UCRL-17838 Rev.

cy. 2

University of California

Ernest O. Lawrence Radiation Laboratory

ON OVERCOMING HIGH-PRIORITY PARALYSIS IN MULTIPROGRAMMING SYSTEMS: A CASE HISTORY

David F. Stevens

September 21, 1967
(Revised January 24, 1968)

TWO-WEEK LOAN COPY

This is a Library Circulating Copy which may be borrowed for two weeks. For a personal retention copy, call Tech. Info. Division, Ext. 5545

Berkeley, California

UCRL-17838 Rev. cy. 2

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

Submitted as Communications of the
Association for Computer Machinery

UCRL-17838 Rev.
Preprint

UNIVERSITY OF CALIFORNIA
Lawrence Radiation Laboratory
Berkeley, California

AEC Contract No. W-7405-eng-48

ON OVERCOMING HIGH-PRIORITY PARALYSIS IN MULTIPROGRAMMING SYSTEMS: A CASE HISTORY

David F. Stevens

September 21, 1967
(Revised January 24, 1968)

ON OVERCOMING HIGH-PRIORITY PARALYSIS IN MULTIPROGRAMMING SYSTEMS: A CASE HISTORY

David F. Stevens

Lawrence Radiation Laboratory
University of California
Berkeley, California

September 21, 1967
(Revised January 24, 1968)

CR categories: 4.30, 4.32, 4.39
Key words and phrases: multi-program scheduling; dynamic priority assignment; scheduling

Abstract

High-priority paralysis is the degradation that can occur in multiprogramming systems when scheduling is based primarily on preassigned priorities. It can be alleviated by modifying the scheduling algorithm to maximize the number of programs active at one time. The case history given here indicates two general methods by which simultaneity can be increased. Possible refinements in the scheduling algorithm for future improvements are considered briefly.

The work described in this note was supported by the United States Atomic Energy Commission.

Introduction. Since this is a case history, much of the discussion must be couched in terms peculiar to the particular computer and operating system involved. Nevertheless, the problem is, potentially at least, a general one, and the steps taken to solve it, and the philosophy underlying them, have general validity.

The body of the note consists of four sections: the patient, the symptoms, the treatment, and evaluation. The first section presents a sketch of the affected system in enough detail to provide an adequate background for the next two sections, which describe the symptoms and the treatment for this particular case of high-priority paralysis. The fourth section offers some evidence of the success of the treatment to date, points out its limitations, and indicates some possible extensions.

The patient: a brief description of the afflicted system. The system under consideration was the Chippewa Operating System on the Control Data 6600 computer. The 6600 is an eleven-computer complex, consisting of a very large,* very fast central computer plus ten small, moderately fast peripheral computers. The central computer has no I/O commands and has no direct communication with any I/O channel or device; the peripheral computers (PPUs), on the other hand, all have access to all I/O channels and devices and also have (read and write) access to the memory of the central computer.

Under the Chippewa system [1,2] one PPU is given the task of driving the operator's console, another is given control of the system via the chief executive program (the monitor), and the remaining eight PPUs, called "pool PPUs," are available for assignment as needed. The central computer and all of the pool PPUs function as slaves to the monitor.

* Although some 6600s have been delivered with only 32K of central memory, the installation described here had 131K.

Communication between the monitor and the pool PPUs is accomplished through PPU communication areas in the memory of the central computer. Other central memory communication areas, called "control points," are used by the monitor to govern the allocation of system resources (central memory, the central processor, I/O devices, etc.) to active programs.

Only one program can be assigned to a control point at a time, and a program cannot be "active" in any sense, or own, lease, or occupy any of central memory, unless it is assigned to a control point. The control point also provides space for storing the contents of the operating registers when the central processor is assigned to a program at a different control point.

There are seven of these control points available; in practice, two are normally used for system input and output (the formation of the job input queue and the printing of jobs in the output queue). (The two system I/O routines use central memory space for buffers but never require the central processor.) The other control points are handled in a straightforward multiprogramming fashion with a simple-minded scheduling algorithm: first come, first served, with conflicts resolved on the basis of a priority preassigned by the programmer. In particular, when a control point becomes free, it is given to the highest-priority job in the input queue; similarly, the central processor is assigned to the highest-priority job which requests it, and is released only upon initiation of an I/O request or the demand of a program with higher priority.

<u>The symptoms</u>. The symptoms by which the high-priority paralysis was recognized are as straightforward as the system itself:

(1) The occupation of one or more control points by high-priority

jobs requiring more central memory than was currently available. The high-priority jobs were unable to execute because they could get no core, and the smaller, low-priority jobs in the input queue were unable to execute because they could not reach a control point.

(2) The hogging of the central processor by a high-priority, compute-bound program when other control points contain other, more I/O-limited, programs waiting to execute.

The result of either of these occurrences was paralysis of one or more control points, at times reducing Chippewa from a four- or five-way multiprogramming system to a sausage-style system with spooling.

The treatment: its philosophy and implementation. The principle behind the treatment adopted was: Maximize the number of programs which are truly active at the same time. ("Truly active" means that the business of the program is being advanced in some manner: programs waiting for storage or for the central processor are not "truly active"; programs waiting for the completion of some I/O operation are.)

The most obvious implementation of this principle, at least with respect to the first of the two forms of paralysis mentioned above, is to increase the number of control points. Unfortunately, the number seven, in the form of three-bit fields and masks, is so deeply imbedded in the system that this course was deemed infeasible. The alternative selected is to give a lower priorty job precedence if it will fit into available central memory. Thus a free control point is assigned the highest priority job that will fit in the memory available; if no job in the input queue will fit, the control point remains idle. This has the disadvantage of

imposing overnight turnaround on very large jobs, but since most such jobs are production runs, such turnaround is acceptable. Other relief, in the form of a priority incrementer and an operator-assignable "must-load" priority, has been provided for the exceptional cases.

To attack the second form of the paralysis it is necessary to maximize the number of jobs which are doing I/O simultaneously. In general, this could best be accomplished by assigning the central processor to that job which would relinquish it soonest (after initiating an I/O request), if there were some way to decide which job that was. A brief attempt was made to have the operators assess the situation and manipulate priorities, but their success was haphazard, and their response to other demands seriously impaired. As a result, an automatic, dynamic priority assignment routine, which periodically recalculates the priority of each executing job as a function of the ratio of compute (central processor) time to I/O (peripheral processor) time charged to the job, was incorporated into the program driving the operator's console. (These times are maintained to near-millisecond accuracy in the control point communication areas by the monitor as a matter of course.) Since peripheral processor time is roughly proportional to I/O time, this procedure tends to assign the highest priority -- and hence, when requested, the central processor -- to the most highly I/O-bound job.

Evaluation: there always is a bottleneck. Rigorous evaluation of the effects of the system changes described above is difficult to achieve, for the development of adequate evaluation techniques came after the installation of the changes. Thus, though we can describe the current state of affairs with some accuracy, our estimate of the original state is necessarily

approximate. Even rough estimates can be of some value, however: they can be found in Table 1. The figures given apply to daytime periods when the system tends to be saturated with short runs. At night, when large, long production jobs predominate, the ranges would be uniformly narrower, and show an increase in central processor activity and a decrease in all other categories. Figures for the original system were obtained by observation with watch in hand and by examining the dayfile, a time-sequenced listing of significant events for all jobs passed through the system; figures for the modified system were obtained from dayfile examination and some PPU-resident monitoring routines described elsewhere [3].

	Original System	Modified System
Memory utilization	70-75%	90%
Jobs(1) with memory assigned	3	4
I/O channels(2) active	1.0	1.3
CPU utilization	50%	68%
Jobs(1)/hour through system	30	60

TABLE 1 (all numbers are mean values)

(1) Excluding the system I/O jobs

(2) Excluding the channel dedicated to the operator's console

It is clear that the modified system gets more out of the computer than the original system, but it also appears that we have traded one bottleneck (control points) for another (memory space). Since the likelihood of fitting additional jobs into core is small, any further improvement must come in the area of increasing simultaneous activity. Three

approaches seem feasible:

(1) Modify the scheduling algorithm to provide a more compatible set of active jobs;

(2) Modify the dynamic priority-assignment routine to provide more accurate prediction;

(3) Reduce, or in some way utilize, the control point idle time caused by operator delays.

The first method is the most attractive, but the most difficult to achieve. Past attempts ([4], for instance) have been most concerned with total utilization of only core and central processor time. In the present case we also need a neat dovetailing of I/O requirements, which in turn requires accurate knowledge of the personality of the job. Penny [5] has pointed out the difficulty of obtaining such information in an open-shop scientific computing environment, but there are periods, even in open-shop installations, when "production" jobs predominate. The characteristics of such jobs can be determined and used to modify the scheduling algorithm. The Brookhaven National Laboratory has taken some tentative steps in this direction in an effort to relieve disk congestion [6].

The second method (improving the predictive ability of the priority assigner) offers a little more promise, primarily because the present algorithm is based on the history of the whole job, instead of only its recent history. Many jobs, for instance, observe the pattern of heavy I/O at the beginning, followed by a more or less compute-bound period, and end with a burst of output. Such jobs acquire a high priority during the initial input phase, then lock out other, currently more I/O-bound, jobs while the initial effect wears off, and finally, are themselves locked out when trying to output their results. By considering only

recent history, the priority assigner will more nearly reflect the current character of each job.

The major cause of operator delays in the Chippewa system is the assignment of tape drives to active programs. On some systems (the Atlas system, for instance [7]), great pains are taken to assign tapes before granting access to central memory and the central processor. In Chippewa, tapes must be assigned through the control point, just like any other system resource, and the control point remains idle from the time the request is posted until the operator completes the assignment. There are two practical approaches to the problem: to emulate the Atlas system and preassign all tapes, or to roll out the job during the delay, so that its core may be freed for a more productive program. The course adopted by a given installation depends upon the local job mix, of course: an installation which averages one tape per job, thirty jobs in the input queue, and six tape drives, for instance, is unlikely to gain much from the Atlas method. This choice (between preassignment and roll-out) should be carefully considered, for reduction of dead time is the most promising of the three possibilities for immediate system improvement.

Conclusions. The treatment described here was based on the principle that the object of the scheduling algorithm(s) for multiprogramming systems should be to maximize the number of simultaneous processes. In applying it to the Chippewa system, a 33% increase (from 1.5 to 1.98) in simultaneous processes achieved a doubling of throughput. The particular steps by which these results were obtained (increasing the number of jobs in core, and dynamically assigning priorities on the basis of I/O-boundedness) were -- and in general will be -- easy to implement. Furthermore,

their efficacy is not limited to the Chippewa System: Penny [5] has shown them effective for any multiprogramming system with one level of storage.

Acknowledgements. The following people, wittingly or not, have contributed to the ideas presented here: the programming staffs at CERN and the Brookhaven National Laboratory, especially Charles Symons of the former and Les Lawrence of the latter; the author thanks them. He also thanks Jeremy Knight of the Lawrence Radiation Laboratory, Berkeley, who has implemented much of what has been described; and finally, he thanks the referees for directing his attention to [5].

References.

1. 6600 Chippewa Operating System, Control Data Corporation Publication Number 60124500, April, 1965.
2. Chippewa Operating System Reference Manual, Control Data Corporation Publication Number 60134400, December, 1965.
3. Stevens, D. F.: System Evaluation on the Control Data 6600, Lawrence Radiation Laboratory Report UCRL-17893, Oct. 1967.
4. Codd, E. F.: Multiprogram Scheduling. Comm. ACM 3, 6(1960), pp 347-350 (Parts I and II); Comm. ACM 3, 7(1960), pp 413-418.
5. Penny, J. P.: An analysis, both theoretical and by simulation, of a time-shared computer system. Computer Journal 9,1 (1966), pp 53-59.
6. Lawrence, L. L.: Private communication
7. Howarth, D. J.: Jones, P. D.; Wyld, M. T.: The ATLAS scheduling system. Proc. AFIPS 1963 Spring Joint Computer Conference, pp 59-67. Spartan Books, Baltimore, Maryland.

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.