# Computer scheduling methods and their countermeasures

*by* EDWARD G. COFFMAN, JR*
*Princeton University*
Princeton, New Jersey
and
LEONARD KLEINROCK**
*University of California*
Los Angeles, California

## INTRODUCTION

The simultaneous demand for computer service by members from a population of users generally results in the formation of queues. These queues are controlled by some computer scheduling method which chooses the order in which various users receive attention. The goal of this priority scheduling algorithm is to provide the population of users with a high grade of service (rapid response, resource availability, etc.(, at the same time maintaining an acceptable throughput rate. The object of the present paper is to discuss most of the priority scheduling procedures that have been considered in the past few years, to discuss in a coherent way their effectiveness and weaknesses in terms of the performance measures mentioned above, to describe what the analysis of related queueing models has been able to provide in the way of design aids, and in this last respect, to point out certain unsolved problems. In addition we discuss the countermeasures which a customer might use in an attempt to defeat the scheduling algorithm by arranging his requests in such a way that he appears as a high priority user. To the extent that we can carry out such an undertaking, the single most important value of this consolidation of the results of analysis, experimentation, and experience will be in the potential reduction of the uncertainty connected with the design of a workable service discipline.

By a grade or class of service we mean the availability of certain resources (both software and hardware), a distribution of resource usage costs, and a well-defined distribution of waiting or turn-around times which applies to the customer's use of these resources. In multi-access, multiprogramming systems throughput may conveniently be measured in terms of computer operating efficiency defined roughly as the percentage of time the computer spends in performing user or customer-directed tasks as compared with the time spent in performing executive (overhead type) tasks. We shall avoid trying to measure the programmers' or users' productivity in a multi-access environment as compared with productivity in the usually less flexible but more efficient batch-processing environment. For discussions on this subject see References 1 and 2 and the bibliography of Reference 3.

With a somewhat different orientation some of these topics have been covered elsewhere. In particular, Coffman,[4] Greenberger[5] and in more detail Estrin and Kleinrock[6] have reviewed the many applications of queueing theory to the analysis of multiprogramming systems. In addition, Estrin and Kleinrock have surveyed simulation and empirical studies of such systems. However, in contrast to the purposes of the present paper, the work cited above concentrates on mathematical models and on service disciplines to which mathematical modeling and analysis have been to some extent successfully applied. We shall extend this investigation to several priority disciplines not yet analyzed and to others which more properly apply to batch-processing environments. Furthermore, as indicated earlier, the present treatment investigates on a qualitative basis the detailed interaction of the customer and the overall system with the service discipline.

### *Classification of priority disciplines*

Before classifying priority disciplines consider the following very general notion of a queueing system. In Figure 1 we have shown a feedback queueing system consisting of a computer (service) facility, a queue or system of queues of unprocessed or incompletely

*This research was supported in part by the Bell Telephone Laboratories, Murray Hill, New Jersey.

**This research was supported in part, under Contract DAABO7-67-0540, United States Army Electronics Command, and also in part by the Advanced Research Projects Agency (SD-184).

processed jobs (or more generally, requests for service), a source of arrivals requesting service and a feedback path from the computer to the system of queues for partially processed jobs. The system will be defined in any given instance by a description of the arrival mechanism, the service required from the computer, the nature of the computer facility, the service discipline according to which the selection of service requests from the system of users is determined, and the conditions under which jobs are "fed back" to the system of queues. In all of the service disciplines discussed in the next section we make the following assumptions: 1) the arrival mechanism is such that if the arrival source is not empty it generates new requests according to some probability distribution, 2) the service disciplines are such that the computer facility will never be idle if there exists a job in the system ready to be executed.

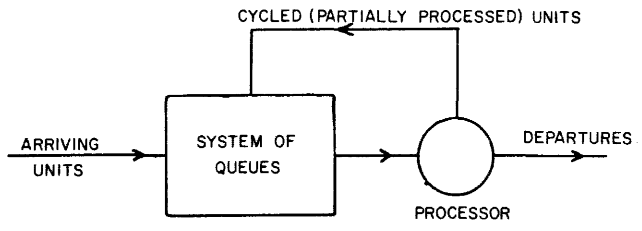

Figure 1 – Feedback queueing system

There are a variety of ways to classify priority service disciplines. Indeed, one point of view is expressed by saying that priorities may be *bought* (e.g., in disciplines where bribing is allowed,[7] *earned* e.g., by a program demonstrating favorable characteristics in a time-sharing system), *deserved* (e.g., by a program exhibiting beforehand favorable characteristics), or a combination of the above. For our purposes we shall classify a given priority method according to the properties listed below.

### A. Preemptive vs. non-preemptive disciplines

This characteristic generally determines how new arrivals are processed according to the given discipline. If a low priority unit is being serviced when a higher priority unit arrives (or comes into existence by virtue of a priority change of some unit already in the system) then a preemptive service discipline *immediately* interrupts the server, returns the lower priority unit to the queue (or simply ejects it from the system), and commences service on the higher priority unit. Note that non-preemptive disciplines involve preplanning in some sense. However, the extent of pre-planned "schedules" may vary widely.

### B. Resume vs. restart

This characteristic determines how service is to proceed on a previously interrupted (preempted) job when it comes up for service again. With a *resume* priority rule no service is lost due to interruption and with a *restart* rule all service is lost. Assuming that the costs of lost service are intolerable in the applications of concern to us we shall treat only resume rules and systems in which such rules are feasible.

### C. Source of priority information

Service disciplines may be classified according to the information on which they base priority decisions. Such a list would be open-ended; however, the sources of the information may be considered to fall in one of three not necessarily disjoint environments: 1) the job environment whereby the information consists of the intrinsic properties of the jobs (e.g., running time, input/output requirements, storage requirements, etc..), 2) the (virtual) computer system environment (e.g., dynamic priorities may be based on the state of the system as embodied in the number of jobs or requests waiting, storage space available, location of jobs waiting, etc.) and 3) the programmers' or users' environment in which management may assign priorities according to urgency, importance of individual programmers, etc.

### D. Time at which information becomes known

Classical service disciplines assume that the information on which priority decisions are based is known beforehand. On the other hand, time-sharing disciplines are a prime example of service disciplines in which decisions are based on information (e.g., running time and paging behavior, which is obtained only during the processing of service requests. Such information, of course, is used to establish priorities based on the predicted service requirements of requests which at some time were interrupted and returned to the queue.

All of the priority scheduling methods to be discussed are applicable to the infinite input population case (in which the number of possible customers is unbounded) as well as to the finite population case (in which a finite number of customers use the system) – see Reference 6.

### *Priorities based only on running times*

The intent of systems using a so-called running-time priority discipline is that the shorter jobs should enjoy better service in terms of waiting times. The FCFS (first-come-first-served) system is commonly used as a standard of reference to evaluate the success of this intent. The first two algorithms below assume job running times are *known at the time they arrive* at the service point, while most of the remainder, which have arisen primarily in connection with multiprogramming systems, assume that no indications of running times are known until after jobs have been at least partially run.

### A. The shortest-job-first (SJF) discipline[8]

This is a non-preemptive priority rule whereby the queue is inspected only after jobs are completely processed (served) at which times that job in the queue requiring the least running time is the next to receive service to completion (and thus there are no cycled arrivals). The SJF descipline has the obvious advantage of simplicity and the somewhat less obvious advantage that the mean customer waiting time in the system is less than in any system (including the FCFS system) not taking advantage of known running times. However, it is clear that significantly long running jobs suffer more in an SJF system than in an FCFS system. Thus, the reduction in the first moment of the waiting time comes at the cost of an increase in the second moment (or variance). We discuss the SJF discipline further in connection with the following discipline.

### B. The preemptive shortest-job first (PSJF) discipline

With this discipline the SJF priority rule is applied whenever a job is completed as well as whenever there is a new arrival. If a new arrival has, at its time of arrival, a service requirement less than the *remaining* running time required by the job, if any, in service, then the latter job is cycled back to the single queue and the computer given over to the new arrival. The job returned to the queue is subsequently treated as if its running time were that which remained when it was interrupted; i.e., we have a preemptive, resume discipline.

The PSJF discipline has the advantages over the SJF and FCFS disciplines of further accentuating the favoritism enjoyed by the short running jobs and further reducing the average waiting time in the system. (Again, the variance will be increased.) Indeed, it has been shown that the PSJF discipline is the optimum running-time priority discipline in these last two respects. This relationship between the SJF and PSJF disciplines is seen in part by observing that time-of-arrival receives some consideration in the SJF discipline but, because of the preemption property, none at all in the PSJF discipline.

The principal disadvantage of the PSJF discipline in the computer application is the cost associated with interrupting a job in progress, placing it into auxiliary (queue) storage, and loading the higher priority job for execution. Although this swapping process with auxiliary storage devices may not always be necessary, depending on the size of main storage, it may outweigh the advantage of PSJF scheduling over SJF scheduling. Since it is usually difficult to expect advance knowledge of exact running times, it is encouraging to note that in a thorough study by Miller and Schrage[9] it is shown that even with partial indications of running times significant improvements in mean flow time are possible at the expense of increases in the second moment.

It is obvious that the major effect of these disciplines on the programmer-users of such systems is a salubrious one in that it causes them to produce faster, more efficient jobs. However, the reaction of a user with a job ready to be submitted to an SJF or PSJF system depends to some extent on what information regarding the state of the system is available to the user. If the user can see only the queue length (and this is usually available) then whether or not he balks (refuses to join the queue) depends on how long his job is, his knowledge of the distribution of the running times of jobs submitted to the system, and his assessment of his chances were he to decide to come back later. If indications of the running times of jobs in the queue are known at arrival time then good estimates of waiting time are possible; thus, longer jobs wanting fast service are more likely to balk. With knowledge only of queue length, however, it would appear that less balking would occur with the SJF and PSJF systems than with the FCFS system. On the other hand, reneging (leaving the queue after joining it and before being completely serviced) would be more likely since long jobs are likely to progress rather slowly toward the service point.

The countermeasures available to the users of the SJF system in attempts to defeat (or take advantage of) the computer scheduling algorithm are rather obvious. Firstly, it is clearly advantageous to submit as *short* a job as possible. The natural consequence of this action suggests that a user partition his request into a sequence of short independent requests. Secondly, unless special precautions and penalties are provided, the users may purposefully lie about (i.e., underestimate) their required running time. Such countermeasures lead to a situation in which the attempted discrimination among jobs becomes ineffective and preferred treatment is given to those users displaying the use of clever and/or unethical tactics. We will continue to observe this unfortunate result in the other scheduling algorithms.

So far we have been discussing computer operating disciplines as if there existed but one queue of jobs waiting in the central processor. This is not generally true if we consider also the queue or queues of jobs waiting for the use of input/output (I/O) devices. The executive or supervisor system itself may be one of the "jobs" in the I/O queues. In a general batch-processing or time-sharing system it is more realistic to assume that jobs consist of phases or tasks whose requirements alternate between the use of the central processor and the use of some I/O device. Then the SJF policies may be defined just for the central processing time (as implied above) or by the sum of central processing and I/O time.

It is not our intention to discuss I/O scheduling disciplines in any detail but it should be kept in mind that a job completion in the central processor system may simply mean that the given job has reached a point where it requires an I/O process before continuing. Similarly, an arrival may mean a job returning from the I/O system for more computing time. This is not to say, however, that I/O and central processing scheduling are independent processes; indeed, it may be necessary that one of the criteria for assigning priority (external or internal to the comput system) be which I/O devices are required and for how much time. This is taken up again below.

### C. The round-robin (RR) discipline

This well-known scheduling procedure was first introduced in time-sharing systems as a means for ensuring fast turn-around for short service requests when it is assumed that running times are not known in advance. As seen below the RR discipline falls within a class of so-called quantum-controlled service disciplines in which the size (q) of the quantum or basic time interval is to be considered as a design parameter. In an RR system the service facility (computer) processes each job or service request for a maximum of q seconds; if the job's service is completed during this quantum then it simply "leaves" the system (i.e., the waiting line and the central processor), otherwise the job is cycled back to the end of the single queue to await another quantum of service. New arrivals simply join the end of the queue.

As can be seen, the use of running time as a means of assigning priorities is *implicit* in the RR discipline, whereas it is *explicit* in the previous two disciplines. Running time priorities are assigned *after* a job has been allocated a quantum of service—if the job requires additional service it suffers an immediate drop to the lowest (relative) priority and sent to the end of the queue. Furthermore, it is clear that the RR policy uses both running time *and* time-of-arrival to make (implicit) priority decisions. This latter dependence is seen by noting that all jobs arriving at any time earlier than a given job will have been allocated at least one more quantum of service when the given job reaches the service point.

The extent to which the RR discipline maintains the shortest-job-first policy (in a posterior fashion) depends on the quantum size. Clearly, if q is allowed to be infinite we have a FCFS system. On the other hand as q approaches zero we have in the limit a so-called processor-sharing system[10] in which the part of the processor not devoted to executive or overhead functions is "divided up" equally among the jobs currently in the system. In short, we have a system with no waiting line wherein it is possible to execute all jobs simultaneously but at a rate reduction for an individual job which is proportional to the number in the system. (The computer systems with multiple program counters approximate to some extent the RR behavior with q very close to zero). Despite the better service for short jobs as q decreases, questions of efficiency generally dictate against quantum sizes too small in conventional computer systems. We elaborate on this shortly.

The extent of discrimination by the RR discipline in favor of short jobs also depends on the distribution of job running times. In particular, the RR discipline clearly does not take advantage of any knowledge gained by the quantum-execution of a job beyond the fact that the job simply requires more. For example, the distribution of job running times may be such that any job requiring more than two quanta will with probability .95 require in excess of 10 quanta. In this event the desire to favor shorter jobs would indicate that all jobs having received two quanta should not come to the service point for the third time until all jobs in the system have received at least two quanta. The distribution of job running times that *is* applicable to RR scheduling in this respect is the exponential distribution, which also corresponds to the assumption that has been found analytically tractable in most queueing theoretic studies of the RR discipline. This arises from the so-called memoryless property of the exponential distribution which means in our application that after executing a job for q seconds (with q arbitrary) the distribution of the remaining time to completion is always the same (and equal to the original distribution). Thus, it is the *continued identical uncertainty* in job running times that constitutes the primary job characteristic making the simple RR discipline desirable.

It is immediately evident from the definition of the RR discipline that the basic disadvantage consists of the *swapping* (removing one job from and placing another job into service) necessary for jobs requiring in excess of q seconds of service. Many approaches to the solution of this problem have been taken. For example, increasing the size of main memory so that many jobs may coexist there eliminates much of the need for swapping. Of course, this is a limited and possibly expensive solution. Also, overlapping the swapping of one job with the execution of another in systems with appropriate memory control and storage capacity makes the swapping process a latent one so that the suspension of the central processor for input/output processes is reduced. Nevertheless, with modern, large-scale multiprogramming systems, the swapping process remains a principal bottleneck to efficient operation with many users.

Several analytical studies of RR disciplines have been carried out[11,12,26] with the goal of determining, for a system defined by a given arrival process and job running time distribution, the interaction between system performance (efficiency, throughput, or waiting times) and the swap-time and quantum size parameters. As verified by experience, analysis has shown how performance deteriorates sharply when the quantum size for a given swap time and system loading is made lower than a certain minimum range of values (or alternatively when loading becomes too heavy for a given quantum size). The priority disciplines described in the next section illustrate techniques whereby this excessive deterioration of service is avoided to some extent.

As regards countermeasures, the mere reduction of one's job length gains little. However, if a user were to partition his job into many smaller jobs, then he would achieve superior performance than a user with an identical job which was left intact. Again, the clever user wins. An interesting property of the $q = 0$ case is worthy of note, namely, that all customers have *identical* ratios of service time to mean time spent in system!

**D. The multiple-level feedback (FB) discipline**

The FB discipline differs from the RR discipline in a way which is analogous to the way in which the PSJF rule differs from the SJF rule. In an FB system a new arrival preempts (following the quantum, if any, in progress) all jobs in the system and is allowed to operate until it has received at least as many quanta as that ~~job(s) which has received the least number~~ of quanta up to the time of the new arrival. Alternatively, the FB system may be viewed as consisting of multiple queue-levels number 1, 2, 3,... with new arrivals put in queue-level 1, jobs having received 1 quantum and requiring more in queue-level 2, etc. After each quantum-service the next job to be operated will be the one at the service point of the lowest numbered, non-empty queue-level.

Once again, shorter jobs receive better service at the expense of the longer jobs, and large jobs are not allowed to interfere or delay excessively the execution of small jobs. However, the mean flow time is the same in the RR and FB systems. (In this regard, we note the existence of a conservation law[13] which gives the contraint* under which one may trade the speed of response among a population of users.) The choice between the RR and FB priority disciplines is determined basically by how much one wants to favor short jobs, for the basic algorithms involve the same amount of swapping. It is true, however that the FB discipline is somewhat more costly to implement in the sense that indicators must be used to keep track of the amount of service received by each job.

In the FB system, the users' countermeasure is again to partition his work into many smaller jobs each requiring a small number of quanta (one quantum each, optimally).

Observe that the RR, FB, and FCFS disciplines may be combined in a variety of useful ways. Two combinations that have been used are described below.

**E. The two-level FB or limited RR discipline**

With this discipline jobs are permitted to "round-robin" only until they have received a fixed number of quanta. At this point they are put into a "background" which is only serviced when there are no other jobs in the system. The background queue may be executed in a FCFS fashion or in a RR fashion with perhaps a larger quantum size. Here, the user countermeasures by forming many jobs from one, each such requiring no more than the fixed number of quanta which prevent his falling into the background queue.

**F. The FB discipline with a finite number of levels**

In this system a job after receiving a fixed maximum of quanta according to the FB rule (case D) is made to join a background which is to be serviced in one of the ways mentioned above. A further degree of freedom may be added to this or the simple FB rule by removing the constraint that the quanta allocated at different queue-levels be the same.

A question that immediately arises is how one goes about establishing the values of the quanta, the number of levels, or any of the other parameters of these running time priority disciplines. With a specified arrival process and job running time distribution, analysis has been only partly successful in the attempt

*In particular, the sum of the products for each class of users of the utilization factor (mean arrival rate times mean service time) and the mean waiting time remains constant.

to relate the performance measures of interest to the structural parameters of the system.[12] For further results simulation or empirical study[25] will, in many cases, prove the more rewarding approach.

Observe that if q is made zero in the basic FB system we have a processor-sharing type of system in which the jobs sharing the processor at any point in time are those which have received the least amount of service. Thus, a new arrival immediately preempts those jobs sharing the processor and is allowed to operate until its attained service is equal to that of the former or until it completes. It can be seen that the RR and FB processor-sharing disciplines differ in structure in precisely the same way as the SJF and PSJF disciplines. Furthermore, from our earlier remarks it is evident that the former priority rules represent the best that one can do with, respectively, non-preemptive and preemptive priority disciplines designed to favor shorter jobs when running time is not known in advance.

### G. Declaration of mode – interactive or batch

The time-sharing system at the University of California at Santa Barbara[14] uses an interesting variation of the above methods. A user is required to state whether his job is interactive (short, frequent requests) or batch (longer, usually single requests). Time is divided into fixed length segments such that during the first half of each segment, the interactive jobs are served in a round robin fashion until their half-segment is exhausted (or until their collective requests are satisfied). The remainder of the segment is then used to service (to completion, if possible) as many of the batch jobs as possible. During the first half-segment, some interactive jobs will drop from the queue after one quantum of service (e.g., those requiring the acceptance of a single button-push), etc; thus in this system, there is a benefit in declaring the nature of your job in an honest way, since in the case where there is only one batch job and many interactive jobs, the batch job receives better service if that user declares himself as being in the batch mode.

The discussion of balking and reneging when running time is assumed unknown applies without change to the RR and FB disciplines analogous to the SJF and PSJF disciplines, respectively. Another effect that may need to be taken into account in the design of a time-sharing discipline is that of "jockeying" among queues of users awaiting service at a console. (This complication of balking and reneging has received little attention in the literature.) As before, users are encouraged to produce fast running jobs for the RR and FB disciplines. In time-sharing applications this might be better stated by saying that users are encouraged to produce "frequently interacting" jobs. In a given multiple-level RR system, for example, long jobs may avoid being put into a background by communicating (artificially, if necessary) with the on-line user at a frequency such that its running time never exceeds the foreground quantum during any operation interval. In effect, the job is alternating between the foreground and input/output queues in a way that provides better service than if it were put into the background. Obviously, this is an example where the more clever users of a system tend to defeat the purpose of the service discipline.

As a final remark it should be noted that the existence of saturation in a system with any of the disciplines we have discussed, except for the RR discipline, depends on the class of jobs being considered. For example, in an SJF, PSJF, or FB system it is clearly possible that loading be such that jobs with less than say five minutes running time will have a finite expected wait while those with greater than five minutes running time will have an infinite expected wait (in the infinite population case). Of course, the system *as a whole* is saturated if a finite threshold of this nature exists, since the processor is not able to complete all jobs submitted to it. For the RR discipline (as with the FCFS system) there exists a single saturation point which, when reached, causes all jobs to have an infinite expected wait. (This stems from the continued interference of long jobs with the execution of short jobs.)

*State-dependent running time priority disciplines*

The principal motivation behind state-dependent disciplines is the desire to reduce the overhead and swapping costs in the execution of quantum-controlled service. It is of particular interest to prevent or minimize the collapse of RR system performance under heavy loading; i.e., to provide a more graceful deterioration of service with increases in loading.

### A. Cycle-oriented RR disciplines

A basic design parameter of such systems is the so-called cycle or response time which is set and used to control as desired the maximum amount of time required to execute one round-robin through all active jobs. In one variation,[12] after completing a given cycle or round-robin, the subsequent round-robin quantum is determined by dividing the fixed cycle time parameter by the number of jobs requiring service; the time represented by the result of this division is then allocated to each of the jobs requiring service at the beginning of the cycle. Subsequent cycles are then determined in the same fashion. Usually, some minimum allocation time (quantum) is always given because of the otherwise seriously degrading effects of swapping during heavy loading. Although the cycle time therefore increases at excessive loading, it is clear that system performance degrades more gracefully than otherwise in that the loss of swapping is reduced as the load increases. The present discipline is state-dependent in the sense that the amount of time received by a given job in a given cycle depends on the number of active jobs in the system at the beginning of the cycle.

In another (two-level) RR variation[26] a maximum cycle time is similarly imposed on the amount of time taken to process a "foreground" queue (i.e., a queue of interactive, on-line user jobs) and a background queue consisting of conventional production type jobs. In this time interval each foreground job is given a fixed quantum; if there is any time remaining in the cycle it is devoted to the background job processing, otherwise another cycle is initiated. To limit swapping overhead (and thereby provide graceful degradation during periods of heavy loading) the cycle is extended in the event there are too many foreground jobs to process for one quantum in one cycle.

It is clear that the advantage of a reduced variance in RR response times is compensated in cycle-oriented disciplines by the slower reaction to changes in loading or input activity (we have implied that the queue is examined only at the end of the round-robin cycles). Statistically, however, this disadvantage would seem to be a minor one. Of course, it is also possible to make the value of the cycle time parameter dependent on system loading (number in the system). Just how this is to be done in a useful way, however, presents a difficult problem. In these disciplines, the countermeasure of partitioning a long job into many small jobs is extremely effective.

### B. Input-dependent disciplines

In one such (RR) discipline each time a new arrival occurs the job, if any, in service is allocated an additional quantum of execution time.[15] In this way the RR discipline reacts to heavy input activity by increasing time allocation and thereby reducing the amount of overhead and swapping. During light to moderate input activity the straight RR discipline is little effected by this change.

Another such discipline orders the queue of interactive user's jobs by interarrival time. Thus, those users communicating with the system at the faster rates will receive the shorter response time. The obvious countermeasure here is to initiate false I/O commands. This technique, of course, may be combined in various ways with both RR and FB disciplines.

### C. Priorities based on storage allocation

Apart from the number in the system the most important other source of internal priority information is the current allocation of storage and the availability of I/O devices to perform storage allocation functions. This, of course is tied in with the swapping problem discussed earlier. In batch processing systems requiring maximum efficiency this information may serve as the only criterion for assigning priorities; at a decision point with this type of discipline a schedule of job operations is computed as far as necessary in advance such that storage is well utilized in some reasonable sense.

This sort of scheduling is also applicable to the cycle-oriented RR disciplines described earlier. Thus a job is executed once per cycle, but where it executes in the cycle is made dependent on what turns out to be the best way (or at least a good way) to sequence the use of main storage so that I/O is minimized and overlapped as much as possible with computation. Clearly, the cost that must be compensated in this operation is the (potentially substantial) overhead time required to produce "good" schedules.

In the latest generation of multiprogramming systems, storage structure and the processes of storage allocation have been made more elegant by the concepts of virtual memory and paging.[16] In paging systems jobs (programs and data) are paginated into sets of fixed length pages or blocks of computer words so that the logical unit of information transfer within the supervisory system becomes a page. This also means that jobs may be operated (at least in part) when only a proper subset of the job's pages are in main storage. The synthesis and analysis of efficient storage dependent, running time priority disciplines that take advantage of this added flexibility is a difficult, important, and as yet unsolved problem.

## *Inclusion of externally generated priorities*

The classical priority disciplines in which priorities are determined external to the computer system are described as follows. At any point in time with preemptive rules or just after service completions with non-preeemptive rules the job next to be serviced is the one with the highest priority (i.e., the job having been assigned the lowest priority number). That discipline receiving the most attention in the literature is one for which the number of priorities is finite or countably infinite; i.e., in one-to-one correspondence with the integers.[17] This gives rise to levels of queues containing jobs of the same priority; these queues are generally ordered by time of arrival to the system. The advantages and disadvantages of preemptive vs. non-preemptive priority rules are the same as those discussed for the PSJF and SJF rules in an earlier section.

There has been a variety of ways by which externally generated priorities have been included in the running time priority disciplines described in previous sections. These are classified as follows.

### A. RR disciplines with external priorities

A technique used with RR disciplines consists of making the quantum size to be allocated dependent on the priorities of the active jobs.[10] Thus, a higher priority job would be allocated a larger quantum than a lower priority job. In the limit where the quantum size is zero we have a processor-sharing system in which the fraction of the processing rate received by a job is determined by an externally assigned priority.

Another technique which smacks of the multiple level schemes given below involves the specification of (relative) time delays as priorities.[28] Consider the following implementation, for example: Each arriving job is assigned a (priority) number which is based on the given job's externally assigned priority *and* on the number currently possessed by the other active jobs in the system. After a given quantum-service (which may consist of multiple quanta) the next job to receive service is the one having the lowest priority number. The jobs having the same priority number are ordered by time of arrival and serviced in that sequence. Each time a job is serviced for one quantum its priority number is increased by one. Thus, with a non-preemptive system we see that the number assigned to an arriving job indicates how much time it is to be allocated for operation before it joins the round-robin of jobs already in the system; this in turn is determined by an external priority assignment.

### B. Multiple level disciplines with externally assigned priorities

The simplest and most natural method of including external priorities applies to the multiple level disciplines in which each level is used to correspond to an external priority number as well as to a given level of attained service.[12] In this way conventional priority scheduling is combined directly with the various FB disciplines described earlier. However, variations in these types of disciplines may be obtained by the selection made for the means of ordering the queue-levels. Specifically, the queues may be ordered by time of arrival to the system, by time of arrival to the queue, or by a combination of one of these with ordering by priority; i.e., by the queue level of original entry to the system. (Note that ordering by time of arrival to the queue and by time of arrival to the system amounts to the same thing in the basic FB discipline without external priorities.)

One time-sharing scheduling discipline of the above type that has received considerable attention is one in which priorities are assigned at arrival time according to job size (storage requirements), queues are ordered by time of arrival to the queue, and quantum sizes are exponentially increasing with the queue level (i.e., level one provides on quantum, level two provides two quanta, level three provides four quanta, and so on.[2] Priorities based on storage requirements are assigned so that the larger jobs receive the lower priorities (enter at the higher queue levels). In this fashion efficiency is kept high by reduction in swapping time, since large jobs are given more time to operate between swaps. Furthermore, the large jobs are not allowed to interfere with the small, presumably faster, and more efficiently scheduled jobs. Clearly, users of such systems are further encouraged to write small jobs, again possibly by breaking larger jobs up into autonomous and small sub-jobs.

For a given application the design of the general disciplines just discussed requires a means for evaluation of the best values for the quantum distribution, the specification of the storage-dependent priority assignment rule, and the selection of the best method for ordering queue levels. Here again is a synthesis problem similar to the one mentioned earlier. At present no generally applicable, well-defined procedure exists; experiment by simulation and empirical study has been used thus far. Some encouraging work towards the optimal synthesis of such systems has been reported by Fife.[18]

In all of the externally assigned priority methods, the user who can influence the assignment of external priorities has a great advantage over the others. This is taken up next.

### C. User controlled priority assignment

According to this type of discipline the user is allowed in some way to bid for, or simply buy the priority he desires (or can afford) for his job. One such discipline is the so-called bribing model[7] in which system users offer a bribe (based on an "impatience" factor of their own) to obtain a preferred position in the queue of waiting jobs. All those bribing strategies are then considered which minimize an appropriately defined cost function over the set of users.

Another (quantum-controlled) system[19] of this sort has been used which provides a quantum size that is proportional to the priority a user decides to assign his job, and which increases with the size of his job (in order to maintain a reasonable operating efficiency). What constrains the priority a user assigns to his job is the fact that he is charged a fee for use of the system which is proportional to the product of the priority he has selected times the sum of the computing and I/O time required by the job. In this particular case, as well as in the bribing case, if a user is capable of learning what all the other users have assigned as their priorities (or bribes) then he need merely "go one better" and choose a slightly higher priority (or bribe) to achieve superior service. Note when the system is heavily loaded that all users see a "slow" system and so they tend to increase their self-assigned priorities (or bribes) in an iterative fashion; the result is an ever increasing cost to the user for a constantly decreasing grade of service! Clearly, the user population as a whole should in such case, act in collusion to prevent such runaway conditions.

*Dynamic or time-dependent priorities*

There have been a number of priority disciplines proposed in which jobs receive an external priority that changes in a dynamic way once the job is in the system. These disciplines were motivated by other applications but it will be clear that they may be considered candidates for scheduling computer operations.

So far we have treated disciplines in which the waiting time experienced by a job is not used to directly influence the priority decision. The disciplines below are structured so that this information, weighted by an external priority number is used in the process of selecting from the queue which job is to be serviced next.

**A. Delay-dependent disciplines**

In the first variation to be discussed[20] a job's priority is increased, from zero at the time of arrival, linearly with time in proportion to a rate (externally) assigned to the job's priority class. Each time a new job is to be selected for service according to the non-preemptive or preemptive variations of this scheme the "attained" priorities of jobs in the system are compared, and that job with the highest attained priority is selected next for service. If the priority classes are assigned according to a shortest-job-first policy it can be seen that this rule moderates the SJF and PSJF rules by reducing the probability of excessively long waits.

In another variation[21] jobs are similarly assigned external priorities related to the urgency of service. However, with this discipline a given job takes precedence over another job in the queue if, and only if, the difference between the former's (external) priority and that of the latter is not less than the time the latter has spent waiting. This scheme may be implemented as a preemptive as well as a non-preeemptive discipline. Again, this is an example in which management becomes concerned about a job that has waited for a long time.

**B. Priorities based on general cost accrual[5]**

In the most general such system arriving jobs have associated with them a cost accrual rate which is some arbitrary function of time. In a preemptive or non-preemptive mode the discipline is executed by servicing (at each decision point) that job which minimizes over all jobs the cost accrued by the subsequent waiting time. The cost accrual attributable to a given job over a given time interval is calculated simply by integrating the cost rate curve over the given time interval. It is of interest also that one may introduce deadlines by making the slope of the cost rate function infinite after a suitable interval of time.

The simplest case of the general discipline exists when the cost functions are constant and identical for each user. It can be seen that this corresponds to a system in which cost accrues linearly with time and where the priority minimizes the average wait (i.e., we have the PSJF or SJF rules depending on whether or not we have preemption). If we assume constant valued functions that may differ for each job we have the so-called c/t rule. This rule amounts to selecting for service that job whose ratio of constant cost rate (c) to known or average service time (t) is the smallest.

In these systems, it is not usually to one's advantage to partition jobs into smaller ones in an attempt to defeat the system. In fact, it may pay to group many jobs together so that they all enjoy an early arrival time to the system.

*Priority disciplines in multiprocessor systems*

In providing the additional degree of freedom of the number (and perhaps types) of processors many different possibilities come into existence, most of which have not been fully tested or analyzed as yet. In this section we shall examine briefly the application of previously discussed disciplines to multiprocessor systems and certain disciplines for which analyses exist in the literature but which arose out of other applications.

**A. Processors-in-series systems**

Multiple channel (server) queueing systems are broadly classified in the literature according to whether the servers are being used in parallel or in phasé type service. By phase type service we mean that the processors are being used in sequence:[22] one phase of a given job are being serviced on a given processor. In general, each processor will have a queue of jobs (phases) which is fed by the output of some other processor or by an external source. Such a system applies, for example, to the alternating I/O and computing job structure described in another section. Here, we assume that at least one central processor and at least one I/O processor is being used in a cyclic way by a given job. Other applications of this discipline )which may be combined with other disciplines at each of the separate queues) are to be found in computer systems dedicated to the processing of certain, large phase-structure jobs.

### B. Processors used in parallel

The simplest extension of the previous disciplines to multiple processors is simply to treat the processors in a first-available-first-used fashion. [8] In large, general-purpose systems in which the processors are identical this single queue approach offers the advantages of simplicity, flexibility, and efficiency. The disciplines may be made more effective in those systems where jobs are designed to operate on one or more processors depending dynamically on availability.[23]

However, several other variations of "parallel" priority disciplines exist when the set of processors or the input jobs are not homogeneous. The simplest such case arises when jobs fall into one of two categories; a foreground of fast service jobs and a background of perhaps less important production type jobs. These conditions are appropriate to the so-called variable-channel discipline in which the number of processors made available to the foreground jobs is made dependent on (increases with) the number of foreground jobs in the system. The number of foreground processors increases only after a maximum queue length has been reached; i.e., with each newly arriving job after the fixed maximum has been reached a new processor is made available (if possible) to serve the job at the head of the queue.

Two other possibilities are 1) the existence of special purpose processors, and 2) processors of different computing speeds that may be allocated by external priority or by job requirements. In this regard, we may ask that a user estimate his job length or type and then assign him to a processor which has been optimized for such jobs. If, after some processing, it is found that a user has made a poor estimate, he is penalized in some way (e.g., by being forced to move to the end of a queue on another processor). His penalty for overestimating his work is to be placed on a processor which is not "tuned" to jobs of this type.[6]

The coming importance of networks of computers[29] creates another source of applications for the above types of multiple-queue disciplines. Computer network disciplines will also have to be dependent on transmission delays of service requests and jobs or parts of jobs from one computer to another as well as on the possible incompatibilities of various types between different computers. The synthesis and analysis of multiprocessor and multiple processor network priority disciplines remains a fertile area of research whose development awaits broader multiprocessor application and an enlightening experience with the characteristics of these applications.

## CONCLUSION

We have listed above a variety of possible computer scheduling methods suitable for many situations. This list is by no means complete. In fact, this wealth of possible algorithms produces an "embarrassment of riches" in that we do not really know how to select the most useful scheduling methods. As we have indicated, the possibilities are considerable.

We have also attempted to discuss briefly the possible counter-measures available to a user of the computer which would allow him to defeat or take advantage of the system for the various algorithms described. Indeed we have shown that in most cases, there is such a countermeasure! One hopes that there exists an efficient scheduling method which is immune to such manipulations.

## REFERENCES

1 J I SCHWARTZ E G COFFMAN C WEISSMAN
*A general purpose time-sharing system*
Proc SJCC 1964

2 F T CORBATO M MERWYN-DAGGETT R C DALEY
*An experimental time-sharing system*
Proc SJCC 1962

3 G BELL M W PIRTLE
*Time-sharing bibliography*
Proc IEEE December 1966

4 E G COFFMAN
*Studying multiprogramming systems through the use of queueing theory*
Datamation July 1967

5 M GREENBERGER
*The priority problem*
MIT Project Rep MAC-TR-22 November 1965

6 G ESTRIN L KLEINROCK
*Measures models and measurements for time-shared computer utilities*
Proc ACM Natl Conf August 1967

7 L KLEINROCK
*Optimum bribing for queue position*
Journal of operations research (to appear)

8 T E PHIPPS JR
*Machine repair as a priority waiting-line problem*
Operations Research vol 4 1956

9 L W MILLER L E SCHRAGE
*The queue M/G/1 with the shortest remaining processing time discipline*
Rand Corp Report P 3263 November 1965
See also
L E SCHRAGE
*Some queueing models for a time-shared facility*
PhD Dissertation Dept of Indust Engineering Cornell Univ February 1966

10 L KLEINROCK
*Time-shared systems: A theoretical treatment*
Journal of the ACM April 1967

11 L KLEINROCK
*Analysis of a time-shared processor*
Naval Res and Log Quart March 1964

12 E G COFFMAN
Stochastic models for multiple and time-shared computer systems
PhD Dissertation Dept of Engineering UCLA June 1966

13 L KLEINROCK
*A conservation law for a wide class of queueing disciplines*
Naval Res and Log Quart June 1965

14 G CULLER
Univ of Calif at Santa Barbara (Private Communication)

15 E G COFFMAN
*Analysis of two time-sharing algorithms designed for limited swapping*
Journal of the ACM (to appear)

16 J B DENNIS
*Segmentation and the design of multiprogrammed computer systems*
Journal of the ACM October 1965

17 A COBHAM
*Priority assignment in waiting line problems*
Operations Research Feb 1954

18 D W FIFE
*An optimization model for time-sharing*
Proc SJCC 1966

19 G SUTHERLAND
Paper presented at the symposium: *Computers and communication: Their system interaction*
Sponsored by the IEEE groups on Communication Technology and Electronic Computers Santa Monica Calif January 1967

20 L KLEINROCK A FINKELSTEIN
*Time-dependent priority queues*
Journal of ORSA (to appear)

21 J R JACKSON
*Waiting time distribution for queues with dynamic priorities*
Naval Res and Log Quart March 1962

22 L KLEINROCK
*Sequential processing machines (SPM) analyzed with a queueing theory model*
Journal of the ACM April 1966

23 D F MARTIN
*The automatic assignment and sequencing of computations on parallel processor systems*
PhD Dissertation Dept of Engineering UCLA January 1966

24 A L SCHERR
*An analysis of time-shared computer systems*
*PhD Dissertation Dept of Electrical Engineering MIT June 1965*
(See also the bibliography in reference 6)

25 B KRISHNAMOORTHI R C WOOD
*Time-shared computer operations with both interarrival and service times exponential*
Journal of the ACM July 1966

26 *Time-sharing system/360 development workbook*
IBM Internal Document

27 E T IRONS
*A rapid turn-around multi-programming system*
Comm of the ACM March 1965

28 E G COFFMAN
*Bounds on the parallel processing of bulk queues*
Naval Res and Log Quart September 1967

29 T MARILL L G ROBERTS
*Toward a cooperative network of time-shared computers*
Proc FJCC 1966