STAN-CS-73-355 SU-SEL-73-013

An Analysis of Central Processor Scheduling in Multiprogrammed Computer Systems

(Digest Edition)

by

Thomas G. Price

October 1972

Technical Report No. 57

This work was supported in part by the Joint
Services Electronics Program: U.S. Army,
U.S. Navy, and U.S. Air Force under Contract
N-00014-67-A-0112-0044 and by Mr. Price's
fellowship from the National Science Foundation.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY . STANFORD, CALIFORNIA

STAN-CS-73-355 SEL 73-013

AN ANALYSIS OF CENTRAL PROCESSOR SCHEDULING IN MULTIPROGRAMMED COMPUTER SYSTEMS

(Digest edition)

by

Thomas G. Price

October 1972

Technical Report No. 57

DIGITAL SYSTEMS LABORATORY

Dept. of Electrical Engineering Dept. of Computer Science

Stanford University

Stanford, California

This work was supported in part by the Joint Services Electronics Program: U.S. Army, U.S. Navy, and U. S. Air Force under contract N-00014-67-A-0112-0044 and by Mr. Price's fellowship from the National Science Foundation.

ABSTRACT

A simple finite source model is used to gain insight into the effect of central processor scheduling in multiprogrammed computer systems. CPU utilization is chosen as the measure of performance and this decision is discussed. A relation between CPU utilization and flow time is developed. It is shown that the shortest-remaining-processing-time discipline maximizes both CPU utilization and I/O utilization for the queueing model M/G/l/N. An exact analysis of processor utilization using shortest-remaining-processing-time scheduling for systems with two jobs is given and it is observed that the processor utilization is independent of the form of the processing time distribution. The effect of the CPU processing time distribution on performance is discussed. For first-come-first-served scheduling, it is shown that distributions with the same mean and variance can yield significantly different processor utilizations and that utilization may or may not significantly decrease with increasing variance. The results are used to compare several scheduling disciplines of practical interest. An approximate expression for CPU utilization using shortest-remaining-processing-time scheduling in systems with N jobs is given.

DISCUSSION

Scheduling in a multiprogramming system can be divided into three major areas. Size of the main memory limits the number of jobs which can be resident simultaneously. Since there will frequently be more jobs to execute than will fit in memory, the system must have a rule for selecting a set of jobs from the jobs waiting to be executed to reside in main memory. This rule will be referred to as the job selection rule or job scheduling. The job selection rule may try to optimize the job mix to give good turnaround time to short jobs, make efficient utilization of the hardware, etc.

The jobs that are currently resident in main storage will alternate between I/O and CPU operations. If the number of jobs multiprogrammed exceeds the number of CPUs, a waiting line or queue for the CPUs may develop. We will refer to the rule used to manage this queue as the CPU scheduling algorithm. Similarly, a queue may form in front of the I/O channels and devices and I/O scheduling must be considered. The purpose of this paper is to present an analysis of CPU scheduling algorithms.

Several authors have given heuristic approaches to CPU scheduling [Stevens, 1968], [Marshall, 1969], [Ryder, 1970], [Wulf, 1969], [HASP, 1969]. Most of the heuristics are based on the idea that some jobs are "compute bound" and other jobs are "I/O bound." The I/O bound jobs should be given priority for the CPU over CPU bound jobs in order to prevent "hogging" of the CPU by a compute bound job and hence poor CPU-I/O overlap. All of the above references describe implementations of heuristic scheduling algorithms on specific systems. Stevens [1968], Marshall [1969], Ryder [1970], and Wulf [1969] have all observed significant improvement in CPU utilization and throughput using their new scheduling algorithms. In [Stevens, 1968] and [Wulf, 1969] other parts of the system were changed at the same time the scheduling algorithm was changed, so it is difficult to determine exactly what the effect of changing the scheduling algorithm was. Ryder's [1970] heuristic is complicated, and it is difficult to determine which features are important. Finally, the relation between the job mix and the CPU scheduling algorithm is difficult to see because an adequate description of the system loads is not available. Ryder [1970] makes an effort to deal with this problem. Sherman [1971] has used a trace driven simulation model to systematically compare several CPU scheduling algorithms and has overcome most of the problems mentioned above.

Theoretical work on computer system scheduling has been performed by using a probability model to represent the computer system and then performing an analysis of particular scheduling algorithms for the chosen model. Probability models of computer systems can be divided into two major classes, finite source models and infinite source models [McKinney, 1969]. Most of the work on scheduling has used infinite source models. This is due primarily to the fact that infinite source models are easier to analyze than finite source models. Although infinite source models can give some insight into the operation of computer systems, they appear to be poor models of multiprogrammed computer systems since only a few jobs can fit in main storage simultaneously. Finite source models have been analyzed for first-come-first-served (FCFS) scheduling [Gaver, 1967], [Jaiswal, 1968], processor sharing scheduling [Baskett, 1971], and preemptive resume last-come-first-served (LCFS) scheduling [Chandy, 1972]. Baskett [1971] has noted that processor sharing can give considerable improvement over FCFS when the compute times are highly variable. Similar remarks apply to preemptive resume LCFS.

In this paper a simple finite source model is used to study the effect of the CPU scheduling algorithm on system performance. CPU utilization is used as the measure of system performance and the reason for this is discussed in section 2. The basic finite source model is shown in Fig 1. When a job arrives at the CPU the processing time for its next CPU operation, which will be called a "CPU time," is selected independently from a CPU processing time distribution which is identical for all jobs. At the completion of its CPU service, the job goes to an I/O processor (or channel). The service time at the I/O processor, which will be called an "I/O time," is selected from an I/O processing time distribution, again identical for all jobs. Each job is either doing I/O or using the CPU or waiting and no overlap is allowed.

Throughout this paper the assumptions necessary for each result are stated explicitly with the result. A few general comments about the assumptions will be given here. Most of the results in the paper assume that the CPU processing time distribution is general and the I/O processing time distribution is exponential. It is usually assumed that no queuing for I/O occurs. With these assumptions the model is equivalent to the M/G/l/N queueing process. This notation is due to Kendall and is described in [Jaiswal, 1968]. This model is sometimes referred to as the machine interference model [Cox, 1961]. The machine interference model has been suggested as a model of multiprogrammed computer systems [Gaver, 1967], [Baskett, 1971] and of time sharing systems [Scherr, 1967], [McKinney, 1969].

Many of these assumptions are not strictly true in real systems. I/O times are usually not exponentially distributed. Successive CPU

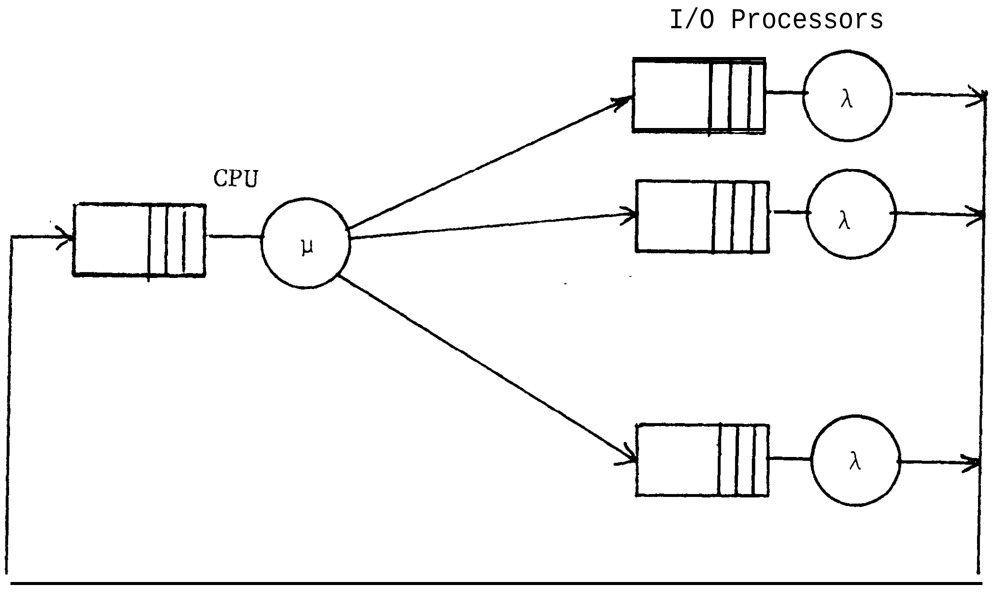

Figure 1. The Finite Source Model

times and I/O times may not be independent. Different jobs have different resource requirements. Many systems do allow for overlap of I/O and computing for a single job. Baskett [1972] has discussed some of the deficiencies of this type of model. Despite all of the problems, models of this type have been shown to be a fair approximation to real systems [Baskett, 1972]. Thus, while the model does not exactly reflect the structure and behavior of real systems, it retains many of the significant aspects of multiprogramming and is close enough to real systems to give us insight into the effect of scheduling.

REFERENCES

Baskett, F. and Palacios, F. G. Processor sharing in a central server queuing model of multiprogramming with applications. Proceedings of the Sixth Annual Princeton Conference on Information Science and Systems, Princeton University (March 1972).

Baskett, F. Mathematical models of multiprogrammed computer systems. TSN-17, Computation Center, University of Texas, Austin, Texas (January 1971).

Chandy, K. M. The analysis and solutions for general queuing networks. Proceedings of the Sixth Annual Princeton Conference on Information Sciences and Systems, Princeton University (March 1972).

Gaver, D. P. Probability models for multiprogramming computer systems. J. ACM 14, 3 (July 1967), 423-438.

HASP. Houston Automatic Spooling Priority System - II (Version 2), IBM Type III Program Documentation, Order No. 360D-05.1.014, IBM Corporation, Hawthorne, New York (1969).

Jaiswal, N. K. Priority Queues. Academic Press, New York, (1968).

Marshall, B. S. Dynamic calculation of dispatching priorities under OS/360 MVT. Datamation (August 1969), 93-97.

McKinney, J. M. A survey of analytical time-sharing models. Computing Surveys 1, 2 (June 1969), 105-116.

Ryder, K. D. A heuristic approach to task dispatching. IBM Systems Journal 8, 3 (1970), 189-198.

Scherr, A. L. An Analysis of Time-Shared Computer Systems. MIT Press, Cambridge, Massachusetts (1967).

Schrage, L. A proof of the optimality of the shortest remaining processing time discipline. Operations Research 16, 3 (May 1968), 687-690.

Sherman, S., Baskett, F., and Brown, J. C. Trace-driven modeling and analysis of CPU scheduling in a multi-programmed system, Proc. ACM SIGOPS Workshop on System Performance Evaluation, Harvard University (April 1971), 173-199.

Stevens, D. F. On overcoming high-priority paralysis in multiprogramming systems: A case history. C. ACM 11, 8 (August 1968), 539-541.

Wulf, W. A. Performance monitors for multi-programming systems. Proceedings of the Second Symposium on Operating Systems Principles (October, 1969), 175-185.

i