing of the job, and a 16 word area called the exchange package.

The exchange package is used by the system monitor to control CPU allocation. A special hardware instruction, called an exchange jump, permits the monitor to interrupt a running CPU, save its register contents, and load all registers with new contents in a single operation. The exchange jump instruction, which executes in 2 microseconds, uses the read portion of the core memory cycle to obtain a word of register contents from the exchange area, and the write portion to store the previous contents of the corresponding registers of the interrupted CPU.

When a job is at a control point waiting for the CPU, the exchange package area contains the register contents that are required to start or resume processing of the job. When the monitor performs an exchange jump for that control point, the registers are loaded from the control point area, and the control point area is loaded with an exchange package that the monitor uses to return control to the system when the job is interrupted or terminated.

The rapid CPU switching capability provided by the exchange jump operation works in conjunction with a relocation and limit register in each CPU to provide an efficient method of memory allocation. The relocation registers in the CPU permit the assignment of a contiguous region of central memory to a program, which is totally isolated from any other area, and which can be moved rapidly to, from and within the user portion of central memory.

Limitations of the Chippewa structure

While the Chippewa Structure in its basic design permits effective multiprogramming use of the 6000 system, it includes some static elements which seriously limit system performance. The major achievement of the MACE system was the relaxation of some of these control point restrictions. This process has been carried further in the Purdue MACE system.

* The memory is to be expanded to either 98,394 or 131,072 words in the summer of 1970.
The major static control point restriction in the original Chippewa Structure is embodied in the fact that once a job was brought to a control point, that control point was committed to that job until it either completed or aborted. In almost all cases this meant that once a job was brought into central memory it remained there until it was completed. This affected the design of resource allocation and job sequencing to such an extent that control point and job became almost inseparable.

The closeness of the association between control point and job seriously affects the ability of a system of this type to respond to changing job loads. Thus, while the system can schedule jobs to control points on a priority basis, a new job of higher priority which enters the queues normally must wait until terminating jobs release sufficient resources.

Early attempts to resolve the problem resulted in processors which permitted the system operator to manually suspend the processing of a control point job, and to dump its allocated core memory to disc storage in order to permit another to be loaded and processed. This process was severely limited by the slowness and inaccuracy of operator intervention, and by the fact that it did not free the control point even though the job itself was no longer in memory.

The Autoroll system

One of the major advances of MACE over the earlier implementations based on the Chippewa model is the ability of the system itself to suspend a job and free its control point. The process, called rollout, consists of the copying of the complete job status, including control point data, to a mass storage file. The control point thus freed can be assigned to another job. In the reverse process called rollin, a rolled out job can be assigned to any available control point, the data in the file can be copied back into any available area in main memory, and the job can be resumed.

The scheduling mechanism in the MACE system is of the type that has been called preempt resume in some recent publications. Among users of CDC 6000 series equipment it is more frequently referred to as an autoroll system. The basic component of the system is a job scheduler that can interrupt jobs and cause them to be rolled out from main memory to make room for other jobs that, at least temporarily, have higher priority. The queues from which the scheduler selects the jobs that are to be brought into memory consist of input files and rollout files. The major function of the scheduler is to use the autoroll mechanism to control the movement of jobs between the job queue and main memory in such a way as to provide for optimum utilization of system resources.

There are many possible job movement strategies that could be implemented within the framework of such a system. The particular strategy described here is the one now in use in the Purdue MACE system. It seems to function well in the university environment, and provides adjustable parameters that permit fairly significant changes to be made in response to changes in the character of the job mix.

Job movement strategy

The job movement strategy of the Purdue, MACE system is a dual function of the system monitor and a peripheral processor program, the job scheduler itself. The job scheduler executes on a short, periodic cycle (five seconds in the present system). It is also executed whenever a job sequencing operation changes the state of the machine—e.g., a job terminates or a new job enters the input queue.

The job scheduler is priority driven. Each job in the system carries a single, twelve bit priority value, called the queue priority. A large value signifies high priority; a small value, low priority. Several priority classes and values are reserved for identifying jobs in special states, such as being rolled out or in, manually rolled out, or waiting for some operator action.

Each time the job scheduler executes, it constructs a snapshot of the executing, control point environment. This includes data about the jobs running and the resources allocated to them. Against this picture, the job scheduler matches the jobs awaiting execution in the input and rollout queues.

In descending order of queue priority value, the job scheduler compares the resources required by jobs in the queues against those available or in use by jobs of lesser queue priority. In the simplest case, where sufficient unused resources are available, the job scheduler requests the assignment of a peripheral processor to the job by the monitor. That processor proceeds to roll in the job or begin its execution for the first time, while the job scheduler continues to search the job queues.

When a waiting job requires resources in use by executing jobs, the scheduler must consider the nature of the resources required. Many of them, such as central memory, the control point, central processor usage, and file pointer space, can be reassigned, since the rollout file will carry the status of their usage. Others, such as magnetic tape units, remain assigned to the job for its duration for practical reasons.

After the job scheduler has selected a job for which resources can be made available, it constructs a rollout
sequence which will free the required resources. The rollout sequence is built from the list of running jobs whose queue priorities are lower than that of the job being scheduled. Central memory space and control point availability are the two factors considered.

Rollout density is controlled by the system monitor. In the normal job scan cycle, a job marked for rollout is assigned a peripheral processor by the monitor, unless a prespecified number of rollouts are already in progress. In the Purdue MACE system, the monitor limits the number of concurrent rollouts to two.

Once the job scheduler has started a rollout sequence, rather than wait for the sequence to complete, it continues to search for lower priority jobs which can be assigned to control points without affecting the rollout sequence, or starting another sequence. When the scheduler exhausts the lists of waiting jobs, it terminates.

The scheduler is recalled periodically, at the end of each rollout step, or when some other job sequencing operation changes the state of the machine. When recalled, the scheduler builds a new snapshot of the environment, effectively “forgetting” the job which started the rollout sequence. Because the scheduler “forgets” that job, it can respond very quickly to changes in the queues. Thus, for example, if a job enters the queues with a priority higher than the one which started the rollout sequence, that job can be executed first. Or, for example, if a job outside the rollout sequence terminates before the sequence is complete, the job causing the rollout sequence can be assigned for execution as soon as the required resources become available.

On a sub-multiple of its basic period, the job scheduler executes an overlay which adjusts queue priorities. The queue priority adjustment overlay modifies the priorities of jobs in the input-rollout queues, and those of jobs in execution at control points. The modification of priorities for queued jobs is essentially an aging operation, to insure that jobs of equal starting priority and resource requirements proceed on a more-or-less first-in, first-out basis.

The queue priorities of jobs in execution are modified as a major tactic in queue balancing. This modification is a portion of a three level management of job queue priority, in which the queue priority of a job is set to a high value when the job enters the input queue, is dropped to a lower value after an allotment of execution time has elapsed, and is incremented each succeeding time the job reaches a control point.

When a job enters the input queue, it is assigned two queue priority values, a “first pass” and an “execution” priority, both based on its resource parameters. The first pass queue priority is based upon a user specified (but account limited) value, an input increment, and an origin increment. Currently each job receives an input increment of 6000, points, and an origin increment of zero for local batch, 100, for remote batch, 300, for remote teletype, and 500, for interactive origination. The user value ranges from zero to 24s.

The second queue priority value is based upon job parameters and account code classifications. The job parameters include central memory requirement, central processor time requested, and the predicted output volumes. The execution queue priority value is constructed from a table of range increments for each parameter. In general, the larger the parameter the smaller the increment it will add to the execution queue priority.

When the job input file is completed, it is queued at its first pass queue priority value. The execution queue priority value is stored in the job input file. When the job reaches a control point, the execution queue priority is stored with other job description parameters in a control point area. Thus it is available to the queue priority adjustment overlay of the job scheduler.

In scanning control point jobs, the queue priority adjustment overlay is preset to consider those jobs which have accumulated a specified amount of execution time. When a job has reached that level, its first pass queue priority is replaced with the execution value. In almost all cases the result is a drop in queue priority.

Currently, the first pass queue priority is replaced by the execution priority after a job has accumulated a total of twenty five seconds of central and/or peripheral processor usage. With a large input stream volume, the modification usually results in the rollout of the job. However, in the Purdue job mix, 75 percent of all jobs complete before the modification takes place. For the user, the chosen time increment permits rapid turnaround for compilation-debugging runs, and usually guarantees that a job which aborts because of compilation errors will pass through the system very rapidly.

The remaining jobs which do not complete before the queue priority modification takes place must run to completion at their execution queue priority values. Several factors combine to enhance their throughput. The first is a dynamic storage reduction performed by the relocatable loader. This improves job throughput because compilation and loading usually require more memory space than execution and usually complete before the queue priority modification takes place. Thus the additional execution time which the job requires can often take place at the reduced field length set by the loader.

Secondly, jobs are aged by the scheduler's queue
priority adjustment overlay. Thus as a job remains in the queues, its priority gradually increases. Finally, each job which is scheduled to a control point receives a small, additional queue priority increment.

The control point increment, which is currently set to four aging units, is designed to protect the rollin time investment. The job is given a queue priority boost in an attempt to keep it in execution for a long enough time to make its rollin time cost reasonable. Otherwise, one could easily envision a job mix in which rollin-rollout operations enter a rapid cycle, induced by the aging process.

**Control point and central processor utilization**

The Purdue MACE system typically runs in an eleven control point configuration with one control point allocated to basic system functions as described in an earlier section. Three others are reserved for use by system input-output processors, one for the queuing (spooling) of peripheral I/O, one for remote batch terminal control, and one for PROCSY, an on-line console system. These three control points require small amounts of memory, determined by the number of active devices. They use very little central processor time, and a larger amount of peripheral processor time for the input-output operations required.

The remaining control points are used for the execution of user problems. The two central processors are cycled among active jobs on a round-robin basis. Each job at a control point which requires a CPU is allocated one for a 65 millisecond time slice. The exchange jump operation keeps the switching overhead very low. Typically it is less than 100 microseconds per transfer.

A job that issues an input-output request may retain the CPU for the full time slice and attempt to overlap its own computing with its I/O transfers. Alternatively, it may give up the central processor for the duration of the I/O transfer. A job that surrenders the CPU when it makes an I/O request is given another 65 millisecond time slice as soon as the I/O transfer is completed.

Other algorithms for the scheduling of central processors to jobs are being considered, but so far there is no evidence that the other algorithms provide any advantage over the round robin with a relatively short time slice.

**Job mix**

Since the jobs that are running in the system may vary greatly in their demands on system resources, it is good scheduling strategy to attempt to maintain a mix of active jobs at control points that require different resources and that make full use of these resources. Ideally there should be one or two jobs whose demands on CPU time are large compared with their input-output requirements, and one or two complementary jobs which require only small bursts of CPU time, and have a great deal of I/O activity involving non-conflicting devices.

The Purdue MACE job scheduler does not now consider these job mix factors in its calculation of queue priorities, since that would require data about the job profile that is not currently available in a form in which it can be used by system routines. Some job mix factors can be introduced manually in the present system through operator typeins that alter the queue priorities assigned by the system.

A more dynamic automatic scheduling algorithm depends on the measurement and efficient encoding of job parameters relative to CPU and input/output and other resource usage on a continuing basis during the course of the execution of each job.

The effectiveness demonstrated by our current use of the priority structure suggests that it would be possible to incorporate job mix factors in the priority value. We are presently considering a priority evaluator system to be implemented as a secondary level in addition to and separate from the scheduler already described. The priority evaluator would use the job profile data, the machine environment, and scheduling constraint parameters to assign priorities which could provide an improved job mix. This type of priority evaluation could be performed at longer intervals, possibly in terms of minutes, could use the faster capabilities of the central processors, and would not affect the ability of the primary scheduler to react to rapid changes in system load.

**Tapes, disc packs and permanent files**

One of the major advantages of the autoroll system is the fact that it permits the handling of requests for allocation and mounting of tapes and disc packs and the queuing of requests for access to permanent files in such a way that little or no system resources are consumed by a job while it is waiting for equipment to become available or for tapes or disc packs to be mounted.

Consider a job that enters the system with a jobcard parameter that indicates that it will use magnetic tape. The jobcard indicates the maximum number of tape units that will be required in parallel, and to simplify the discussion we shall assume that this number is one. The job is scheduled to a control point based on
its first pass queue priority and processing continues
as for other jobs until a call for a tape occurs in the
control statement stream. When such a tape request
occurs the job is rolled out, and is marked as a job
waiting for a tape unit.

When a tape unit becomes available it is assigned
to the job of highest execution queue priority that is
waiting for a tape. The tape unit remains assigned to
that job until the job terminates unless the job itself
releases the tape unit prior to termination.

The job is not automatically rolled back in by virtue
of the fact that is has a tape unit assigned to it, but
its execution queue priority is raised by 3008 points
to help speed it through the system.

While a job has a tape unit assigned it is rolled out
every time a tape mount request is processed, and
remains rolled out and ineligible for scheduling until
the operator mounts the requested tape and types
a message to that effect.

Since the number of tape units is quite limited, it
is very desirable that a job with assigned tape units
be allowed to run to completion as soon as possible.
This would suggest that the very highest priority
should be given to jobs with assigned tape units. How­
ever, this approach might produce time periods in
which only tape jobs could be run, a situation that
might be intolerable because of the requirements of
on-line users, and the stated goal of providing fast
turnaround for short debugging runs. As in most
aspects of job scheduling, it is necessary to compro­
mise between the very desirable goal of making most
efficient use of a resource (such as tape units) and the
many other goals established for the system as a whole.

Mechanisms similar to those used for tape staging
are used for access to disc packs and for write access
to permanent files. The situation is complicated in
these latter cases by the fact that more than one job
in the system at a given time may require access to a
particular disc pack or permanent file.

Console support

A very large number of jobs come into the System
by way of PROCSY (Purdue Remote On-Line Console
SYstem). PROCSY, which is described in more detail
in another report, uses an IBM 7094 to drive a large
number (50 at present) of teletype consoles as a remote
job entry system for the 6500. The 7094 creates job
files on a common disc pack unit. The jobs are exec­
uted by the 6500 and output may be returned to the
consoles by way of the 7094. Rapid response during
file creation and editing is provided by the 7094. Fast
turnaround for job execution is guaranteed by the
scheduling strategy that provides a special increment
for console origin in addition to the 6000 point first
pass increment. If the job is such that it can be com­
pleted in less than 25 seconds of CPU and/or PPU
time the results will be available at the console very
rapidly. If the job takes more than 25 seconds it will
probably be rolled out one or more times before com­
pletion, and may be in the system for quite a long time.
The user at the console could ask that the results be
stored in the permanent file system for his later re­
terval, or he could simply come back later and list
his output file, or he could divert the output file to
the high speed printers in the computing center.

In addition to PROCSY there are several interactive
systems that can be operating on the System at any
given time. These include NAPSS, the Numerical
Analysis Problem Solving System, PICLS (Purdue
Instructional and Computational Learning System)
ALFIE (Algebraic Language For an Interactive En­
vironment), and CRT, an interactive graphics system
using the CDC 252 graphics console.

The details of handling teletypes and the graphic
console are slightly different from those in PROCSY,
but the basic system is the same. The same scheduler
using the same mechanisms causes the appropriate
programs to be rolled in when needed to handle a line
of text or some other interaction. They are rolled out
when done to free system resources for other jobs.
Here again, if the interaction stays within the basic
first pass time limit, the response time is very good.
If it requires more time per interaction, it is not con­
sidered a proper interactive job in this environment,
and response time may be very poor.

In most cases the same mechanism that guarantees
good turnaround for relatively short batch jobs also
provides good response for interactive users without
placing an intolerable load on system resources.

Efficiency of job movement

The job movement system discussed here differs
from that in most systems using preemptive scheduling
techniques in that the whole job is moved as a unit
between peripheral storage and main memory. Most
systems of this type make use of paging and/or seg­
menting hardware and a software system that moves
parts of a program between peripheral and main mem­
ory as required. In most paging systems, a fixed length
page of 512 or 1024 words is the unit of information
that is moved.

Rollout compared to paging

Consider a job in a paging system. In order for
the job to become active a relatively large number of
pages (the working set according to Denning 5.6) must be loaded. When the job finishes its time slice, the pages that it had been using are scheduled to be rolled out to the paging drum or disc. In an active system it is very likely that all of the storage that was occupied by a job will be needed by other jobs, so that by the time the interrupted job is once again scheduled into core memory none of the pages that it had been using during its preceding time slice are still in core. This situation is almost exactly the same as if the job had been rolled out in its entirety from core memory. Various strategies have been suggested for such paging systems that would roll out all active pages on completion of a time slice. A prepping strategy would then roll the job, or at least a working set of the job, back in when it was again made active. A system of this type comes very close to an autoroll system in which the whole job is rolled out and brought back in when it is reactivated.

There are some advantages in moving a whole job rather than individual pages. These advantages arise because of the greater efficiency of writing tracks rather than individual blocks during peripheral transfers. In the particular storage system in use at Purdue, a half track consisting of 3136 60-bit words is read or written during every 50 msec disc revolution, possibly after an initial delay of 20-100 msec for seek time. It does not take much longer to move the whole job than it would take to move a few selected pages of the job.

There are of course other advantages, and possibly other disadvantages to paging systems. It is not our intention to discuss these here. Rather it is our intention to point out that autoroll systems are not necessarily inherently less efficient than paging systems.

Use of extended core storage

The efficiency of the autoroll system is enhanced in the Purdue MACE System by the use of Extended Core Storage (ECS) as a buffer for the rollout process. Extended core storage is a large core storage system designed to be used for streaming data to and from central memory. In its full configuration, with a minimum of 500,000 60-bit words, a streaming rate of 100 nsec per word can be realized. The present 125K ECS at Purdue transfers data at 400 nsec per word. The 250K configuration scheduled to be installed in the summer of 1970 will increase the streaming rate to 200 nsec/word.

In the Purdue MACE system, whenever a rollout is signalled, the entire central memory field length of the job being rolled out is moved to ECS at the full ECS streaming rate. The space that was occupied in central memory is then immediately released for use by other jobs. The contents of the field length that was streamed to ECS is then moved from ECS to a disc storage file at the same rate as it could have been moved directly from central memory to disc storage. Since the transfer rate to disc storage is about 62000 words per second, the use of ECS to buffer the rollout process makes the field length of central memory that is being freed available from several hundred milliseconds to several seconds earlier than in the unbuffered system.

The reverse process of staging input or rollout files in ECS is under consideration, but its implementation would require some major changes in job movement strategies which are now being studied and evaluated.

Performance

Every job that goes through the system causes a sequence of messages to be written in a system file called DAYFILE. These messages tell when the job entered the input queue, how long it took in compilation and loading, how many times it was rolled out, what error conditions were encountered, when the job entered the print queue, etc. The DAYFILE data is used for billing purposes, and also serves as a data base for a number of programs designed to present a picture of the performance of the system. Some of the details of the programs and techniques used will be presented in another report.

The Purdue MACE operating system was phased into operation during the summer of 1969 and took over as the only production system by the end of August. In the first full month of operation, September, 1969, a total of just over 25,000 jobs were run. Of these about 9000 were remote console jobs submitted by way of the newly introduced PROCSY system. By February of 1970, the last month for which statistics are available at the time this is being written, the total number of jobs was over 60,000 of which about 25,000 were PROCSY jobs. The reverse process of staging input or rollout files in ECS is then moved from ECS to a disc storage file at the same rate as it could have been moved directly from central memory to disc storage. Since the transfer rate to disc storage is about 62000 words per second, the use of ECS to buffer the rollout process makes the field length of central memory that is being freed available from several hundred milliseconds to several seconds earlier than in the unbuffered system.

The console system is now almost exclusively a remote job entry system. During the next few months we expect a very large volume of interactive computing to be added as the new interactive text editor and a new interactive algebraic language processor come into full production. Some hardware and system software

From the collection of the Computer History Museum (www.computerhistory.org)
changes are being made to accommodate this increased load, but it will be essentially the same system with the same scheduling and job movement mechanisms.

On a typical busy day there are now in excess of 10,000 rollouts. Short jobs, whether entered through card readers or through typewriter consoles get very good turnaround. Longer jobs are mostly relegated to the rollout queues during the main shift in which input activity is very heavy. Most of them are completed during the late night shift when the console system is turned off.

There is of course a very substantial amount of system overhead associated with rolling out and rolling back in over 10,000 jobs per day. This overhead does not seem to be too high a price to pay for the ability to handle interactive jobs, and the ability to implement scheduling strategies like those that give fast turn-around to short debugging runs.

In addition, statistics gathered before and after the introduction of the autoroll system show that the Purdue MACE system is more efficient in its CPU utilization and in its central memory utilization than its predecessor systems.

REFERENCES

1 E G COFFMAN JR  L KLEINROCK
   Computer scheduling methods and countermeasures
   AFIPS Conference Proceedings Vol 32 Spring 1968 p 11-21
2 Control Data Mace operating system preliminary reference manual
   Control Data Corporation Publication No 44613900
3 Control Data Chippewa operating system reference manual
   Control Data Corporation Publication No 60134400
4 Control Data 6400/6500/6600 computer systems reference manual
   Control Data Corporation Publication No 60100000
5 P J DENNING
   The working set model for program behavior
   Comm of the ACM 11 5 May 1968 p 323-333
6 P J DENNING
   Virtual memory
   Technical Report Number 81 Computer Science Laboratory
   Princeton University January 1970