*Reprinted from* IBM Systems Journal

Volume Seventeen | Number Four | 1978

# Performance investigations with a DOS/VS-based operating system model

by W. Kraemer

*This paper describes an operating system model that is based mainly on the DOS/VS supervisor but also reflects various design alternatives, providing a flexible tool for operating system design and tuning. The model is characterized by the subdivision of I/O activity into normal I/O, page I/O, and fetch I/O, corresponding to the different supervisor services involved. The model has been evaluated by analytical queuing methods in a set of APL functions that allow a flexible specification of the supervisor, the configuration, and the workload. Validation has been done by simulation and by benchmarking of a real system.*

*The main features of this performance tool are described, and its capabilities are illustrated by performance results that show the impact of workload and various supervisor changes on system performance.*

# Performance investigations with a DOS/VS-based operating system model

by W. Kraemer

Predicting the overall performance of computer systems is a very complex problem and is influenced by many parameters between which arbitrarily close interrelationships exist. Performance prediction is met by modeling and measurement techniques. In modeling it is often necessary to restrict attention to those parameters that act as the main influences on a system in order to make the abstraction reasonable and accessible to simulation or analytical techniques. Both methods—simulation and analytical techniques—have specific advantages and disadvantages, but they both can contribute to a better understanding of system behavior if properly used. The system behavior, obviously, is also influenced by the operating system itself, i.e., the path lengths, scheduling, structure, etc.

This paper describes the modeling of DOS/VS (Disk Operating System, Virtual Storage) for the purpose of determining its influence on system performance.[1] DOS/VS was developed for the low end of the System/370 line[2] and was extrapolated from System/360 DOS by introducing virtual storage concepts. Its design was influenced especially by small real storage size requirements. Performance was improved in subsequent releases. The functional structure of DOS/VS has been described in a general survey by Birch;[3] specific details can be obtained from the DOS/VS Supervisor Program Logic Manual.[4] Within the context of the paper, DOS/VS is described only to the extent necessary to develop and understand the queuing model.

In this paper, the model is described first. Then the queuing analysis and the APL program are discussed thus leading to the validation and the performance results.

## DOS/VS-based supervisor and system model

**modeling highlights**

Modeling computer systems consists of a careful deduction of a system model, hopefully being neither too gross nor too detailed. Generally speaking, a model can be considered as a sound abstraction of reality if all parameters are included that may have a noticeable influence on the investigated performance values. For the sake of brevity, only some highlights are given within this section to illuminate the important procedure of modeling.

The *first modeling step* is a reduction of the detailed flowcharts in the Program Logic Manual.[4] These flowcharts are vast, reflecting the complexity of an operating system that is further increased by:

- The variety of devices supported.
- The need for compatibility with former systems.
- The demands for high reliability, availability, and serviceability.
- The different access methods.

Simplification of the flowcharts was done for various "requests" that call for supervisor code to be processed. In DOS/VS, these requests can be subdivided into interrupt requests and so-called "requests for system tasks." Interrupt requests have the highest priority and are subdivided into:

- MC —machine checks (equipment malfunction)
- PC —program checks (improper use of instructions or data, addressing exception, page fault)
- SVC —supervisor calls (calls for special supervisor services, approximately 80 different SVCs used)
- I/O —input/output interrupts (channel end, device end)
- EX —external interrupts (timer, attention)

The system tasks represent different supervisor services, which also have different dispatching priorities. They are listed below with their abbreviations:

| | |
|---|---|
| • RAS | is the reliability, availability, and serviceability system task. |
| • PMGR | is the page manager system task (page fault handling with selection of page frames, page in/out of pages). |
| • SUPVR | is the supervisor system task or fetch task (fetching of user phases and nonresident parts of the supervisor). |
| • CRT | (cathode ray tube) is the screen manager console support. |
| • ERP | is the error recovery procedure system task (I/O malfunction). |
| • PAGEIN | is the special service to page in several pages. |
| • ATTN | is the attention system task. |

The flowcharts associated with these requests have been systematically reduced to slimmer charts that contain branches to different paths, but in most cases wind up at the general exit routine EXIT, the dispatcher.

To perform an I/O operation, many of these different levels are needed. For example, during a page I/O the program check, SVC, I/O interrupt, and the page manager levels are involved. At the end of the first modeling step, all of these different levels have been represented by their essential parts.

*The second modeling step* led from representation by level to the final supervisor model by

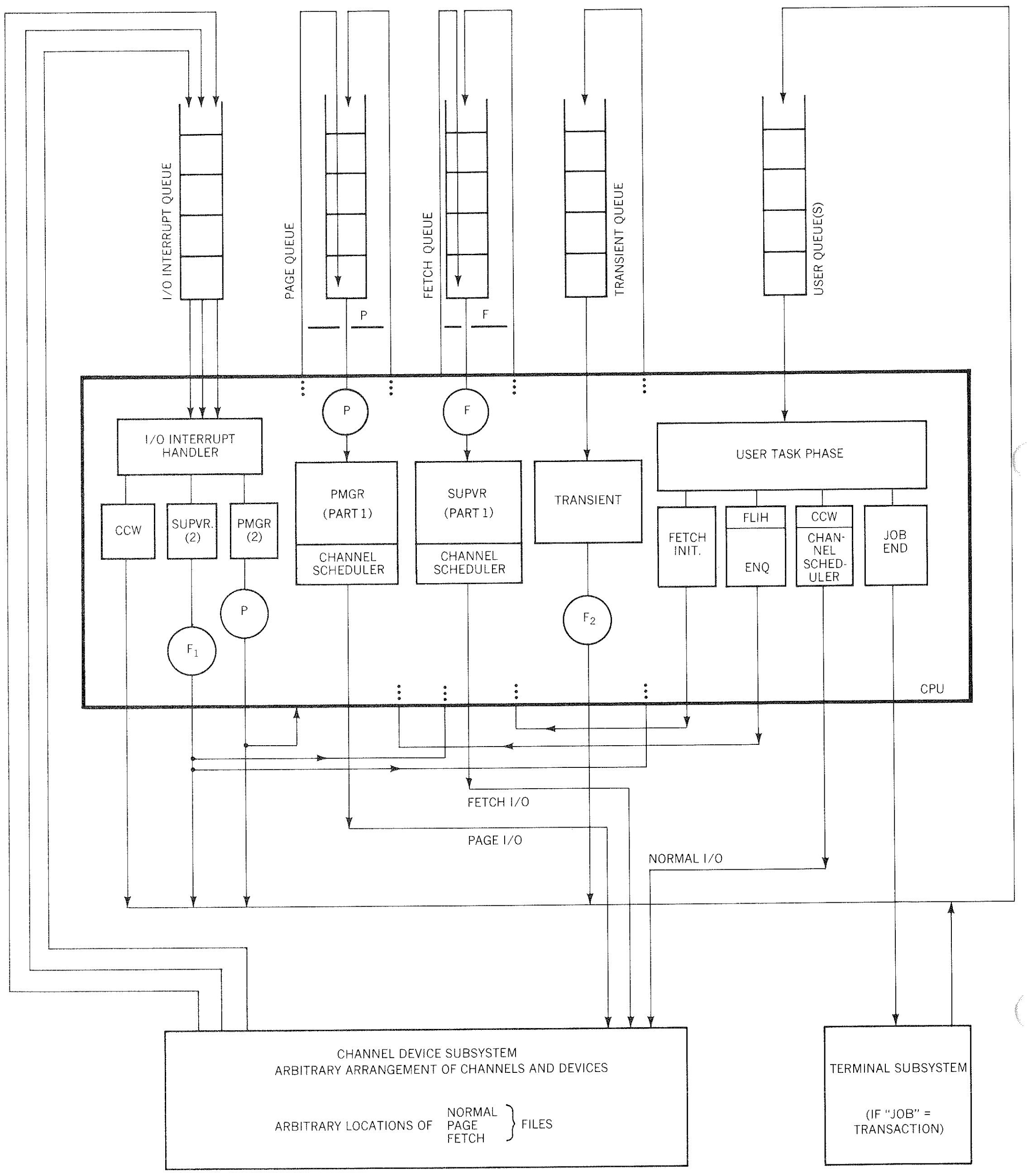
- Omission of parts that are uncritical with respect to performance (machine checks, external interrupts, and program checks except page faults)
- Reduction of the number of system tasks (RAS, CRT, ERP, and ATTN all use the supervisor fetch task to fetch transient parts or phases of the supervisor, whereas PAGEIN is using the page manager)
- Consideration of (synchronous) SVC interrupts either as calls for supervisor subroutines or (when issued by user tasks) as switches into the supervisor state.

Furthermore, according to the various needs for supervisor support, three types of I/O operations have been distinguished:

1. Normal I/O} user-specified I/O
2. Page I/O (using the page manager system task)
3. Fetch I/O (using the supervisor fetch task)

$\left.\begin{array}{l}\text{2. Page I/O (using the page manager system task)}\\ \text{3. Fetch I/O (using the supervisor fetch task)}\end{array}\right\}$ "supervisor I/O"

Figure 1 DOS/VS supervisor model



The final important step was to introduce the channel device system and to close the loops for normal I/O, page I/O, and fetch I/O, thus obtaining a system model that is described in the following sections.

**global description** Figure 1 shows the resulting model for the DOS/VS supervisor. The different boxes within the CPU can be described by path lengths that lead to varying service times with the mean depending on the appropriate MIPS (million instructions per second) value. The CPU has four supervisor queues and one queue for problem processing. The queues are arranged according to their priorities, the highest one being formed by I/O interrupt requests, followed by requests for the page manager (PMGR) and supervisor fetch task (SUPVR). The lowest-priority supervisor queue (transient queue) consists of requests for the processing of transient phases that have already been fetched into real storage.

If no supervisor work can be done, the CPU tries to do problem processing, i.e., to serve a user task in the user queue(s). Within the context of this paper, "user task" may denote a batch job or a data base transaction. Note that the entire processing of a single user task is cut by its own I/O operations into several compute intervals of user code processing, here called *user task phases*. Therefore, the user queue is formed by all active user tasks that are selectable either for the beginning of the next user task phase or the continuation of the currently interrupted one.

In DOS/VS (Release 32) up to five different batch partitions with *different* so-called dispatching priorities may be active. For such a case of multiple batch partitions, the user queue may be considered to consist of several subqueues arranged in priority order (sublevels). The case where all tasks in the user queue have the *same* dispatching priority (multitasking within a single partition) has also been considered and is discussed later.

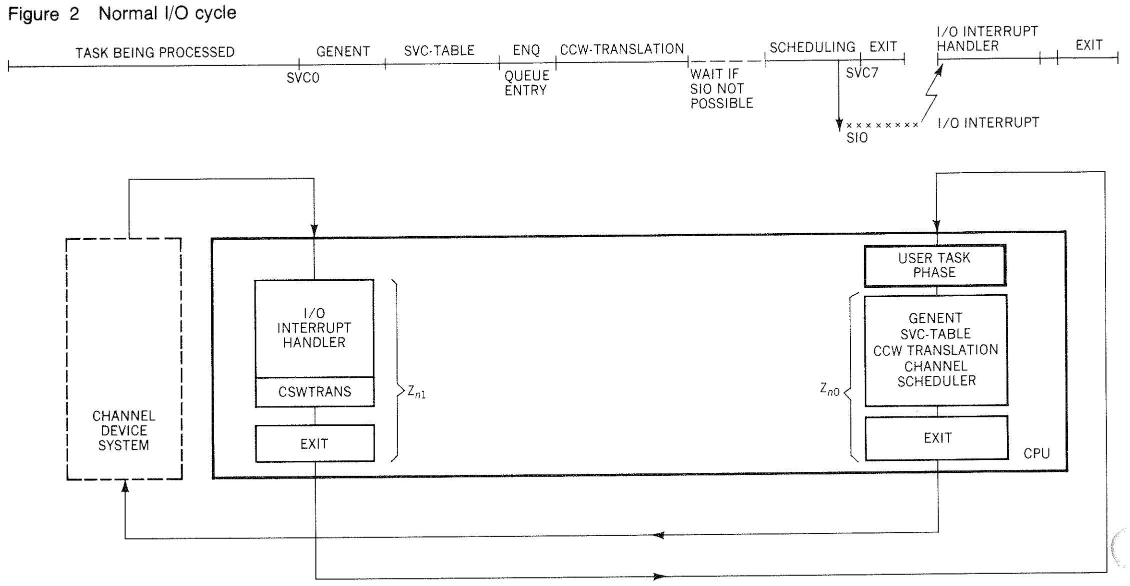
A user task that may be in different states—waiting or being served—(problem state, supervisor state for normal/page/fetch I/O, or performing I/O) is said to "walk through the model," where the present location indicates its momentary status. Please note in Figure 1 that all boxes within the CPU except the user task phase box are processed in the supervisor state.

Consider a single user task that goes through a user task phase. This phase of user code processing can be terminated because of four possible reasons: (a) a request leading to normal I/O (SVC0), (b) a request leading to page I/O (page fault program check), (c) a request leading to fetch I/O (SVC1 . . .), or (d) a task end.

In the case of normal I/O, after the CCW (channel command word) translation, the SIO (start I/O) command is given. An I/O interrupt signals the completion of the I/O transfer, making the corresponding partition or task dispatchable again.

In the case of a page fault, the page manager is needed to select the page frame and determine whether the contents of the frame have to be transferred out in advance, thus possibly being needed twice (page out and page in).

Figure 2 Normal I/O cycle



In the case of a request to fetch a transient phase, a request for the supervisor fetch task is initiated. When the transient is in real storage, it has to be processed before the appropriate user task is dispatchable again.

Note that the last parts of the page manager and supervisor fetch task here have been appended to the I/O interrupt handler. In DOS/VS these two system tasks are only serially reusable in order to serialize processes and to obtain smaller control tables within the supervisor. Therefore, the model also contains two possible gates P and F, which are explained later.

For a better understanding of the function and application of the model, the different supervisor components will be treated separately in the subsequent discussion. This treatment serves to calibrate the model. It is very helpful that, apart from paging activity, such models can be calibrated to a wide extent by supervisor and problem state measurements in a single-thread environment.

**normal I/O cycle**

Normal I/O has been defined as being all I/O operations not needing the page manager or supervisor fetch task service. Figure 2 shows both the time diagram for a normal I/O cycle and the associated schematic flow through the model.

The total number of supervisor instructions being processed from an SVC0 (supervisor call 0—execute channel program) until the next task is selected by the dispatcher is

$$Z_{n0} = Z_{\mathrm{GENENT}} + Z_{\mathrm{SVCTAB}} + Z_{\mathrm{ENQUEUE}} + Z_{\mathrm{CCW}} + Z_{\mathrm{CHQ}} + Z_{\mathrm{SIOH}} + Z_{\mathrm{SVC7}} + Z_{\mathrm{EXIT}}$$

where

| | |
|---|---|
| $Z_{\mathrm{GENENT}}$ | is the number of instructions required to proceed through the general entry subroutine GENENT. |
| $Z_{\mathrm{SVCTAB}}$ | is the number of instructions required to analyze the type of SVC. |
| $Z_{\mathrm{ENQUEUE}}$ | is the number of instructions from the channel scheduler to the label GIOADR (mainline, excluding CCW translation for normal I/O). |
| $Z_{\mathrm{CCW}}$ | is the number of instructions necessary to translate one channel program for one physical normal I/O request. |
| $Z_{\mathrm{CHQ}}$ | is the number of instructions from GIOADR to SIO (proper channel queue scheduling). |
| $Z_{\mathrm{SIOH}}$ | is the number of instructions for start I/O issue and handling. |
| $Z_{\mathrm{SVC7}}$ | is the number of instructions for the WAIT macro (processed if no I/O compute overlap for the same job follows, including additional GENENT and SVCTAB part). |
| $Z_{\mathrm{EXIT}}$ | is the number of instructions in the general exit routine for task selection (the dispatcher). |

The completion of the transfer is signaled by an I/O interrupt. Let $Z_{n1}$ be the (mean) total path length from a normal I/O interrupt until this I/O action is fully completed. Then

$$Z_{n1} = Z_{\mathrm{IOIH}} + Z_{\mathrm{CSWTRANS}} + Z_{\mathrm{EXIT}}$$

where

| | |
|---|---|
| $Z_{\mathrm{IOIH}}$ | is the number of instructions for the I/O interrupt handler (from label ENTIO over GENENT, INTRTN, CHNDRT, TRNOFF to EXIT; excluding CSWTRANS) |
| $Z_{\mathrm{CSWTRANS}}$ | is the number of instructions necessary for normal I/O retranslation in CSWTRANS. |

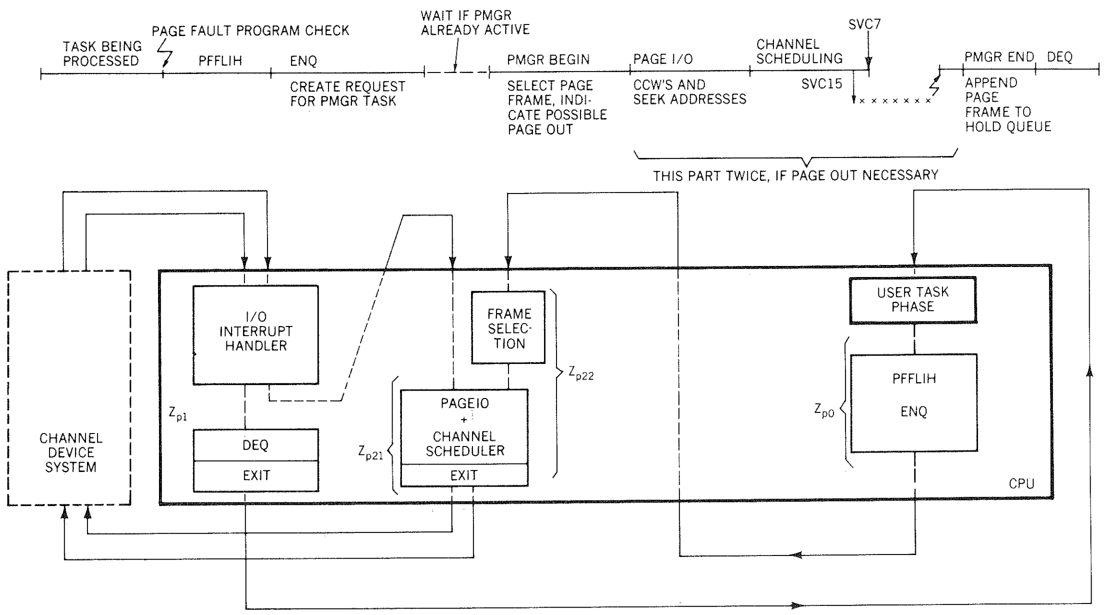
In the case of multiprogramming, a SIO command cannot always be issued directly after the CCW translation because the channel or device may be busy. At such times, the SIO command actually is issued later as a consequence of an I/O interrupt caused by an I/O completion of another task. It is of little importance whether the corresponding path length $Z_{\mathrm{CHQ}} + Z_{\mathrm{SIO}}$ is said to be included in $Z_{n0}$, as used here, or in $Z_{n1}$.

Note that, as indicated in the time diagram, I/O activity and supervisor processing due to the same task are overlapping each other. This overlapping means, for example, that a seek, beginning after the SIO, runs in parallel with SVC7 and the processing of EXIT.

**page I/O cycle**

A page I/O cycle includes all actions to fetch a page into real storage (page in). The cycle begins with a page fault program check

Figure 3 Page fault cycle



and ends when the requested page has been transferred from secondary storage to real storage. Most of these actions are performed by the page manager system task (PMGR) as denoted in Figure 3.

Each page fault program check first is handled within the page fault first-level interrupt handler (PFFLIH), from which a request for the page manager is set up (path length $Z_{po}$). When control is passed to the page manager, a frame for the requested page has to be selected with the help of the page replacement algorithm. In DOS/VS the so-called Q-class algorithm (see Reference 4) subdivides all frames into five queues. One of them is the hold queue, consisting of all newly occupied frames, the other four being characterized by all combinations of reference bit and change bit values. The associated path length for frame selection can be assumed to increase linearly with the size of the selection pool containing all selectable pages. Then, it has to be indicated whether a preceding page out is necessary, i.e., whether the selected frame contains a changed page. In this case, two separate accesses to the page data set are necessary, also using the page manager and the scheduler twice.

After the I/O interrupt of the page in has been handled by the I/O interrupt handler, this page is protected momentarily by appending the associated frame to the hold queue. When the request for the page manager is dequeued, the next work to be done by the CPU is determined by the dispatcher EXIT.

Table 1 Transients in DOS/VS

| *Name* | *Purpose* | *Remarks* |
|---|---|---|
| Logical transients (B transients) | Attention routines<br>Terminator routines<br>Special service routines<br>Operator console support (CRT transients) | Transients for normal operating conditions |
| Physical transients (A transients) | Error recovery<br>Error recording | Transients for special conditions |
| RAS transients (R transients) | Reliability<br>Availability<br>Serviceability | Transients for special conditions |

As previously mentioned, the model contains a gate P for serialization purposes (Figure 1). It is closed when a request has passed it ((close-gate-)P circle) and opened when the transfer of the page has been completed and the control table has been updated. This is done after the page manager part 2 in the (open-gate-)P circle. Naturally, by omitting gate P, the design alternative of a reentrant page manager is included in the model.

**fetch I/O cycle**

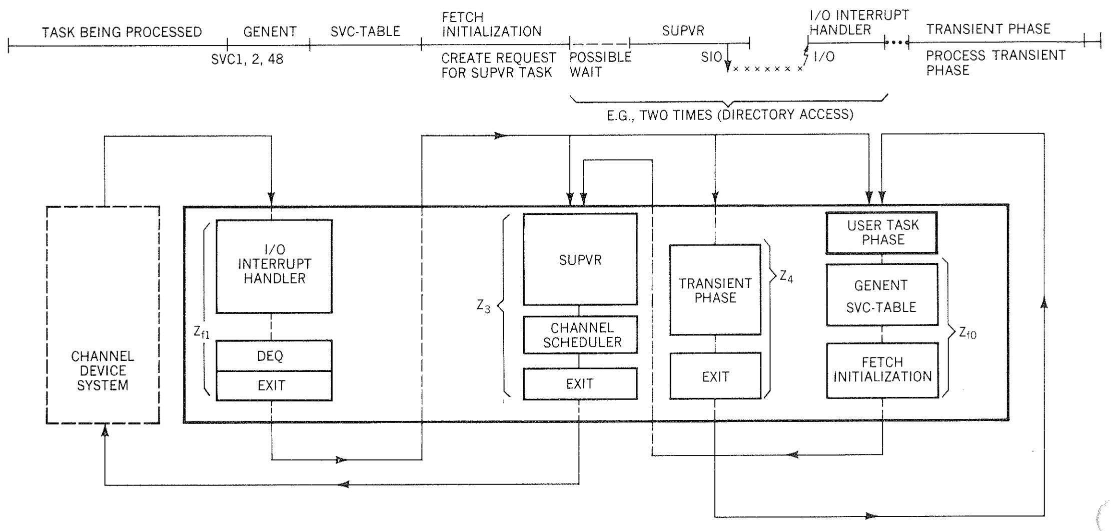
Fetch I/O in the DOS/VS model is that I/O activity using the fetch macro and, therefore, the supervisor fetch task (SUPVR). This task is a serially reusable software resource for DOS/VS, i.e., nonreentrant. Primarily this task serves to fetch nonresident parts of the operating system (transient phases) initiated by supervisor calls which here are interpreted as being in the user program. Table 1 shows such transients as they relate in DOS/VS.

For the DOS/VS model, the transients for normal operating conditions (logical or B transients) are especially of interest. A certain fraction of these B transients are fetched into the so-called Logical Transient Area (LTA) with fixed location in real storage.

In Figure 4 a fetch I/O cycle begins with a fetch-SVC instruction and ends when the transient is processed. Such a cycle may consist of two physical I/O accesses, including a previous access to the appropriate directory. In addition, the model contains a path directly from the I/O interrupt handler to the user queue in order to reflect the possibility that fetched code runs in the problem state with partition priority.

Furthermore, it should be noted that in DOS/VS an entire sequence of transients may be fetched piecemeal. This fact is not explicitly

Figure 4 Fetch I/O cycle



represented in the current status of the model, but can be considered globally by increasing the number of transients fetched by a user partition.

To reflect the serial reusability of the fetch mechanism in DOS/VS, a gate F has been introduced in the model (Figure 1). This gate is closed in the F circle when a request has passed it. As previously indicated, a certain fraction of the transients must be loaded into the (single) Logical Transient Area which is then occupied until the transient is processed completely. Therefore, this transient area can be used only after the processing of the previously fetched transient. This potential bottleneck is reflected in the model by providing the means to open the F gate not in the $F_1$ circle but in the $F_2$ circle after the transient has been processed.

Note that for page I/O and fetch I/O no CCW translation is necessary and that these two types of "supervisor I/O" are preferred in DOS/VS within the device queues by the so-called head queue supervisor call SVC15 instead of using an SVC0 as for normal I/O.

**priority considerations**

The CPU allocation problem is solved by the dispatcher which is now discussed. The dispatcher (or general exit routine EXIT) is called when an interrupt request, a request for system task, or a user task request has been processed and the decision has to be made as to which task will be activated next.

Apart from I/O interrupt requests, coming "asynchronously" from outside the CPU, the interrupts taken into account by the model (SVCs, page fault program checks) are "synchronous" because they are switches into the supervisor state, originated by user tasks, or calls for special supervisor services, originated by the supervisor itself, which are reflected in the different supervisor path lengths.

These SVC and program check routines have highest priority for the allocation of the CPU. Therefore, some of these routines (the ones that are directly originated by user tasks via synchronous program checks or SVCs) have been quasi-attached to the user task phases. These routines are processed with highest priority and normally would only be interrupted by machine checks, which are ignored here because they are exceptional conditions. Therefore, requests are never waiting for this level.

For the remaining levels in the model, priority is given in the following order (Figure 1):

1. I/O interrupt queue
   Page queue
3. Supervisor fetch task queue } supervisor queues
4. Transient queue
5. User queue

Within the page queue and the supervisor fetch task queue a request may have head-of-the-line priority. To use an example, in the case of a serially reusable page manager, a request to fetch a page into a page frame that was freed by a previous page out will be handled before a page manager request for another task.

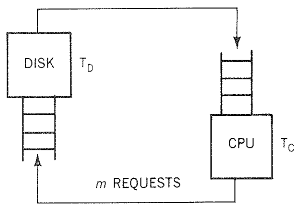
The user queue is last and consists of all dispatchable user tasks, which in addition may have different partition dispatching priorities.

Two possible cases have been considered: (1) a multitask case with one partition and (2) a multipartition case with one task (job) per partition.

he first case assumes *m* tasks with the same dispatching priority, e.g., on a FIFO base (first-in, first-out). When a user task *i* is interrupted by an I/O interrupt request that belongs to another task *j*, then the interrupted task *i* is resumed after the I/O request has been handled, resulting in multitasking within one partition. This mode of operation is valid in such cases as a pure teleprocessing application (transactions); therefore, the model includes a terminal subsystem for this case.

The second case is normal multiprogramming with several active batch partitions that are interrupted by I/O interrupt requests. If, for example, such an interrupt belongs to an I/O request of a task that resides in the partition with the highest dispatching priority, a lower-priority task is further delayed.

Figure 5 Simple multiprogramming model



These dispatching priorities will be further clarified in the next section.

The code that can be interrupted by I/O interrupt requests must be specified. There is no question that it is useful and even necessary to let these interrupt requests immediately interrupt user code (being processed in the problem state). Naturally, this is reflected in the model.

For lower-priority supervisor code it might be useful to allow interruptions only at certain points, in order to preserve the integrity of control tables. If I/O interruptions are not admitted, this supervisor code is said to run disabled. The degree of interruptibility depends on design objectives and may be large for very sophisticated operating systems with fast response times for extremely urgent tasks, or may be significantly limited as is the case with DOS/VS, an operating system for the low end of a computer line.

Therefore, two different borderline cases of interruptibility have been implemented in the model, namely (1) I/O interrupt requests intercept all lower-priority supervisor code, and (2) all supervisor code runs disabled. For DOS/VS the second case is more appropriate than the first one.

## Queuing analysis

**general remarks**

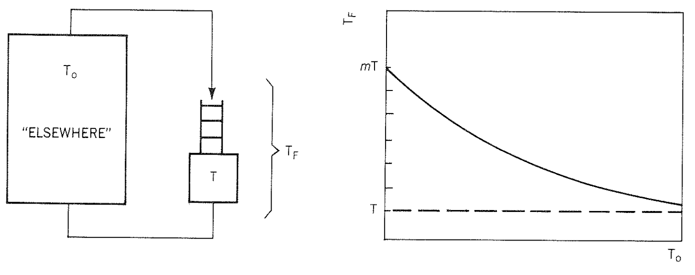
Queuing models can be analyzed by analytical queuing methods as well as by simulation. Both methods have their specific advantages and disadvantages, and both methods can enhance performance prediction.

Normally, the first step in making a queuing model manageable by analytical methods is to randomize the processes, i.e.,

- To describe service times by independent random variables
- To describe the routing strategy by independent branching probabilities
- To homogenize task behavior by assuming fixed mean values of service times and branching probabilities
- To base the investigation upon the assumption of steady state.

The method developed for the DOS/VS model is based on the decomposition of a queuing network into quasi-independent parts, such as so-called machine-repair models, as was done by Florkowski in the XMODEL tool.[5] This highly efficient network decomposition principle, which is summarized shortly, has been applied in a similar way to the DOS/VS-based system model, which in addition is characterized by priorities, gates, etc. Actually, the main

Figure 6 Machine-repair model and principal result for flow time



problem was to calculate the complex CPU part that differs remarkably from a simple machine-repair single server. For this reason efficient procedures and iterative approximation methods have been developed.

**network decomposition principle**

The decomposition method developed by Florkowski is basic to the queuing analysis employed for the model and is therefore briefly summarized. The decomposition principle is further described in Reference 5. Consider a simple multiprogramming model of a system with one CPU and a single disk, which is characterized by $m$ programs or requests cycling around as depicted in Figure 5.

Let $T_C$ be the mean time a program is in a CPU phase, and let $T_D$ be the mean time the disk is busy. Each node (CPU, disk) of this queuing network is modeled as a machine-repair single server with $m$ sources. The mean idle time of each source is set equal to a so-called "elsewhere time" $T_0$ related to the node under investigation (Figure 6). For the CPU the elsewhere time corresponds to the flow time of the disk, whereas the elsewhere time for the disk is the flow time of the CPU node.

Figure 6 also shows the mean flow time $T_F$ of a single node as a function of the mean elsewhere time $T_0$. The throughput rate $\lambda$ for a single node turns out to be

$$\lambda = \frac{m}{T_F + T_0}$$

To obtain a certain throughput rate for a single node, the mean elsewhere time $T_0$ must be changed iteratively. The machine-repair procedure used performs this iteration internally. The solution of the whole network is achieved as follows. Starting with an assumed initial value $\lambda_0$ for the throughput rate $\lambda$ in the whole network, we calculate each node separately with the machine-repair procedure. The result of this first step is a throughput rate value

$$\lambda_0{}^* = \frac{m}{T_{FCPU} + T_{FDISK}}$$

Based on this resulting value $\lambda_0^*$, the assumed throughput rate is changed in subsequent iteration steps until the assumed rate and the resulting rate are nearly identical. This solution point is graphically demonstrated in Figure 7.

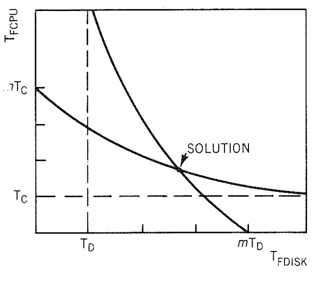
The solution point is generally found in a few numerical iteration steps. In the APL program for evaluating the model, less than 10 iteration steps are necessary for a relative error of less than $10^{-4}$ in the throughput rates.

**CPU queuing model**

To determine the different waiting times within the CPU queues it is not sufficient to separate the CPU from the residual queuing net (the channel device system), since the CPU server is governed by priorities and feedbacks that are not included in normal machine-repair models.

Figure 8 shows the queuing model of the CPU, which is an offspring of the supervisor model obtained by calculating resultin service times and branching probabilities. Essentially there are five queues, four supervisor queues (priority levels 1 to 4) and one user task queue (priority level 5) for all dispatchable user tasks. This lowest-priority level can be further subdivided into different priorities (partitions) $5_1 \cdots 5_m$. Note that due to the present status of the APL program, directory accesses are not directly included ($P_D = 0$). But they can nevertheless be considered to be globally included by way of the fetch path taken with probability $P_u$.

Figure 7 Common solution point



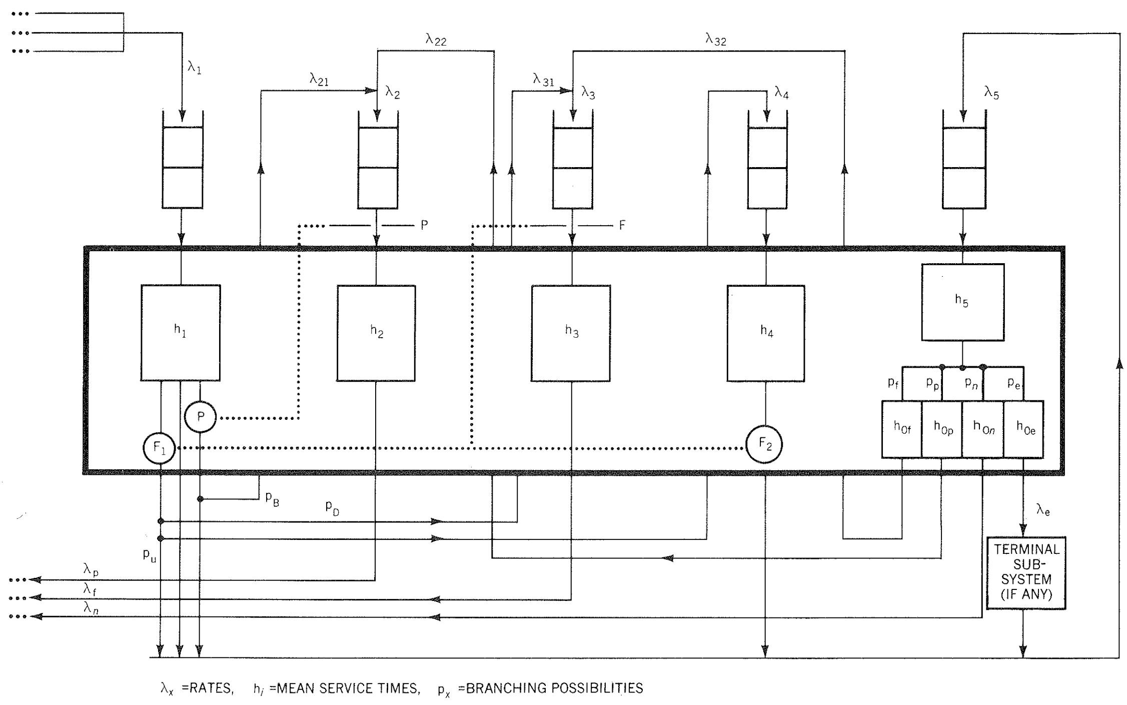
The mean service times $h$ (holding time) are indicated in Figure 8 and can be determined by the different supervisor path lengths, specified earlier in the paper, and in the associated MIPS values. If a service time has been completed in the model, the dispatcher selects the next request having highest priority, if any. The supervisor service at level 0 has the highest dispatching priority and is never interruptible. Since all user task phases are fully interruptible, no queue for level 0 is necessary. Therefore, level 0 need not be considered in the dispatching scheme for the queues a shown below.

| Supervisor queues | | | | User task queue* |
|---|---|---|---|---|
| I/O interrupt | Page manager | Supervisor fetch task | Transient | |
| 1 | 2 | 3 | 4 | $5_1\ 5_2 \cdots 5_m$ |

increasing dispatching priority $\longleftarrow$

*For the multitask case with one partition, tasks $5_1 \cdots 5_m$ have the same priority.

Figure 8 Queuing model of the CPU



The dispatching priorities shown are also valid if a request has been interrupted. For example, the fact that in DOS/VS an "interrupted" supervisor fetch task has higher dispatching priority than a new request for the page manager is taken into account by attaching the supervisor portion after SIO of the supervisor system task to the I/O interrupt service in the model (level 1).

It is obvious that a page manager request in the case of a serially reusable page manager can only be dispatched if the associated P gate is open. It must be further observed that page manager requests coming directly from queue 1 (rate $\lambda_{21}$) have higher priority than page manager requests coming directly from level 0 (rate $\lambda_{22}$). For this reason, in the case of a combined page-out/page-in, the serially reusable page manager is effectively locked until the page has been brought into real storage.

The only requests arriving asynchronously at the CPU are the I/O interrupt requests; therefore, service interruptions can only be caused by I/O interrupt requests.

The queuing analysis allows two different borderline cases of interruptibility. They are indicated by a logical variable *NI*:

$$NI = \begin{cases} 0 \text{ if I/O-interrupt requests may interrupt page manager} \\ \quad \text{and supervisor system tasks as well as transients} \\ 1 \text{ else (not interruptible)} \end{cases}$$

Table 2 Interrupt scheme

| *I/O interrupt requests interrupt class* i *service?* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $i$ | 0 | 1 | 2 | 3 | 4 | $5_1$ | $5_2$ | . . . | $5_m$ |
| $NI = 0$ | no | no | yes | yes | yes | yes | yes | . . . | yes |
| $NI = 1$ | no | no | no | no | no | yes | yes | . . . | yes |

Table 2 shows the interrupt scheme with $NI$. For DOS/VS the second case $NI = 1$ is most appropriate.

As shown earlier, for the dispatching of user tasks two possible cases have been assumed: (1) a multitask case with one partition and (2) a multipartition case with one job per partition.

At the end of each user task phase (having mean $h_{5i}$ for task $i$, each task branches in different directions according to calculated branching probabilities. If a task takes the end branch, it is assumed for the queuing model that the next task with the same characteristics runs in the same partition.

The queuing model for the CPU is very complex and has been further split up for the calculation of the waiting times into quasi-independent servers, in which all effects causing waiting times in a certain queue are reflected. This splitting was done by (a) defining resulting service times (increased by I/O interruptions or closed gates) and (b) splitting up waiting times into different components such that the total queuing structure of the CPU model is reflected in the approximations.

**CPU waiting times**

As a major part of the work, the CPU model will now be sketched. Only the most significant points of the queuing approximations method are sketched in the sequel.

The time a request spends in a queue up to the beginning of its service will be referred to as initial waiting time. If the service is allowed to be interrupted (once or several times), an additional subsequent waiting time occurs.

In principle each waiting time in a queue is subdivided (approximately) into four partial waiting times:

a. Initial waiting time due to an uninterruptible *lower-priority* service (greater than zero only for requests coming from outside the CPU).
b. Initial waiting time due to *higher-priority* requests already present at the CPU and arriving during the following time, which is necessary to serve these requests.

c. Initial waiting time due to requests of the *same priority* already present, taking into account that their service may be delayed by interruptions.
d. Subsequent waiting time due to service interruptions by I/O interrupt requests, possibly increased by higher-priority non-interrupting requests being dispatched subsequently (greater than zero only if service is interruptible).

The subdivision into these four parts allows the determination of the mean waiting times in the different queues. Depending on what the queue index $i$ is and on what the supervisor specification (gating and interrupt scheme) is, some of the partial waiting times may become zero.

These partial waiting times are shown in the Appendix for CPU levels 1, 2, and 5, including the complex queuing considerations. A simplified similar model with Poisson input is calculated exactly in Reference 6.

**channel device system**

A very large influence on system performance is imposed by the configuration of a computer system, i.e., the number, arrangement, and types of I/O devices. Within the scope of this paper, attention has been focused upon DASDs (direct access storage devices) because disks or drums are normally of major interest. Nevertheless, tapes can also be modeled, provided that proper input service-times are selected.

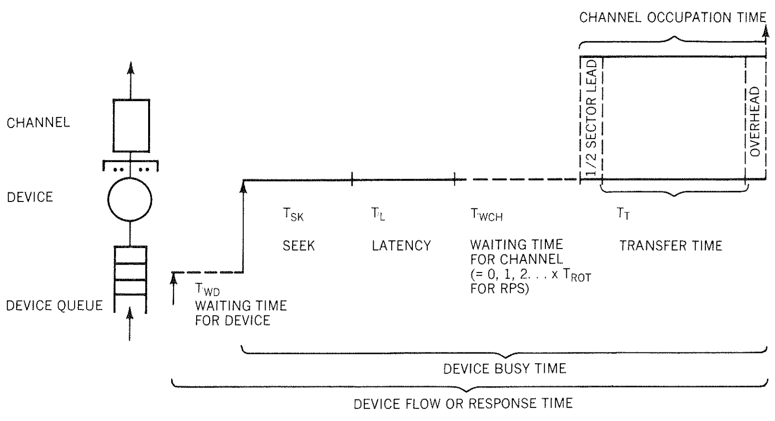
The three different types of I/O traffic (normal I/O, page I/O, and fetch I/O) are distributed according to arbitrary branching probabilities among the devices. Thus, the associated files may reside on arbitrary devices, e.g., the page data set is either only on one device or resides on several devices (split page data set). It is generally assumed that the devices are DASDs using the RPS (rotational position sensing) feature,[7] which is not further discussed here. The channels must therefore be of the block multiplexer type.

Figure 9 shows the principal occupation scheme assumed to calculate the waiting times in the channel device system. The fact that the channel must be available for seek initiation is ignored in the calculation and so is the existence of a control unit. Such simplifications are common and have little impact on the queuing results.

For each device attached to a certain channel, the following input data are needed individually:

| | |
|---|---|
| $T_{\mathrm{SK}}$ | mean seek time (equals 0 for fixed head devices, e.g., drums) |
| $T_{\mathrm{ROT}}$ | rotation time |

Figure 9 Principal occupation scheme



| | |
|---|---|
| $T_{\mathrm{SL}}$ | sector lead time (time during which device is ready for transfer) |
| $T_{\mathrm{T}}$ | transfer time (equals block size/transfer rate, possibly including control overhead) |

Note that it is possible to adjust $T_{\mathrm{SK}}$ and $T_{\mathrm{T}}$ to the individual workload, thus taking into account the specific cylinder locations and block sizes.

To calculate the response times of each device, their waiting times for channel service must be determined first. During this waiting time for its ''secondary resource'' (channel waiting time), the device is not available for other requests, which not only increases the device busy time but also the waiting time of the requests for the device in the device queue (device waiting time).

**channel waiting times**

To calculate channel waiting times let $\rho_{\mathrm{CH}j,i}$ be the partial channel utilization for channel $j$ induced by device $i$, which is zero if device $i$ is attached to another channel. It is obvious that

$$\rho_{\mathrm{CH}j} = \sum_i \rho_{\mathrm{CH}j,i}$$

is the total channel utilization of channel $j$. Since the device is a rotating one, transfers can start only when the desired sector is located below the read/write heads. The connection with the channel is attempted during the sector lead time. If it failed to seize the channel, it is said that an RPS miss occurred, and the same procedure is repeated after one revolution. Therefore, the channel waiting times are integer numbers of revolutions

$$T_{\mathrm{WCH}i} = K \cdot T_{\mathrm{ROT}i} \qquad K = 0, 1, 2, \cdots$$

The probability of an RPS miss for device $i$ attached to channel $j$ can be determined approximately by the probability that channel $j$ is occupied by another device at an arbitrary instant:

$$p_{\mathrm{RPS}i} \approx \frac{m-1}{m} \cdot \sum_{\nu \neq i} \rho_{\mathrm{CH}j,\,\nu}$$

The ratio $(m - 1)/m$ takes into account that the partial channel utilizations come only from other partitions/tasks, such that, for a single batch, $(m - 1)\ p_{\mathrm{RPS}} = 0$.

The mean channel waiting time for device $i$ turns out to be

$$E(T_{\mathrm{WCH}i}) = \frac{p_{\mathrm{RPS}i}}{1 - p_{\mathrm{RPS}i}} \cdot T_{\mathrm{ROT}i}$$

which is the expected number of RPS misses times the rotation time of the device.

**device waiting times**

To calculate the waiting time $T_{\mathrm{WD}}$ in the device queue, it is necessary to apply a machine-repair model. The influence of the channel waiting times then is taken into account by using the device occupation time as "service time" in the machine-repair model. The associated number of sources depends on the degree of multiprogramming, the specification of the system tasks, and the location of the files.

This principle is also applied in the (normal) case where the head queue feature (SVC15) for supervisor I/O (page/fetch I/O) is used. To apply machine-repair models for this head queue case, page/fetch and normal I/O operations have to be specified separately (population, rates) taking into account that page/fetch I/O has additional waits caused by normal I/O operations already occupying the device. This is the principal way to obtain device response times.

Together with the arbitrary splitting of normal/page/fetch I/O across several devices, the mean response times of the total channel device system for the three different types of I/O operations can be deduced in a simple way.

## Program features

**general remarks**

In this section the APL program DOSDP (DOS Design Program), which implements the operating system model with the help of the queuing analysis previously discussed, is described. The program consists of numerous APL functions, which also support the self-prompting input facility.

The run time requirement of the program is very low, since the effective decomposition method used and the additional approximation principles developed only require a small number of itera-

Table 3 Input data for APL program

| *Program division* | *Input parameters* |
|---|---|
| Supervisor | Supervisor path lengths (total of 12)<br>Supervisor instruction execution rate<br>Page manager gated/reentrant<br>Supervisor fetch task gated/reentrant<br>Interruptible/noninterruptible system tasks |
| Workload | Multipartition/multitasking case<br>Number of active tasks<br>User task times per partition<br>Time in terminal subsystem<br>Fraction of page faults causing page out<br>Fraction of transients running in LTA<br>Fraction of phases fetched running in user partition<br>Normal, fetch, and page I/O activity per partition<br>Percentage of I/O-compute overlap of same task |
| Channel device system | Number of DASDs<br>DASD timings (seek, rotation, sector lead, transfer)<br>Number and arrangement of channels<br>Distribution of disk accesses for normal, fetch, and page I/O |

tion steps (say, less than or equal to 10). When the input parameters were specified, the output of the final results started in most cases within less than one minute on the Boeblingen interactive APL system (then a System/360 Model 65). For each run, an average of 30 seconds of CPU time was necessary.

**main input parameters**

The input data for the APL program can be subdivided as shown in Table 3. For ease of input control, all these data are compiled in an input summary (Figure 10) together with some calculated values (service times, branching probabilities).

**main output parameters**

The output results of the program comprise the CPU-related data, channel device system-related data, and workload-related data.

For the *CPU* the utilizations are determined (problem state, supervisor state) as well as their subdivision into different partitions and supervisor services. For each CPU queue, mean waiting times and queue lengths are calculated.

The *channel device system* results comprise the access rate, utilization, RPS miss probability, mean waiting time, and mean queue length for each RPS device. For each channel the access rate, mean channel waiting time, and utilization are given. Channel device system response times for normal/page/fetch I/O accesses are also provided.

The *workload* results are mainly characterized by partial CPU utilizations in the problem state and the job run times or the teleprocessing transaction response times, respectively.

Figure 10 Input summary

```
DOSDP VERSION 01| 30| 76  INPUT SUMMARY (TIMES IN MSECS)

                    COMPLETELY ARBITRARY SAMPLE RUN

***SUPERVISOR
PMGR  TASK GATED,FRACTION  .5000 OF PAGEFAULTS CAUSING PAGEOUT
SUPVR TASK GATED,FRACTION  .5000 IN SINGLE TRANSIENT AREA
PMGR/SUPVR TASKS + TRANSIENTS RUNNING DISABLED

SUPERVISOR PATHLENGTHS
NORMAL-I/O (ZN0,ZN1)          :       400       400
PAGE-I/O (ZP0,ZP1,ZP22,ZP21):       100       400
FETCH-I/O (ZF0,ZF1,Z3,Z4)     :       100       400       400       400
OVERLAPPED PATHS (ZSIO)       :       100
TASK END (ZE0)                :       400
SUPERVISOR MIPS VALUE         : .1000

                         INTRPTS     PMGR      SUPVR     TRANS
CPU SERVICETIMES     :    4.0000    4.0000    4.0000    4.0000
                          NORMAL     PAGE      FETCH    TASK END
SUPERVISOR TIMES H0 :     4.0000    1.0000    1.0000    4.0000

I/O-OVERLAPPED SUPERVISOR TIME HSIO  :    1.0000

***WORKLOAD
3  PARTITIONS (TASKS) WITH DIFFERENT PRIORITIES

TOTAL TASK TIMES IN USER STATE
UNOV+OVERLAPPED :   17000.0000   21000.0000   24000.0000
?VERL FRACTIONS :        .1800        .2000        .0000

TASK PHASE SERVICE TIMES
UNOVERLAPPED     :       8.7071      10.4934      14.9906
OVERLAPPED       :       3.0600       4.2000        .0000

                              NORMAL     PAGE   FETCH-I/O   TASK END
NUMBER OF CYCLES        :   1000.0000  400.0000  200.0000    1.0000
BRANCHING PROBABILITIES:        .6246     .2498     .1249     .0006

FRACTION  .1000 OF FETCHES FOR PHASES IN USER PARTITION

***CHANNEL DEVICE SYSTEM CDS CONSISTS OF 4 DEVICES AND 1 CHANNELS
DEVICE                             1          2          3          4
DEVICE SEEK TIME          :    30.0000    30.0000    30.0000    30.0000
       ROTATION TIME      :    16.7000    16.7000    16.7000    16.7000
       SECTOR LEAD TIME:        1.0000     1.0000     1.0000     1.0000
       TRANSFER TIME      :     4.0000     4.0000     4.0000     4.0000

HEADQUEUE PRIORITY FOR PAGE/FETCH-I/O IN CDS

FRACTIONS INDICATING LOCATION OF DATA SETS
DEVICE           1       2       3       4
NORMAL-I/O    .4000   .3000   .2000   .1000
PAGE  -I/O    .0000   .0000   .0000  1.0000
FETCH -I/O    .2500   .3000   .2500   .2000
```

The full scope of the output results is shown in Figure 11, all times specified in milliseconds.

## Program and model validation

**validity check of queuing approximations**

Prior to making comparisons with measurements, the APL program had to be checked for logical consistency with respect to the waiting times in context with the queuing approximations. The check was done by investigating many special cases, some of them having degenerated to queuing systems with known exact solutions. These degenerated systems turned out to be single-server systems with feedback or normal closed networks (without priorities). For such systems the remaining errors in waiting times have been identified as being due to the decomposition principle.[5]

The only possibility for checking the quality of the queuing approximations for the more complex structures discussed here was

Figure 11 Output of results

```
**********************     CALCULATION RESULTS     **********************

***CPU
TOTAL CPU UTILIZATION   :      .6057    PROBLEM STATE   :        .3420
                                      SUPERVISOR STATE  :        .2637

                              INTRPTS       PMGR       SUPVR       TRANS       PROC
CPU UTILIZATIONS        :      .1207       .0402       .0134       .0121      .3420
    QUEUE LENGTHS       :      .0217       .2969       .0251       .0047      .2934
    WAITING TIMES       :      .7179     29.5059      7.4744      1.5536    10.9289
    ARRIVAL RATES       :      .0302       .0101       .0034       .0030      .0268

HIGHEST PRIORITY SUPERVISOR UTILIZATION :       .0772
FRACTION OF TIME  PAGE-GATE CLOSED      :       .6920
FRACTION OF TIME FETCH-GATE CLOSED      :       .2175
PMGR WAITING TIMES      : W21=        .3502   W22=    44.0838

JOB CPU UTILIZATIONS    :        .1050       .1204       .1167
    QUEUE LENGTHS       :        .0273       .0807       .1854
    QUEUE WAITING TIMES:        2.7571      8.7965     23.8180
    QUEUE ARRIVAL RATES:         .0099       .0092       .0078

***CHANNEL DEVICE SYSTEM CDS
DEVICE                              1           2           3           4
DEVICE RPS MISS PROB    :        .0761       .0812       .0874       .0598
       OCCUPATION TIMES:       44.2259     44.3257     44.4492     43.9115
       UTILIZATIONS     :        .3337       .2676       .1863       .5449
       QUEUE LENGTHS    :        .0994       .0610       .0288       .3044
       WAITING TIMES    :      13.1698     10.1054      6.8654     24.5276
       ARRIVAL RATES    :        .0075       .0060       .0042       .0124

     NORMAL-I/O WAITING:       13.5906     10.6495      7.3440     78.8303
 PAGE/FETCH-I/O WAITING:        9.8026      7.3852      4.9509     16.0428

CHANNEL                             1
CHANNEL UTILIZATIONS    :        .1358
        ARRIVAL RATES   :        .0302

                              NORMAL       PAGE       FETCH        -I/O
I/O RESPONSE TIMES      :     62.2520     59.9543     53.3613
    CYCLE TIMES         :     69.9699    146.7671     74.5518
    RATES               :       .0168       .0101       .0034

PAGE FAULT RATE         :       .0067

JOB SYSTEM TIMES        :       161945.0985    174474.2216    205723.6521


TOTAL NUMBER RESULTANT :        3.0004
NUMBER OF ITERATIONS    :       10
LAST RELATIVE CHANGE IN CDS RATES =   .000119
```

an extra simulation program. This program was written using the SIMPL/1 simulation language.

Numerous cases have been simulated, beginning with queuing models having only one type of I/O and ending with systems with all three types of I/O. For four of these general cases, Tables 4 and 5 show comparisons between calculation and simulation results. All four cases have a multiprogramming degree of five and three disks in the channel device system. They differ in gated or reentrant system tasks and disabled or enabled supervisor services (levels 2, 3, and 4).

Table 4 shows the total CPU utilization, the maximum error for all four cases being only 1.6 percent. This percentage is also a consequence of the careful channel device calculation, the results of which are omitted here. Queuing analysts know that total CPU utilization is relatively uncritical with respect to approximations of waiting times. Table 5 shows the CPU user utilizations for the five different partitions, having different dispatching priority. As

Table 4 Four calculation-simulation comparisons (total CPU utilizations and waiting times)

| *Case* | *Mean waiting times (milliseconds)* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Total CPU utilization* | | *I/O interrupt queue* | | *Page manager queue* | | *Supervisor fetch task queue* | | *Transient queue* | | *User queue* | |
| | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* |
| Gated/Disabled | 0.775 | 0.776 | 1.56 | 1.62 | 42.1 | 35.4 | 30.5 | 26.0 | 4.30 | 3.0 | 30.0 | 25.7 |
| Gated/Enabled | 0.777 | 0.790 | 0.91 | 1.03 | 42.4 | 39.5 | 30.8 | 29.7 | 5.35 | 4.94 | 30.1 | 28.1 |
| Reentrant/ Disabled | 0.811 | 0.800 | 1.63 | 1.78 | 0.44 | 0.40 | 0.44 | 0.60 | 1.80 | 1.7 | 27.2 | 27.8 |
| Reentrant/ Enabled | 0.812 | 0.805 | 0.95 | 1.09 | 1.32 | 1.50 | 1.18 | 1.25 | 2.93 | 3.5 | 27.3 | 28.3 |

Table 5 Four calculation-simulation comparisons (partition utilizations)

| *Case* | *CPU utilizations in problem state* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Partition 1* | | *Partition 2* | | *Partition 3* | | *Partition 4* | | *Partition 5* | | *All partitions* | |
| | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* | *Calc* | *Sim* |
| Gated/Disabled | 0.093 | 0.108 | 0.087 | 0.095 | 0.079 | 0.079 | 0.068 | 0.064 | 0.055 | 0.047 | 0.383 | 0.393 |
| Gated/Enabled | 0.093 | 0.106 | 0.087 | 0.093 | 0.079 | 0.078 | 0.068 | 0.063 | 0.055 | 0.051 | 0.384 | 0.391 |
| Reentrant/ Disabled | 0.098 | 0.109 | 0.092 | 0.094 | 0.083 | 0.078 | 0.071 | 0.063 | 0.056 | 0.050 | 0.411 | 0.395 |
| Reentrant/ Enabled | 0.098 | 0.109 | 0.092 | 0.095 | 0.083 | 0.078 | 0.071 | 0.063 | 0.056 | 0.050 | 0.411 | 0.385 |

can be seen, the error is typically less than 10 percent for the individual utilization. Many other comparisons confirm the error ranges indicated above.

**comparisons with measurements**

Because of the variety of input data for the program, a complete and profound comparison with DOS/VS measurements needs many input parameters, which are normally not available for one sample measurement run. However, to make statements about the applicability of the model, existing measurements have been taken.

The measurements were taken in August 1975 on a System/370 Model 125 with DOS/VS Release 32.[8] They include single and double batch runs for so-called VSAM (Virtual Storage Access Method) macro sequences (F1 and F2) which are not reproduced here in detail. The same applies to the APL program results.

Table 6 Comparisons of Measurements and Program Results

| | *F1 in foreground F2 in background* | | *F2 in foreground F1 in background* | |
|---|---|---|---|---|
| | *measured* | *calculated* | *measured* | *calculated* |
| Total CPU utilization | 0.92 | 0.91 | 0.92 | 0.926 |
| Supervisor state | n.a. | 0.38 | 0.44 | 0.412 |
| User state | n.a. | 0.53 | 0.48 | 0.514 |
| Throughput | | | | |
| F1 cycles/second | 4.4 | 4.77 | 2.62 | 2.61 |
| F2 cycles/second | 1.4 | 0.92 | 2.60 | 1.96 |

*Single* batch measurements can serve as additional help for calibration and verification of path lengths and of device service times. All data obtained from the single batch measurements have been consistent and confirmed the assumed Release 32 path lengths. Also, the difference in the MIPS values for supervisor and VSAM processing has been confirmed.

By using these calibrated values of the single-batch case for *double*-batch runs of the program, we obtained the results in Table 6. As can be seen in the table, the error in the partial CPU utilizations is less than seven percent.

The throughput results, from which the response times can be directly calculated, are fairly good for the F1 sequence, whereas the F2 throughput is lower in the calculation (irrespective of the partition used). The cause is the different locations of the files for F1 and F2 during the measurements, which is not reflected in the present version of the program.

## Performance results

**general remarks** In order to show the application of the program, two examples have been selected, one for batch, the other for a pure teleprocessing workload. These examples have been extracted from numerous runs.

It was also demonstrated elsewhere how the calibration of the model can be done, including paging behavior. Please note that the aim of this section is to show only the application of the program and not to claim that each input value is realistic in every case.

**triple batch**

Let a batch job be characterized by

- The total CPU time in user state ETUS
- The fraction of ETUS that overlaps with I/O activity for the same job, FOL
- The three-component vector ZIO indicating the number of normal I/Os, page faults, and transients fetched.

In order to cover a wide workload spectrum, 12 different sample jobs have been adopted. Each is characterized by three million user instructions, and runs on a CPU with 0.1 MIPS and a value of ETUS = 30 seconds. For reasons of simplicity, FOL = 0 has been adopted here.

By means of ZIO, different I/O intensities can be achieved. This has been done by varying the number ZIO (1) of normal I/Os:

ZIO (1) = 500, 1000, 2000 normal I/O accesses.

The number of page faults ZIO (2) has been selected to cover usual paging rates:

ZIO (2) = 0, 300, 600, 1200 page faults.

The third component ZIO (3) is the number of transients fetched, which has been fixed:

ZIO (3) = 200

For the resulting 12 sample jobs, 12 runs have been performed for a system with three active partitions (triple batch) with the same job in each partition. Furthermore, it was assumed that 50 percent of the page faults required a previous page out. The supervisor path lengths have been selected according to experience values, and the system tasks were specified as being gated.

Out of the vast variety of channel device systems, two identical disks have been chosen with a mean seek time of 20 milliseconds, a rotation time of 16.7 milliseconds, and a sector lead time of 0.874 millisecond. Both disks have been attached to the same block multiplexer channel, the transfer times being 2.48 milliseconds. The data sets have been located such that the page data set was fully on disk 2, all transients on disk 1, and the user (normal I/O) files split equally on both devices. Figure 12 shows the resulting CPU utilizations for the 12 runs (total and problem state part) as a function of the number of page faults of a job in one partition.

For the different types of I/O traffic, Figure 13 shows the cycle times, which have been defined as being the time a job is not in the dispatchable user task queue since it performed a normal I/O, or has produced a page fault, or initiated the fetching of a transient phase.

Figure 12 Triple-batch CPU utilizations (12 runs each with the same three jobs)

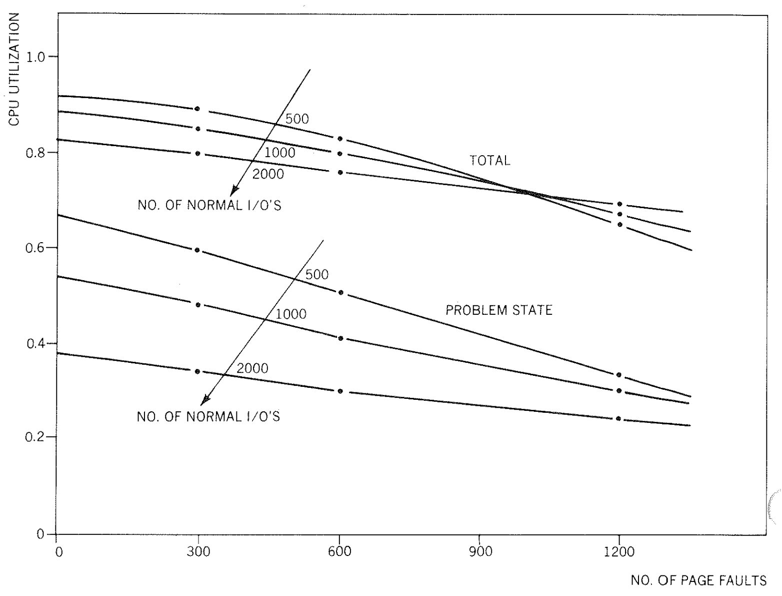
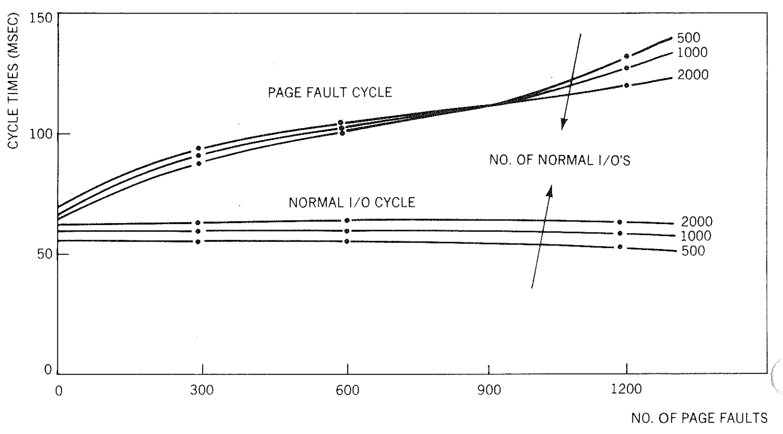


Figure 13 Normal I/O and page fault cycle times for triple-batch runs



Because of the head queue priority of page I/O accesses in the channel device system, the page fault cycle times do not depend so much on the number of normal I/O operations of the jobs. Note that for jobs with 1200 page faults the page fault cycle times decrease with an increasing number of normal I/O operations. This decrease is due to the decreasing paging rate which then relieves the page manager. The normal I/O cycle times remain nearly constant, though the normal I/O access rates to the disks are decreasing (due to the increasing page I/O having head queue priority).

Figure 14 Triple-batch prolongation factors for partition 1 and 3 (assimilation with increasing I/O activity)

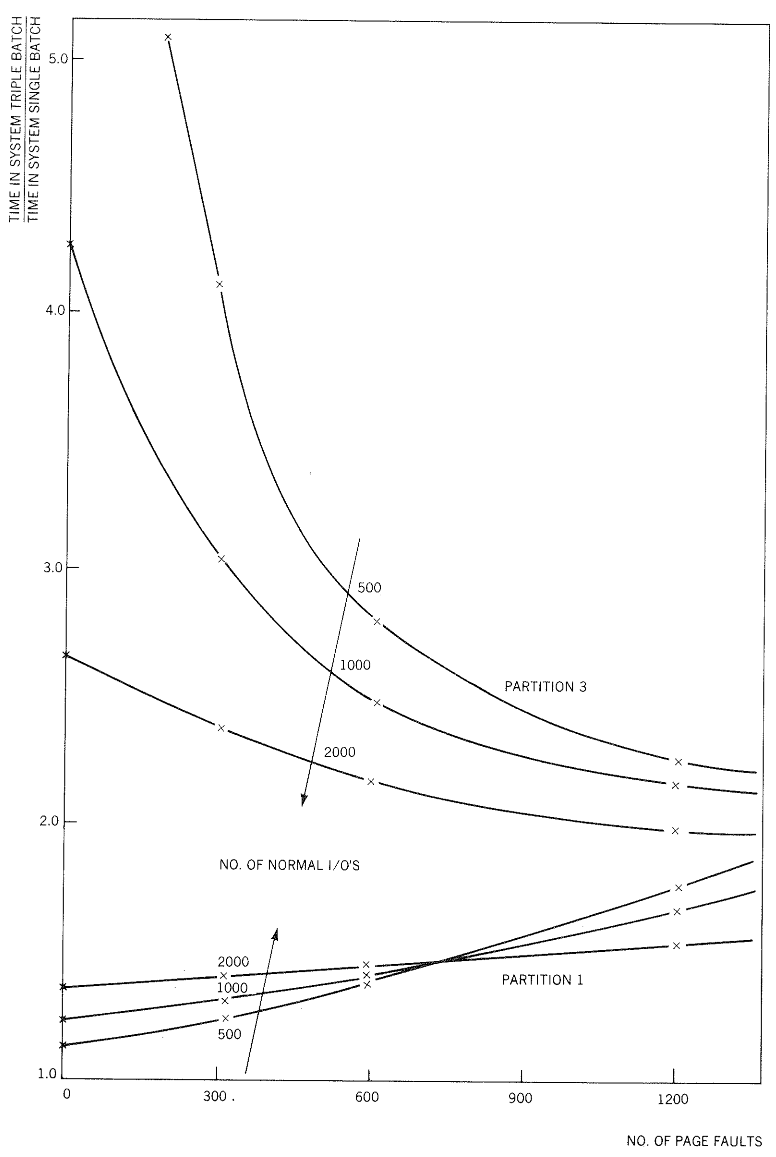
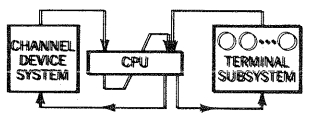


Figure 14 shows triple-batch prolongation factors for the job run times. With increasing I/O activity the effect of the preference given to the first partition diminishes remarkably since partition priorities are not considered for scheduling purposes in the channel device system.

**pure teleprocessing workload**

In the case of a pure teleprocessing application, it is often useful to assign the same priority to each user transaction (multitasking within one teleprocessing partition). Figure 15 schematically shows the teleprocessing configuration, paging paths being omitted.

Figure 15 Configuration scheme for teleprocessing



To define a user transaction, a mean number of, say, 70 000 instructions in the user state has been chosen. This value is arbitrarily chosen; real values would have to consider such items as VTAM, CICS, VSAM, and user code.

Let us assume that each transaction consists of six disk accesses (normal I/O), thus rendering a mean user task phase path length of 10 000 instructions, corresponding to 92.6 milliseconds on a 0.108 MIPS CPU. The supervisor path lengths have been chosen as in the previous example.

The channel device system consisted of three identical RPS disks (e.g., IBM 3330s) with a mean seek time of 30 milliseconds. It was assumed that the page data set is located on one disk, the normal I/O accesses again being split equally across all three disks (all of them attached to one common channel). Fetch I/O has not been considered in this example. Throughout all runs a terminal subsystem time of 15 seconds (user think time plus time for transfer has been assumed.

The number of active users or terminals has been varied from five up to 30, with four different cases:

- No paging
- With paging and gated page manager
- With paging and reentrant page manager
- With paging and gated page manager as above but without head queue priority for page I/O.

In the paging case, the mean number of page faults per transaction had to be determined as a function of the number of active users. This determination has been done with the help of (normalized) parachor curves, indicating the probability of a page fault as a function of the available real storage size.[9] Only the paging calibration result is shown here:

| number of active users | 5 | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|---|
| mean number of page faults | 0.5 | 1 | 2 | 4 | 8 | 16 |

Figure 16 shows the resulting CPU utilization (total and supervisor state) for these four different cases. For the same paging behavior of the transactions, the differences are obvious. The nonhead-queue case (no HQ) has been included only for comparison reasons.

Figure 17 shows the associated mean response times for the four different cases as a function of the number of terminals. These curves could be used to select the maximum number of terminals for prescribed response time requirements.

Figure 16 Pure teleprocessing CPU utilizations for four different paging cases

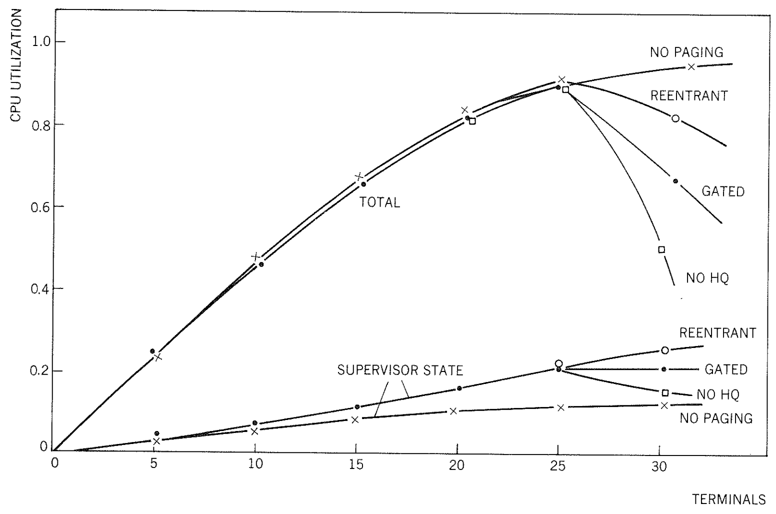


Figure 17 Mean response times for a transaction

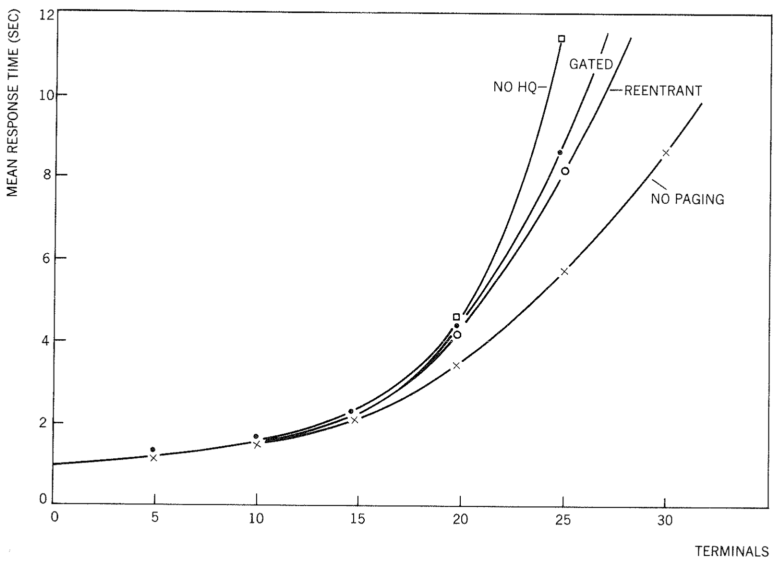


Figure 18 Resultant teleprocessing throughput rates

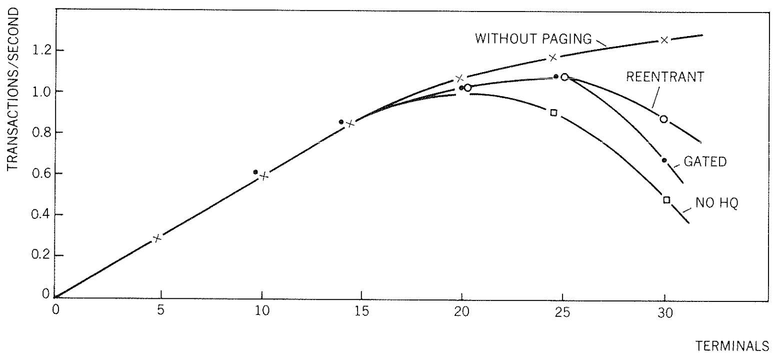


Figure 19 Mean number of transactions in CPU and channel device system

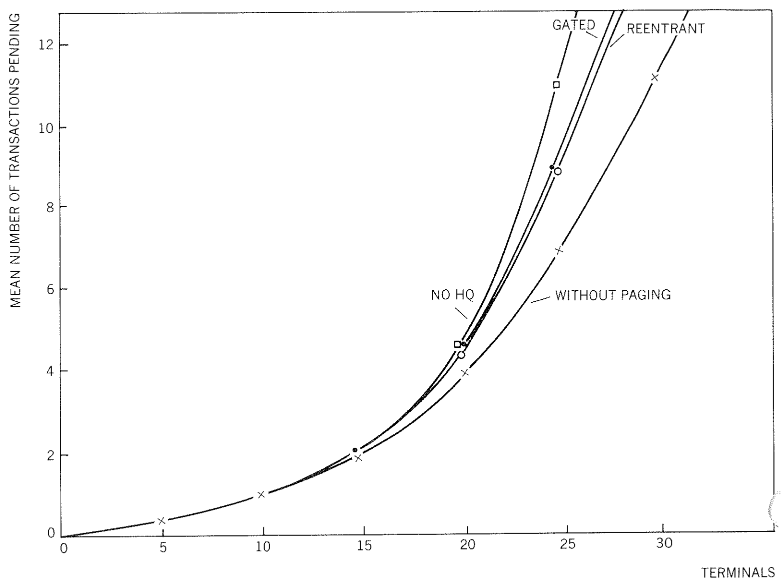


Figure 20 Fraction of time the gate of the serially reusable page manager is closed

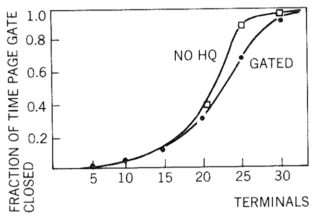
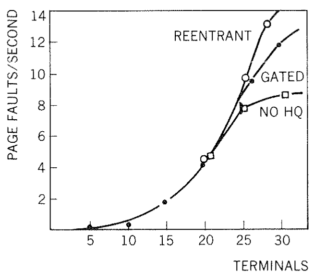


Figure 18 demonstrates the system throughput expressed as the number of transactions per second, growing linearly in the beginning and either tending to an upper value in the cases of no paging or eventually decreasing because of thrashing.

Figure 19 contains the mean number of transactions in the CPU and channel device system. This number corresponds to a medium degree of multiprogramming, which, apart from the number of terminals, has not been restricted further in the queuing calculations.

For the two cases where a gated page manager is used, Figure 20 shows the fraction of time with the gate of the page manager closed. This value indicates to what degree the paging capacity of a system is utilized.

Figure 21 Page fault rates managed



Finally, Figure 21 depicts the page fault rate managed by the different page managers.

## Summary

In this paper the development of a DOS/VS-based operating system model that serves as a basis for performance predictions has been demonstrated. The model has been developed on a level that is neither too gross nor too detailed with respect to the desired performance results. Nevertheless, it reflects those hardware and software components that may have a significant influence on system performance.

The approximate and iterative queuing analysis, on which the model implementation in APL is based, has been sketched. This APL program can be characterized by a flexible description of the supervisor, the workload, and the channel device system and gives quick and detailed answers to performance questions, although the analytic model only applies to stationary and randomized processes.

Nevertheless, the model in Figure 1 can be used as a basis for an even more detailed simulation program in which inhomogeneous task behavior (e.g., different job steps, or job control) can also be considered explicitly.

ACKNOWLEDGMENTS

The work being reported here was done at the IBM Boeblingen Laboratories as part of a 15-month Post-Doctoral Fellowship that ended in March 1976.

The stimulae to develop and analyze the model as well as continuous management support have been provided by K. Roehr, K. M. Schulz, W. G. Spruth, and E. Vogt. Many system aspects have been discussed and clarified with V. Degering and K. F. Finkemeyer. Special acknowledgment is given to U. Steinemann for simulation. The author also thanks several anonymous referees for their valuable criticism and many helpful comments.

## Appendix: CPU waiting times

Before describing the partial CPU waiting times (defined in the section on CPU waiting times) let us summarize some notations used:

| | |
|---|---|
| $h_i$ | = mean service time for queue $i (i = 0 \cdots 5)$ |
| $E(T_{wi})$ | = total expected waiting time of a request in queue $i$ $(E(T_{w0}) = 0)$ |
| $\rho_i$ | = $\lambda_i \cdot h_i$ = CPU utilization of level $i$ |
| $MR(m, \lambda, h)$ | = mean waiting time in a single-server machine-repair model with $m$ sources, throughput rate $\lambda$, and mean service time $h$ |
| FAC | = $(m - 1)/m$ |
| $\rho_i^*$ | = FAC $\cdot \rho_i$ = effective (reduced) utilization of level $i$ |

**I/O interrupts (queue 1)**

The mean waiting time for the I/O interrupts in queue 1 is

$$E(T_{w1}) = \underbrace{\rho_0^* \cdot h_0 + NI(\rho_2^* \cdot h_2 + \rho_3^* \cdot h_3 + \rho_4^* \cdot h_4)}_{a}$$

$$+ \underbrace{0}_{b} + \underbrace{\mathrm{MR}(\mathrm{MI}, \lambda_1, h_1)}_{c} + \underbrace{0}_{d}$$

where the designated groups of terms mean the following:

a is the expected rest-service time of a noninterruptible supervisor service. The weighting factors of this sum are the effective utilizations. The individual rest-service times here are set to the mean service times, since negative exponentially distributed service times are assumed.

b is zero since there is no higher-priority queue.

c is the mean waiting time in an equivalent machine-repair model with MI sources. MI depends on supervisor specification and the channel device system.

d is zero since level 1 service is never interrupted.

For queues 2, 3, and 4 the calculation must consider the reentrant and the gated case. As an example, the page manager waiting times are demonstrated.

**page manager requests (queue 2)**

For queue 2, there are the following page manager requests:

*Page Manager Reentrant*:

1. Requests coming from queue 1 (queue $2_1$) are specified as follows:

$$E(T_{W21}) = \underbrace{0}_{a} + \underbrace{(1 - NI) \cdot \rho_{22}{}^{*} h_2 + \frac{\rho_1{}^{*}}{1 - \rho_1{}^{*}} \cdot h_1}_{b}$$

$$+ \underbrace{\mathrm{MR}\left(m, \lambda_{21}, \frac{h_2}{1 - \rho_1{}^{*}}\right)}_{c} + \underbrace{(1 - NI) \cdot \frac{\rho_1{}^{*}}{1 - \rho_1{}^{*}} \cdot h_2}_{d}$$

where

a is zero, since these requests are created by the CPU itself, and therefore the CPU cannot be occupied otherwise at this moment.

b is the initial wait due to an interrupted page manager request being dispatched first and due to other I/O interrupt requests left behind in queue 1 which are also selected first.

c is the mean waiting time in an equivalent machine-repair model, where $h_2/(1 - \rho_1{}^{*})$ is an extended service time seen by a request that only sees the same priority predecessors. It does not necessarily mean that interruptions are allowed.

d is the mean time the request waits because it is interrupted by I/O interrupt requests (subsequent waiting time).

2. Requests coming from queue 5 (queue $2_2$) are specified as follows:

$$E(T_{W22}) = \underbrace{0}_{a} + \underbrace{\frac{\rho^*}{1-\rho^*} \cdot h_0}_{b} + \underbrace{0}_{c} + \underbrace{(1 - NI) \cdot \frac{\rho_1^*}{1-\rho_1^*} \cdot h_2}_{d}$$

where

b is the initial wait caused by requests that arrived in queue 1 while the request was in the noninterruptible service at level 0. This expression can be deduced from the fact that during the mean time $h_0$, $\lambda_1 \cdot h_0$ class 1 requests arrive. Each of these arriving requests requires service by level 1 and possibly level $2_1$; therefore, $\rho^* = \rho_1^* + \rho_{21}^*$. When these requests have been served, further class 1 requests may be present, and so forth, resulting in this sum of a geometric series.

c is zero. There are never several requests waiting, since the page manager is reentrant in this case.

*Page Manager Gated:*
The main differences in the calculation of the waiting time components for the gated case versus the reentrant case consist of two points:

1. The population in a queue may degenerate to one due to the gating mechanism. Here, part c for queue $2_1$ will become zero.
2. The gates are taken into account by replacing the service times in the machine-repair model by the times the gates are closed. These times are also the result of the iterative calculations.

**user tasks (queue 5)**

There are two cases handled by the program:

1. Multitask case within one partition.
2. Multipartition case with one task (job) per partition.

Since user tasks only are dispatchable as a consequence of I/O interruptions and, in addition, are running with lowest dispatching priority, part a of the waiting time is always zero.

The multitask case is characterized by $m$ user tasks of the same dispatching priority.

$E(T_{W5}) =$

$$\underbrace{\frac{\text{FAC} \cdot \left\{ \sum_{i=1}^{4} \lambda_i \cdot [E(T_{Wi}) + h_i] \cdot h_i + \rho_0 \cdot (h_0 + p_p \cdot h_2 + p_f \cdot h_3) \right\}}{1-\rho^*}}_{\text{b}}$$

$$+ \underbrace{\text{MR}(m, \lambda_5, h^*)}_{\text{c}} + \underbrace{\frac{\rho^*}{1-\rho^*} h_5}_{\text{d}}$$

where

b is a term in which the numerator is the total amount of higher-priority work to be done by the CPU at an arbitrary instant. During this time for the "initial amount of work," further requests may arrive from outside the CPU. This is taken into account by

$1/(1 - \rho^*)$

where essentially

$\rho^* = \text{FAC} \cdot (\rho_1 + \rho_{21} + \rho_4)$

c represents the initial waiting time due to the same class of requests and is determined via machine repair with

$$h^* = h_0 + \frac{h_5 + p_p \cdot h_2 + p_f \cdot h_3}{1 - \rho^*}$$

as an effective service time.

In the multipartition case the waiting time calculation is influenced by the fact that dispatchable higher-priority user tasks may increase the waiting time of a job in partition $i$ and in an extreme case increase it even to an infinite value. The increase depends on the partition index $i$. This leads to a more complex calculation for part b, whereas part c will become zero. In the section on validation, these approximations are compared with simulation results.

## CITED REFERENCES AND NOTE

1. The model itself is not dedicated to a specific DOS/VS release, though calibration normally depends on features and path lengths of an individual release and on data specified for system generation. Therefore, this model is not intended to be used as a reference to the latest available improvements of DOS/VS.
2. *IBM System/370 Principles of Operation*, GA22-7000, IBM Corporation, Data Processing Division, White Plains, NY 10604.
3. J. P. Birch, "Functional structure of IBM virtual storage operating systems, Part III, Architecture and design of DOS/VS," *IBM Systems Journal* **12,** No. 4, 401–411 (1973).
4. *DOS/VS Supervisor Logic*, SY33-8551, IBM Corporation, Data Processing Division, White Plains, NY 10604.
5. J. H. Florkowski, *Extended Analytic Models for Systems Evaluation*, Technical Report TR 00.2549, IBM Corporation, Poughkeepsie, NY (August 1974). (ITIRC 74A03196.)

6. W. Kraemer, "Processor priority model with different user tasks and operating system phases," *Proceedings of the International Symposium on Computer Performance, Modeling, Measurement, and Evaluation*, Yorktown Heights, NY (August 1977), pp. 305–325.
7. H. Katzan, Jr., *Computer Organization and the System/370*, Van Nostrand Reinhold Company, New York (1971).
8. H. E. Kaller, Private Communication, IBM Corporation, Boeblingen, Germany.
9. L. A. Belady and C. J. Kuehner, "Dynamic space sharing in computer systems," *Communications of the ACM* **12,** No. 5, 282–288 (1969).

## GENERAL REFERENCES

H. A. Anderson, Jr., M. Reiser, and G. L. Galati, "Tuning a virtual storage system," *IBM Systems Journal* **14,** No. 3, 246–263 (1975).

Y. Bard, "Performance analysis of virtual memory time-sharing systems," *IBM Systems Journal* **14,** No. 4, 366–384 (1975).

Reprint Order No. G321-5082.