as priority disciplines, and are categorized according to priority
dependence on running time, system state, external factors, or wait-
time. A single-server system is assumed for most of the discussion,
but a section on multiple-server systems is included at the end of the
paper.

The summary would seem to be of most value to someone enter-
ing queuing theoretic work with a commitment to computer appli-
cations. The bibliography is an acceptable entry to the literature,
where the mathematical developments are given. The discussion
points out a sufficient number of alternatives to show the potential
of this area for interesting and challenging problems.

D. W. Fife
Lexington, Mass.

C. MULTIPROGRAMMING

R68-48 A Multiprogramming Monitor for Small Machines—G. D.

This paper describes a monitor system for a small display com-
puter, attached as a satellite to the Berkeley SDS-940 time-sharing
system. It describes some modifications made to the interrupt-
handling hardware of the small machine, a DEC PDP-8. It also gives
details of a set of calls to the monitor, which allow multiprogramming
of tasks in the satellite.

The author's competence is demonstrated by the way he has
tackled three important features of the system. First, he has added
a useful extension to the PDP-8 interrupt system: this includes an
arm/disarm feature, such as can be found on a number of recent SDS
designs, and a flag scanner. Second, he has kept the size of the monitor
down to 300 words, a remarkable achievement considering the weak
instruction set of the PDP-8. Third, he has recognized that a satel-
lite display computer must be capable of executing the user's real-
time programs. A number of similar systems have been produced
which do not possess this capability, because the satellite can only
refresh the display and process interrupts.

The author appears to have devoted little attention to a fourth,
and equally important, aspect of the system, namely, the provision of
a simple and convenient language for writing real-time graphical
programs. Instead, these must be written in PDP-8 assembly lan-
guage. The author has provided a number of monitor calls, such as
fork, wait, and continue, which may be imbedded in such pro-
grams; in this way the user can multiprogram the various input and
output tasks. The paper does not offer any convincing reason why
I/O should be multiprogrammed in this way. An example is given,
of identifying objects on the screen by pointing at them; applying
multiprogramming techniques, this becomes a program of quite
needless complexity, with four forks and six separate multipro-
grammed tasks. This may have been a poor choice of example, or
the author may have programmed it in this way in order to demon-
strate the use of multiprogramming. One is left with the suspicion
that it could have been greatly simplified by not using multipro-
gramming techniques, and that the same probably applies to the
majority of graphical programs.

The paper is well written, and presents a very clear account of
the system.

William M. Newman
University of Utah
Salt Lake City, Utah

R68-49 Virtual Memory Processes and Sharing in MULTICS—R. C.

This paper gives a description of the "virtual machine" of MULTICS,
the multiple-user computer system conceived at M.I.T. and imple-
mented on the General Electric Model 645 time-sharing computer.
The virtual machine is the machine which is seen by the system
"user," i.e., the application programmer working at the assembly
language level. The MULTICS virtual machine has a number of salient
features which distinguish it from conventional machines.

1) The virtual machine provides a very large (2^32) words "virtual
memory" for each of potentially many concurrent processes, where
a process is defined to be the activity carried out by the system in
executing a given user's program. A virtual memory address has two
components: a) a segment number, identifying one of as many as 2^4
segments per process, and b) a word number, identifying a word within
a segment. Segments may consist of executable instructions ("pro-
cedure segments") or of data ("data segments"). The large size of the
virtual memory relieves the user of the burden of preplanning the
transfer of data and instructions between physical storage devices,
and makes users' programs independent of the nature of these stor-
age devices. The "two-dimensional" virtual memory simplifies the
writing of programs involving multiple data arrays of dynamically
varying size; the program has, in effect, two edges to grow on instead
of the usual one.

2) Procedure segments and data segments may be shared among
a number of concurrent processes. This prevents core storage from
becoming cluttered with multiple copies of the same routines and
files.

3) A process may refer to procedure segments and data segments
dynamically, i.e., the decision to use a given segment may be deferred
until the actual running of the program. In this manner, virtual
memory is allocated only to those segments which are actually re-
quired.

4) Any segments may be recompiled and substituted for its
original version without change to the segments which refer to it
(provided the change does not affect the external characteristics
of the segment). This feature simplifies the development of segment
libraries which are shared by many users.

The foregoing features imply certain capabilities in the system
hardware and software which are not found in conventional systems.
For example, the large virtual memory implies a hardware "paging"
mechanism in which pages of instructions and data are transferred
between core storage and some form of secondary storage as required
by the active processes; and an address translation mechanism which
translates virtual memory addresses into actual core addresses while
the program is being executed (dynamic program relocation). The
sharing of procedure segments among unrelated processes imposes
certain restrictions on the way such segments are written, e.g., they
cannot modify themselves. The dynamic referencing and substitution
of segments implies that such references be held in symbolic form,
and that a translation from the symbolic form to the virtual memory
address form be carried out the first time the reference is encountered
in a process.

This paper is concerned mainly with explaining how the foregoing
capabilities are accomplished in MULTICS, particularly the capability
for the dynamic translation of symbolic addresses. Almost no attempt
is made to justify these capabilities, presumably on the assump-
tion that the reader is already convinced of their value. (For readers not
so convinced, a number of references are given. In particular, an
earlier work of Dennis1 gives some useful arguments for the concept
of a two-component virtual memory.) The paper is carefully and
accurately written.

W. C. McGeer
IBM Corporation
Palo Alto, Calif.

1 J. B. Dennis, "Segmentation and the design of multiprogrammed computer