PROJECT MAC

Computer Systems Research Division

July 2, 1974

Request for Comments No. 55

M.S. THESIS PROPOSAL: A Simple Implementation of Processes as Basis for a Certifiable Computer Utility

by David P. Reed

Attached is a copy of my proposal for a Master's thesis dealing with simplification of Multics traffic control algorithms and interfaces. I've been working on these problems since I completed the proposal, so it is slightly out of date with respect to my own current thoughts. Consequently, I will hopefully be generating further memos as my thoughts crystallize. I am exceedingly interested in others' thoughts on these matters; feel free to come talk to me.

---

Massachusetts Institute of Technology

Project MAC

Cambridge, Massachusetts

Proposal for Thesis Research in Partial Fulfillment

of the Requirements for the Degree of

Master of Science

Title: A Simple Implementation of Processes as Basis for a Certifiable Computer Utility

Submitted by: David P. Reed
9 Hearn St., Apt. A
Watertown, Massachusetts
02172

Signature of Author:

Date of Submission: May 15, 1974

Expected Date of Completion: January 1975

Brief Statement of the Problem:

In order to make sure that the data in a Computer Utility is secure, it is important that the operation of the supervisory programs be certified correct by either human auditing or mechanical means. Since a natural structure for a multiprogrammed operating system employs many asynchronous processes to accomplish its many tasks, the method of realization of these processes plays a major part in the realization of the operating system as a whole. A simple and reliable implementation of processes such as that proposed herein will serve as a useful basis for structuring such an operating system in a certifiable way. The ideas will be tested by a detailed design forimplementation within the Multics operating system.

Supervision Agreement:

The program outlined in this proposal is adequate for a Master's thesis. The supplies and facilities required are available, and I am willing to supervise the research and evaluate the thesis report.

M. D. Schroeder, Asst. Prof. of Elec. Eng.

## I. Introduction

A currently important direction in computer systems research is the development of techniques for construction of large multi-access information utilities which can serve the needs of society for information handling in a reliable, secure way. While the reliability of the hardware part of a computer system has been under continuous research and improvement, until recently there has been little effort to systematize the generation of correctly functioning software operating systems to support multi-access information utilities.

To reach this goal of correct functioning it is important that the algorithms which make up an operating system for an information utility such as Multics be simple enough that a few hours' study of its structure would be enough to convince the average programmer that the system is correctly programmed. Unfortunately, systems such as Multics, TSS/360, and even OS/VS2 generally provide an exceptionally large number of features within an ad hoc structure that is expressed with many tens of thousands of lines of source language.

I propose to attack a small part of this problem in my thesis. The specific set of algorithms which I would like to study in this context are the algorithms which are grouped under the

classification traffic control.

Traffic control is that portion of the functionality of a computer system which is devoted to multiplexing processor and primary memory resources among user computations. Since the functionality of traffic control is essential to any multiprocessing operating system, it is important that it be correct for the correct operation of the computer system as a whole. Unfortunately, the traffic control function of many computer systems is one of the most complicated parts of the system, and certainly causes many of the subtle bugs in those systems. For this reason, research into ways to simplify the structure of the traffic control algorithms and certify their correctness is important in the context of computer systems research today.

Since traffic control serves as a basis for many of the operating systems functions, its functional interface generally exerts a pervasive influence on the structure of the rest of the operating system. The structure of operating systems algorithms which perform asynchronous I/O, for example, is constrained by the particular primitives (properly a part of traffic control) which allocate the processor resources to the I/O control algorithms. In many computer systems these primitives are defined by the concept of "interrupt", and are collectively known as the interrupt structure of the operating system. While recognizing

that the hardware implementation of interrupts is exceedingly simple, resulting in more reliable hardware, I also note that the software algorithms concerned with handling & coordinating interrupts are often among the most complex algorithms in a computer system. By centralizing the software mechanisms which interpret and coordinate interrupts in traffic control, and hiding the very existence of interrupts from the rest of the operating system algorithms, we can hope to obtain a much simpler operating system structure.

As proof of the viability of the resulting design, I will propose a detailed implementation within the current Multics operating system. The Multics system is valuable as a testbed for these ideas for two reasons. First, it is a good example of a state-of-the-art computer utility, and provides good examples of the types of problems which make the implementation of traffic control very difficult. Nevertheless, the Multics system is better structured than most existing operating systems, so I expect few problems to result from issues unrelated to traffic control per se. The problems of Multics traffic control are thus more clear than they would be in another operating system. Secondly, the Multics operating system is a "real" operating system, serving real users, so any positive results of this research are thus more obviously practical than those dealing with limited purpose systems such as, for example, Dijkstra's

THE operating system,[1] or the MITRE PDP-11/45 system kernel.[2] A final reason is that the Multics system is within reach. It is currently under development at MIT, and the author has considerable experience with the system as a user and system programmer.

To summarize, the proposed research will attempt to simplify both the internal structure of the Multics traffic control algorithms, and the external interface structure of these algorithms.

---

1) Dijkstra, E., "The Structure of the 'THE'-Multiprogramming System", Communications of the ACM, Vol. 11, No. 5, May 1968.

2) Schiller, W.L., "The Design of a Security Kernel for a PDP-11/45", MITRE Technical Report MTR-2709, June 30, 1973.

## II. Simplification

A large quantity of recent computer research has been devoted to the problem of proving that programs satisfy certain assertions about their behavior. This proof process, which I will call "validation", consists of showing that certain properties of data are preserved by the application of the program to the data, under some semantic model of the program. As a consequence of this research, the technique of structured programming is being developed as a technique for constructing programs which can be proven to satisfy assertions about their behavior. Several techniques have been developed as part of the structured programming methodology which are particularly applicable to the problems with which I am attempting to deal. I propose to use these techniques where applicable in the context of my new traffic control design.

The concept of "levels of abstraction", which forces the design of the system to correspond to a lattice of models of its behavior, seems to be a useful technique. The design of a system is constrained by this approach to be a lattice of program modules which is similar in structure to the lattice of models. A homomorphism from modules to models allows an easy proof of correspondence between the properties of the models and the properties of the program modules taken as a whole. The important requirement of such a design is that the homomorphism

can be easily demonstrated. As a consequence of this, program modules are necessarily small in size, and simple in construction. Of course, the individual models in the lattice must also be simple.

Another useful technique is the modularization of a designed system of modules based on Parnas's[1] rule of "hiding of information". This rule requires that interfaces between program modules be simply defined, and that all interaction between program modules occur at the interfaces. This rule allows the technique of "stepwise refinement" to proceed in an orderly manner, as well as aiding in construction of proofs of correctness which are invariant under changes to the methods used by modules to achieve their goals. This latter feature is useful in the development of algorithms for operating system programs, which must of necessity change with time and with the requirements of the operating system customers.

A third important program development technique has been described by Wulf,[2] and seems to be primarily applicable to developing operating systems and other multi-purpose environment support software. This technique is called policy/mechanism

---

1) Parnas, D.L., "A Technique for Software Module Specification with Examples", Communications of the ACM, May 1972.

2) Wulf, et. al., "Hydra: The Kernel of a Multiprogramming Operating System", memorandum from Computer Science Department of Carnegie-Mellon University, June 1973.

separation by Wulf, and consists of an attempt to segregate into different program segments the necessary, or mechanism-related, features of algorithms, from the policy-enforcement algorithms required by the particular utilizations of the system. Of course there is no hard and fast dividing line between the two areas, but by choosing an arbitrary line, one can come up with a design which allows a fair amount of flexibility, while retaining those properties which one wishes to prove correct in all conceivable utilizations of the system. A consequence of the division is that the policy algorithms need not be validated with respect to the properties preserved by the system exclusive of the policy mechanisms. Thus failure of the policy mechanism need not necessarily result in "hard" failure of the system. This is not to say that one might not want to validate the policy mechanisms for other reasons, e.g. to verify that the enforced policy corresponds with the expectations of the specifiers of that policy.

An extension of the policy/mechanism separation technique seems to be especially applicable to operating system algorithms in particular. Taking note of the fact that operating systems serve two purposes in multiprocessing environments, resource allocation and resource usage optimization, we can divide the mechanism of traffic control into two parts. I will call these two parts "necessary mechanism" and "optimizing mechanism" for further reference. An example of a necessary mechanism would be code

which assigns processor resources to individual processes, since it is necessary to provide processes with CPU resources. The code which assigns priority to processes in order to maximize system throughput, response time, or other system performance measures would be an optimizing mechanism.

If the separation between optimizing and necessary mechanisms is performed properly, then it will hopefully be the case that the optimizing mechanism need not be validated to the same extent that the necessary mechanism has to be. Once again, the necessary mechanism will operate "correctly" in the face of failure of the optimizations. Obviously the division is arbitrary, since all of an operating system can be viewed as superfluous optimization, or alternatively, all can be viewed as necessary to correct operation of the system. The crucial criterion is exactly what "correct operation" encompasses for the person validating the system, or ultimately on the partial model of the system which the system is being validated against. I will attempt to perform a useful division of this sort on traffic control with two kinds of goals in mind. The first goal is to make traffic control's necessary mechanism simple enough so that it may be validated against a security model. The second goal is to make the optimizing mechanism of traffic control operate in such a way that it need not be validated as strictly to ensure correspondence against the model.

## III. Specific Problem Areas of the Multics Traffic Control Implementation

As I see it there are four specific problem areas in the current Multics traffic control design that relate to issues of complexity and certifiability. In this section I will discuss each of these areas separately, to give the reader an idea of the precise problems I will attempt to solve; in the next section I will discuss a variety of angles of attack on these problems.

A major problem with respect to certifiability is the complexity of the current Multics traffic control code. I know of no one person who has understood all of the functions of the Multics traffic controller in just one sitting with the listings of the code. There are two reasons for this. First of all, all of the code is written in assembly language, resulting in code whose higher order purposes are largely inscrutable. Secondly, the modularity of the code does not correspond easily to the functions provided by traffic control: the design is too well-integrated to isolate separable features. For example, the subroutine "getwork" in the current traffic controller implements, as well as its natural function of giving up the processor to the most worthy process(mechanism), the priority scheduling algorithm (policy), loading/unloading of processes (optimization), process initialization(mechanism), and several

other functions. The problem is that these functions are implicitly provided for by the code, in such a way that they are not functionally separable. Simplifying the code of traffic control will thus be a task of decoupling the code for conceptually separate functions of the traffic controller.

Traffic control's impact on the complexity of other parts of the system kernel is primarily due to the simplicity of structuring separable parallel computations as sequential processes. Dennis explains this as well as anyone:

> If two parts of a system are independently designed, then the timing of events within one part can only be constrained with respect to events in the other part as a result of interaction between the two parts. So long as no interaction takes place, events in the two parts may proceed concurrently and with no definite time relationship among them. Imposing a time relation on independent actions of separate parts of a system is a common source of overspecification. The result is a system which is more difficult to comprehend, troublesome to alter, and incorporates unnecessary delays that may reduce performance.[1]

Unfortunately, in the current Multics implementation, there are a number of process-like tasks which do not take advantage of the existence of parallel sequential processes in Multics. One class of these tasks consists of those parts of the system which handle I/O devices such as disk, teletype, network, etc. These are

---

1) Dennis, J.B., "Concurrency in Software Systems", Computation Structures Group Memo 65-1, Project MAC, MIT, June 1972.

presently coded as "interrupt-driven" algorithms, which must be coded in an extremely awkward manner. The other class is not so obvious, consisting of algorithms which perform the ongoing tasks of binding and unbinding physical and virtual resources to each other. One example of the latter class is the core resource allocation algorithm. Currently coded as the "findcore" algorithm of page control, this algorithm is logically a process which operates in parallel with the demand paging code, and interacts only to a limited extent with that code. In particular, the part of the "findcore" algorithm which deals with moving pages out of core can proceed in a relatively parallel manner with user program execution, and need not run only when the user code encounters a page exception, requiring that a page be brought into core. "findcore" is also an optimization algorithm, the details of which are mostly inessential to the success or failure of the rest of the operating system to operate correctly, while the demand paging algorithm is a necessary mechanism of the virtual memory system. Clark has tentatively explored the construction of a page control algorithm which is structured as a set of parallel processes, and this construction shows promise in terms of simplifying the page control algorithms.

The problem with implementing these process-like computations as processes in Multics is that traffic control as currently

implemented cannot provide the functions needed by I/O interrupt handlers and resource allocation algorithms for two basic reasons. The first is inefficiency, which results from the fact that there are properties of the interrupt handlers and resource allocation algorithms which allow them to utilize a smaller amount of resources than that currently required by the smallest process. The second is delay, due to the relatively large cost of scheduling a process under the current traffic control implementation, as well as delays which may be introduced due to a deterioration of the locking scheme used by traffic control under a heavy scheduling load. Ultimately, both of these problems result from a poorly structured traffic control algorithm, since the inefficiency and delay are caused by traffic control's attempt to make features available to these processes whether or not they are required by the processes.

The third problem is that certain functions in traffic control itself are most easily viewed as processes. For example, the loading (allocating resources to a process so that it is runnable) of a process is currently handled by the last process to run before scheduling the process to be loaded. Thus the process doing the loading bears no relationship to the process which requires the loading. The process doing the loading must wait before doing anything else until the process is loaded. Thus processes have unnecessary knowledge of other processes' resources, making the system as a whole much less secure (due to

violation of Parnas's information hiding principle). This problem is more general, since any process can be forced to do computations logically part of another process, and on another process's data, by an interrupt. This global knowledge of data about individual processes, even within the confines of the supervisor, is cause for greater difficulty in the certification than is necessary. One must prove that the supervisor algorithms do not let such privilieged data out to the user algorithms in the same process, rather than showing that the supervisor has no ability to know such data in the first place.

The last specific problem area in traffic control is the diversity of poorly interacting interprocess syncronization and communication primitives. As well as the two classic mechanisms described by Saltzer[1] and Rapaport[2] there are several others which serve slightly different purposes. A partial list of these mechanisms, which I will call IPSC mechanisms, includes:

A. Block-wakeup (Saltzer's mechanism, improved in several ways)

B. Process Wait and Notify (PWN, due to Rapaport, but simplified by Webber; very similar to Dijkstra's semaphores)

---

1) Saltzer, J.H., "Traffic Control in a Multiplexed Computer System", Sc.D. thesis, MIT, 1966; Project MAC Technical Report MAC-TR-30.

2) Rapaport, R., "Implementing Multi-Process Primitives in a Multiplexed Computer System", S.M. thesis; Project MAC Technical Report MAC-TR-55.

C. Page-wait and Page-notify (like PWN, but with a different priority scheme)

D. Loop-wait (used in some unusual locking strategies)

E. Interrupt masking (used where locks cannot be used for synchronization, because a process is potentially being multiplexed to look like two processes)

F. Interprocess Signals (IPS, used in rings other than the supervisor ring to arbitrarily interrupt a computation)

G. Process Interrupts (a low-level mechanism used to implement IPS)

While these mechanisms are defined in an operational way by the behavior of the associated code, there is no simple axiomatic abstraction of the semantics of these mechanisms. Such an abstraction would be useful in an attempt to understand the interaction of these mechanisms, but the interactions are probably too complex to allow a simple model. Worse, several of the mechanisms (e.g., interrupts) violate the transparency of control flow normally associated with an algorithm in a sequential process, thus complicating the semantics of <u>any</u> code written to execute in a process. Such questions as, "What will happen if I get an interrupt here?" result from this complication, and make the kernel of the system full of unfathomable interactions.

## IV. Proposed Method of Attack

Basically, my goal in the proposed thesis research will be to generate a design for a traffic control system similar in capability to the traffic controller presently implemented in the Multics system. An attempt will be made to solve the problems noted in the previous section by applying techniques described in the section on simplification. As a result of application of these techniques, the new design should be conceptually simpler than the present Multics traffic control algorithm.

A necessary prerequisite for contruction of a simple traffic control design is that the functionality of traffic control be modeled in a conceptually simple way. The levels of abstraction concept can usefully be applied here. A particular example of the use of the principle will be the separation of multiprogramming function (a necessary mechanism) from scheduling policy. To separate these concepts in the model, one must introduce two concepts: a readied-process and a general process. A readied-process is a process which can be assigned a processor at any time, without interaction with schedulers, memory managers, or other unrelated system modules. Thus it is analogous to Dijkstra's Level 0 process abstraction. The general process concept is analogous to the Multics definition of process, and is subject to scheduling policy, memory management decisions, etc. It will then be the job of a scheduler to make

the processes for which it is responsible into readied-processes, by using a create-readied-process primitive which initializes the relevant parameters of a readied-processfrom the description kept by the scheduler. Unscheduling will use an analogous destroy primitive. These primitives seem to be a reasonable start on an interface between the necessary mechanism of traffic control and the policy/optimization mechanisms. It will of course be necessary to extend the set of primitives to deal with the problems of interprocess communication and other functions which must be implemented at a higher level.

Further, I propose to use readied-processes in the implementation of some of the higher-level traffic control functions, such as loading (allocation of certain necessary resources needed by a process before it can become a readied-process), timer management, and scheduling policy algorithms.

These readied-processes may also be of use in constructing a more restrictive type of process which can be of use in replacing interrupt handlers in the current Multics design by processes executing the interrupt handler code. These processes need not be as general as the general process implemented in the design, for interrupt handling does not require many of the features of general processes. This last consideration leads to the realization that other levels of abstraction may exist in building up the concept of general processes. I propose to try

to determine a natural set of abstractions for use in this construction, using a similar approach to that which led to the idea of readied-processes above.

The traffic control algorithms in the design should be capable of being coded in a higher-level language. This requires taking note of the relative costs of features in the higher-level language of choice in order to choose the most efficient implementation. The higher-level language chosen will be PL/1, since the Multics operating system uses this language for much of its implemenation.

An attempt will be made to unify the IPSC mchanisms, by constructing them on a common base mechanism. It was once thought that Saltzer's block-wakeup mechanism was sufficient for implementation of an operating system, but Rapaport describes the "lost wakeup" problem which results. I will attempt to provide a basic mechanism similar to block-wakeup which is both simple, and capable of supporting all of the present mechanisms.

Finally, I will investigate the security problems of traffic control, by attempting to describe the communication paths between processes caused by the algorithms used to implement the basic functions of traffic control. There are serious research questions here, since the necessity of sharing limited resources

among a set of suspicious processes causes a number of potential information pathways which seem to be very hard to block. I will try to deal with these problems by eliminating as many possible paths from the design, as well as describing the ones which cannot be eliminated.

## V. Research Plan

I plan to pursue the proposed research in three phases. The first phase will involve characterizing the functionality of traffic control as presently implemented in Multics. The final form this characterization will take is probably a combination of a high level description of the interfaces provided by traffic control and a lower-level description of the algorithms actually used. Part of this work is already completed, but I am not satisfied with the clarity and completeness of the characterization. A reasonable estimate of the time needed to complete this work would be one or two months over the summer, though some of the work can proceed in parallel with the second phase.

The second phase will involve the generation of a design for traffic control which supports the same functionality, but which is simpler in structure. This will require the development of an abstract model of the traffic controller function against which the design can be compared; I expect that such a model can be generated using the results of the first phase. The time require for this phase will be on the order of a month or two at most.

The third phase will involve verifying the success of the second phase, and can proceed in two ways. First of all, a test

implementation within the Multics system can be made. This will allow a determination of whether the design works at all. Secondly, I will attempt to outline the correspondence between my model of the traffic control functions and the actual code which implements those functions. We can thus determine whether the valuable properties exhibited by the model are retained in the actual design. I expect that the time required by this phase will be several months. I will require Multics development machine resources during this phase, also, as well as computer time to develop the test implementation.