[1]

THE HITAC 5020 TIME SHARING SYSTEM

Reprinted from—
Proc. ACM. 24th, Nat.
Conf., 1969.

August, 1969

HITACHI

THE HITAC5020 TIME SHARING SYSTEM

by Shigeru Motobayashi, Takashi Masuda,
and Nobumasa Takahashi
Hitachi Central Research Laboratory
Tokyo, Japan

Summary

HITAC5020 time sharing system with a two-dimensional addressing feature was developed to accomplish the following: 1) to establish a design for a time sharing system with a two-dimensional addressing scheme, 2) to establish the structure of the required file system, 3) to explain the intrinsic nature of the man-machine interaction, 4) to expedite studies for productivity improvement of software. This type of time sharing system has great merit in comparison with conventional time sharing systems and will likely be the basis for many future large scale information processing systems. In this paper, we first describe the segmentation and paging mechanism; then we discuss some important parts of the supervisor which are characteristic of two-dimensional addressing, especially scheduling and swapping, dynamic linking, and how to process common segments.

Introduction

It is expected that the information industry of the future will progress towards centralized systems. Typical of this trend are the time-sharing systems of large-scale computers. Project MAC of M. I. T. (Massachusetts Institute of Technology) published in 1965 the MULTICS (Multiplexed Information and Computing Service) system. It is based on the experience obtained with the CTSS (Compatible Time Sharing System), and it incorporates an "information service" which will characterize the information industry of the future. In this system, the concept of hardware segmentation was first introduced. It allowed an extention of regions for existence of programs to a two-dimensional system of segments and locations.
The HITAC5020 time-sharing system (hereinafter abbreviated as 5020 TSS) was developed with an aim of accomplishing the following by experimental implementation of such two-dimensional addressing as were already introduced into the MULTICS system.

(1) To establish the design (including hardware and software) for the time-sharing system which uses two-dimensional addressing.

(2) To establish the structure of file system.

(3) To explain the intrinsic nature of the man-machine interaction.

(4) To expedite studies for productivity improvement of software.

This report gives the pertinent system configuration and hardware mechanism and some important characteristic properties of the supervisor accompanying the use of two-dimensional addressing.

Outline of the 5020 TSS

System Configuration

Fig. 1 shows the system configuration. The TRC (Transmission and Reception Controller) is a computer which controls transmission and reception of information to and from many terminal units connected through communication lines to the system. It shares the main memory with the 5020 CPU. Both computers are able to interact with each other by mutual interrupts. The 5020 CPU is connected through channels with various input and output units; the disc is used for file storage, and the drum is used for program swapping. The 5020 CPU is augmented by the DAT (Dynamic Address Translator), a device which plays an important role in the 5020 TSS for realization of two dimensional addressing.

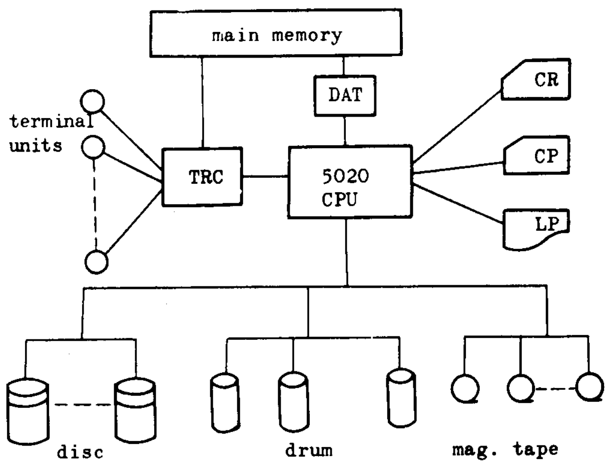

Fig. 1. Hardware Configuration

Hardware Features

Novel functions of the TRC and the DAT as part of the 5020 TSS are as follows.

(1) TRC

The TRC controls transmission and reception of input and output information between the terminal users and the 5020 CPU. At present, it is designed to accomodate up to 32 lines. Outstanding features of this unit are:

(a) Its hardware provides for parallel control of input or output from many input or output lines.

(b) It is a stored program computer sharing the main memory with the 5020 CPU. Its command system is constructed with a subset of the 5020 CPU.

(c) The TRC and the 5020 CPU have mutual interrupt capability. Transfer of information between both is accomplished through the main memory by control of interrupts.

(2) DAT

The DAT is a device incorporated in the 5020 TSS to realize the desired two-dimensional addressing capability. In this two-dimensional addressing scheme, each word in a program is specified by means of a two-dimensional logical address which consists of a segment name and a location in the segment. When an access to the memory is made, the logical address is translated into an absolute address in the main memory by the DAT through mapping tables. Mapping tables used for the address translation are prepared and controlled dynamically by the supervisor.
The DAT also provides a paging function. Each program or data set is divided into units each being called a page. Pages are of constant length to facilitate control of the memory and also to permit relocation by pages. The Mapping table itself is also paged. In the 5020 TSS, to avoid complexity of the whole system, each page has a fixed size of 256 words.
Mapping tables have three echelons: SPDT (Segment Page Descriptor Table), SDT (Segment Descriptor Table) and LPDT (Location Page Descriptor Table).
The SDT is a set of segment descriptors each of which defines the segment. The LPDT is a set of page descriptors each of which defines the page of individual segments. Each segment descriptor points to the heading address of the LPDT corresponding to the segment. Each page descriptor points to the heading address of the corresponding page. The SPDT is a set of descriptors each of which points to the heading address of each page of the SDT when it is paged.
The address space formed by mapping tables in this way is determined uniquely by the DBR (Descriptor Base Register) which points to the heading address of the SPDT (Fig. 2). By replacing the contents of the DBR, process switching is accomplished.
The translation of two-dimensional logical addresses into absolute addresses in the main memory through mapping tables is illustrated in Fig. 3.

To avoid useless repetition of the address translation by mapping tables at each memory reference, a correspondence table is placed in the associative register whenever a page in the segment is referenced. An entry of this table includes the segment number and the page number (upper 24-bits of the logical address). It indicates also the heading address of that page (contents of the page descriptor), so that, upon memory reference to the same page, the absolute address can be obtained immediately without resorting to mapping tables. Three types of base registers storing segment numbers are used to obtain logical addresses on which memory reference should be based for read-out of an instruction or reference to data during execution of a program.

(a) PBR --- To store the segment number of the program presently in execution

(b) ABR1, ABR2 --- To store the segment number to which memory reference is made as data reference

(c) JBR --- To store the segment number as destination for jump instructions

The logical address for instruction read-out is obtained by combining the contents of the PBR with the contents of the sequential control counter which controls locations in the segment presently in execution. The logical address for data reference by an instruction is obtained by combining the contents of the ABR with the address part of the instruction. The logical address as the destination for a jump instruction is obtained by combining the contents of the JBR with the address part of the jump instruction

(3) Memory protection

The protection of memory in the 5020 TSS is based on the so-called ring protection mechanism. It differs from the conventional method in terms of its range in core memory and protection for each logical segment is possible. The protection information has a 2-bit access control, a 4-bit lock in the segment descriptor of each segment, and a 4-bit key in the DBR which defines the address space of each process; these elements are combined together to provide necessary protection. The access control describes properties of individual segments in four types: read-execute possible, read-write possible, all possible, and read only. The key and lock are used to apply protection, when segments are grouped according to their importance, in accordance with the relative importance of the group to which the referencing segment belongs and the group to which the referencing segment belongs.
The protective mechanism as a combination of the access control and the key and lock is illustrated as shown in Fig. 4.

In Fig. 4, a mark O shows that an access is allowed. In the case of ×, an interrupt occurs at the access and control is transfered to the supervisor. If key = 0, every access is allowed irrespective of the access control and the lock. This is applied to the supervisor, and yet only in its vital part. If key $\leq$ lock, an access matching the access control setting is allowed. If key $>$ lock,

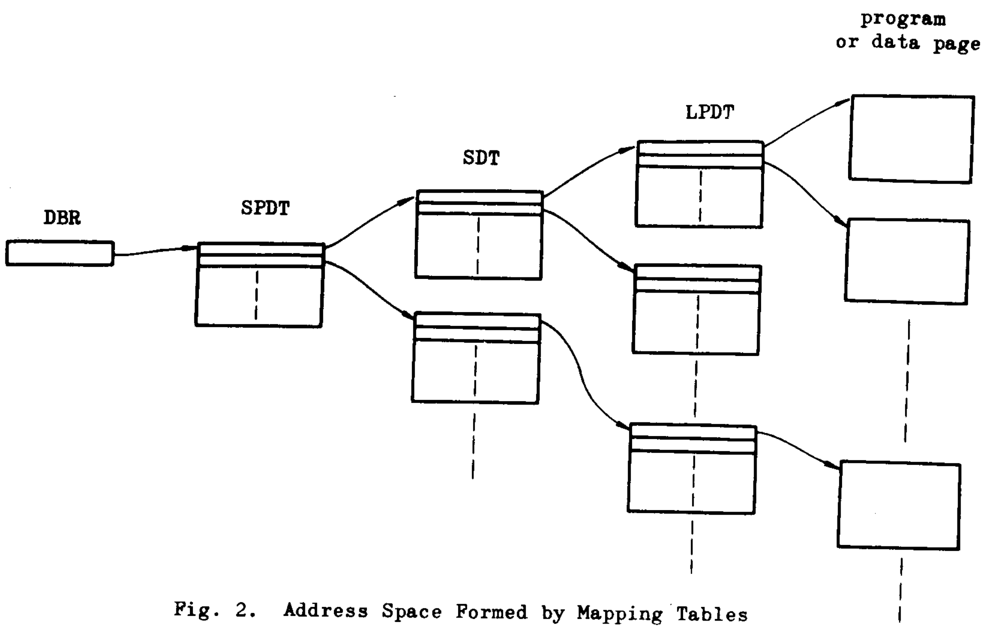

Fig. 2. Address Space Formed by Mapping Tables

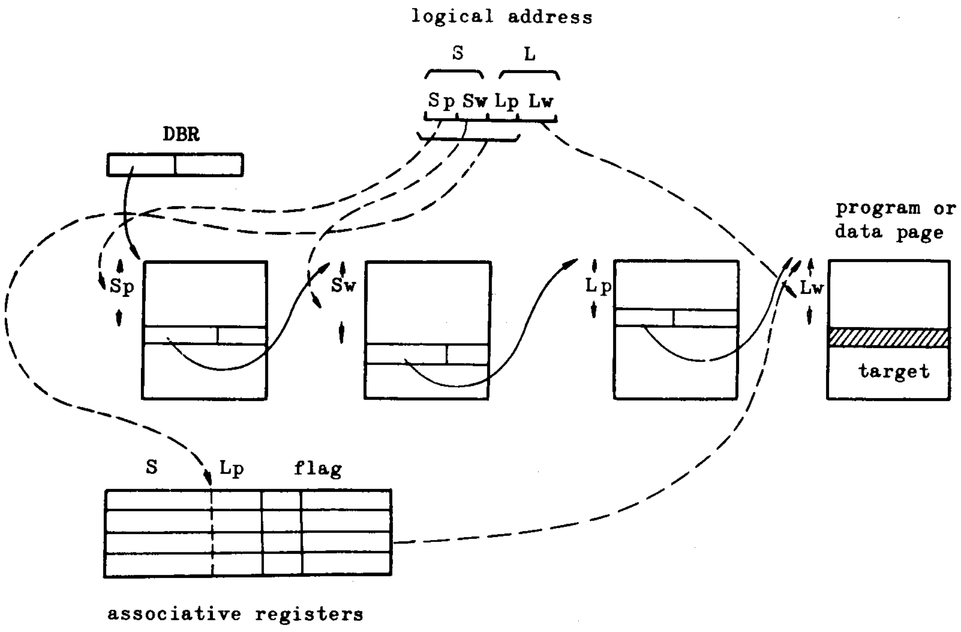

Fig. 3. Address Translation Mechanism by DAT

read only is allowed; write is never allowed. An attempted jump to any segment having a lock dissimilar to the key causes an interrupt.

Supervisor (Especially in Relation to The Segment Mechanism)

Swapping between Main Memory and Secondary Memory

If the page referenced in the course of address translation through mapping tables is not in the main memory, an interrupt occurs. This interrupt is called missing-page-fault and it can occur while a specific bit in each descriptor is

access control	key=0			key < lock			key=lock			key > lock		
	R	W	E	R	W	E	R	W	E	R	W	E
0 0	○	○	○	○	○	×	○	○	○	○	×	×
0 1	○	○	○	○	×	×	○	×	○	○	×	×
1 0	○	○	○	○	○	×	○	○	×	○	×	×
1 1	○	○	○	○	×	×	○	×	×	○	×	×

R -- read W -- write E -- execute

Fig. 4. Protection Mechanism of DAT

on.
In the case of the 5020 TSS, if a certain page is not in the main memory, the page address of that page in the secondary memory is set in the corresponding descriptor. (Fig. 5)
This means that information at swapping time is saved and it guarantees unified treatment of swapping. If the main memory has no idle space and requires some page to be swapped out, that page is brought to an idle space on the drum, the page-not-in-core-fault bit is set in the corresponding page descriptor, and the page address on the drum is also set. Then, when the page is referenced and a missing-page-fault interrupt occurs, swapping from the page address on the drum in the descriptor to the idle page on the main memory is provided for.
The swapping procedure just mentioned is applied between the main memory and the drum. In some cases, however, if the referenced page is not in the main memory, the page address on a disc may be set in the descriptor. This procedure is applied for loading a program or data first from a disc. The disc stores a file page table for each file which references addresses on the disc for each page. At the time when a segment is assigned and the segment descriptor is newly registered, the address on the disc of the corresponding file page table is also recorded. Since an actual reference to the segment is now made, a missing-page-fault interrupt occurs in the segment descriptor and the file page table is loaded as LPDT of the segment. By another reference, the desired page is loaded from the disc. Thus, unified treatment of swapping also applies to the loading of files.
The paging and missing-page-fault functions described so far have the following advantages:

(1) Relocation in units of pages is possible.

(2) Large programs can be prepared without regard to the capacity of the main memory.

(3) The main memory is used to store only those pages which are actually referenced. Effective utilization of the main memory is thus realizable and unnecessary swapping can be avoided.

(4) Since the main memory and the secondary memory can be controlled in units of pages, memory allotment and control are greatly simplified.

Reentrant Programs

In the case of multi-programming with a number of users effective utilization of the main

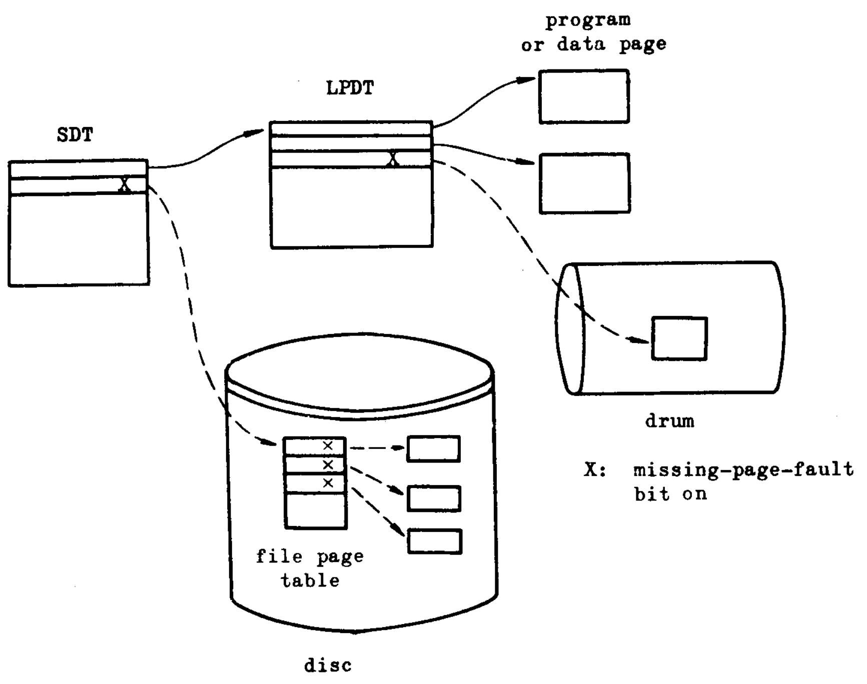

Fig. 5. Swapping Between Main Memory and Secondary Memory

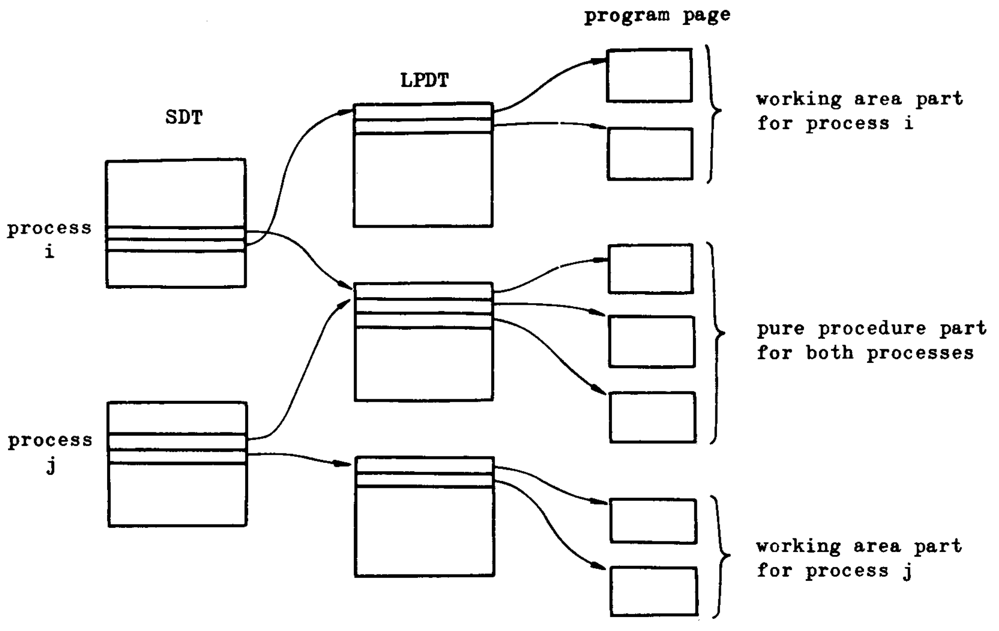
program page
LPDT
SDT
working area part for process i
process i
pure procedure part for both processes
process j
working area part for process j

Fig. 6. Relation Between Pure Procedure and its Working Area

memory is most desirable. When a program is already in use by one user, the same program can be called and shared by another without unnecessary copying. The program must be written in reentrant format; It consists of a pure procedure part which does not vary during program execution and a working area part which is affected by the execution. The working area parts are assigned separately to each user but the pure procedure part is in common use. The concept of segments, as applied to such reentrant programs simplifies processing to a great extent.

As shown in Fig. 6, the pure procedure part has different segment numbers assigned to different users, but is connected to a common LPDT so that it can be shared by all users. The working area part is assigned as a separate segment in each address space.

When a reentrant program is executed, segment numbers in each address space are registered on the aforementioned segment number storing base registers in the calling sequence (segment number for the pure procedure part in the PBR, and segment number for the working area part in the ABR); as the program operates within each address space, the working area part is automatically selected for each specific user.

In the 5020 TSS, all parts of the operating system, such as supervisor, command, library, etc. are stored in this reentrant format.

Dynamic Linking

In the conventional system, a given program and various subprograms called from within it are linked together, before being executed, by a linkage-editor and the whole program is then loaded into main memory for execution. This process is called static linking. In contrast, the 5020 TSS employs dynamic linking; The programs necessary for a given process are dynamically loaded in units of files from the file system as needed during program execution; They are then assigned appropriate segment numbers and linked.

Units of programs or data assigned to each user are registered as named files on the file system. When a symbolic call from one program file to another, or symbolic reference to a data file is made, or even if assignment of working segments is required during execution, the object program of the compiler or assembler refers to an area called linkage section in the working area part of the program and indirect jumps or references from the procedure part to the linkage section are made.

The form of the linkage section is shown in Fig. 7. The area enclosed within $<\ >$ in the same figure shows that the information shown is to be recorded there by the compiler or assembler.

Let us consider a symbolic call from program A to another program B registered on the file system. In the working area part of A, a linkage section (see Fig. 7) is set up, and "B" is recorded in the area "symbolic name". After an appropriate calling sequence, a call for an indirect jump to α in the linkage section (Fig. 7) is formulated. If the symbolic call references any parameters, an area in which the addresses of these parameters must be stored is developed, two words at a time, below the area "control information" of Fig. 7. For symbolic references to

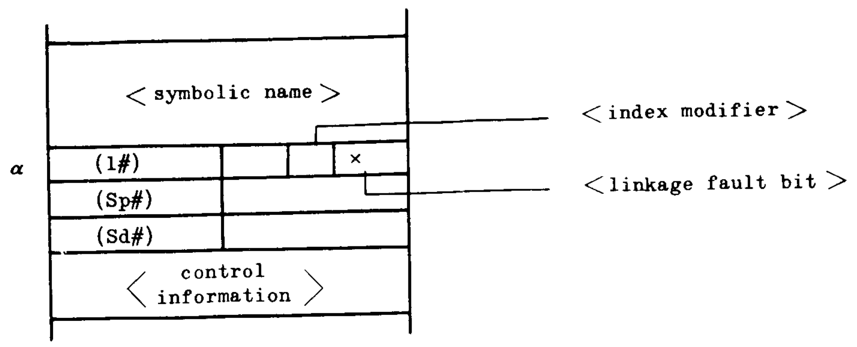

Fig. 7. Form of Linkage Section

other segments, a similar linkage section is set up.
When execution begins and an address α is referenced indirectly, a linkage fault interrupt occurs for the first reference, because linkage-fault bit of Fig. 7 is on. Then control is transfered to the supervisor called BFS (Basic File System). In the area "control information" of the linkage section, the various pieces of information for the BFS are recorded. Upon transfer of the control to the BFS, it looks for one of the following:

(1) Jumps to a program file,
(2) References to a data file,
(3) Assignment of a working segment.

In the first two cases, the file system is searched according to the given rules until the file being sought is found. (The present example is the case of a jump to a program file.) The file is then assigned a segment number and it is recorded as $S_p\#$ in the linkage section. In $S_d\#$, a segment number for the working area part corresponding to the program file is recorded. l# shows a location (entry point) in the segment of the assigned file and is obtained as a result of the file search. At the same time, segment descriptors corresponding to the two segments just assigned are recorded on the SDT.
If the file being called is already an assigned segment, the file search need not be made, but only the segment numbers will be stored in the appropriate columns of the linkage section.
As to working segment assignments, it is only necessary to assign one segment and store the segment number in the linkage section. After the segment assignment by this procedure, linkage-fault bit is turned off and control returns to the address which has caused the fault. In turn, an automatic jump (or reference) to $< S_p\#,\ l\# >$ is made. For the second and subsequent references, already assigned segments are referenced automatically.

Common Segments

As earlier mentioned, many programs of the 5020 TSS use common segments. Segment assignment, swapping and deletion of common segments are made as follows. In the construction of common segments by mapping tables, common use of LPDT is applied to general common segments, but a structure as shown in Fig. 8 is employed to process special common segments.
The common segment procedure, shown in Fig. 8, has a feature which assigns for each process same segment number and the SDT is used also in common. Certain resident common segments are always used for all processes, such as supervisor programs MCP (Master Control Program), BFS (Basic File System), TIO (Terminal I/O processor), CI (Command Interpreter), etc. In these cases, the principle shown in Fig. 8 have a number of advantages which include sayings in occupied areas of mapping tables through common use of the SDT and residence of LPDT's for segments just long enough for individual segments. For this type of common segments, fixed segment numbers are assigned in the acsending order, beginning with the smallest number.
Common use of the LPDT, as shown earlier in Fig. 6, is the form of general common segments. As the number of users and their assigned segment numbers increase, a need occurs frequently for swapping of programs and mapping tables of common segments. In this case the system does not allow swapping any common segment without regard to connectivities with other processes; the same applies to the deletion of segments.
As a solution to these problems of common segments the ACFT (Active Common File Table) is built up to store information on active common segments. Furthermore, each segment is referenced by two words of the segment descriptor (this means that all actual segment numbers are even). The second of these two words is used as software control information and in the case of a common segment, a pointer to the corresponding entry in the ACFT is provided. (Fig. 9)
All the segment descriptors pointing to same common segment are thus linked with the corresponding entry in the ACFT and the status of individual common segments can be controlled through information contained in the ACFT.
The segment descriptor mechanism operates as follows: If the corresponding LPDT is in core, the first word points to the heading address of that LPDT. If the corresponding LPDT is not in core, the address for the secondary memory of the

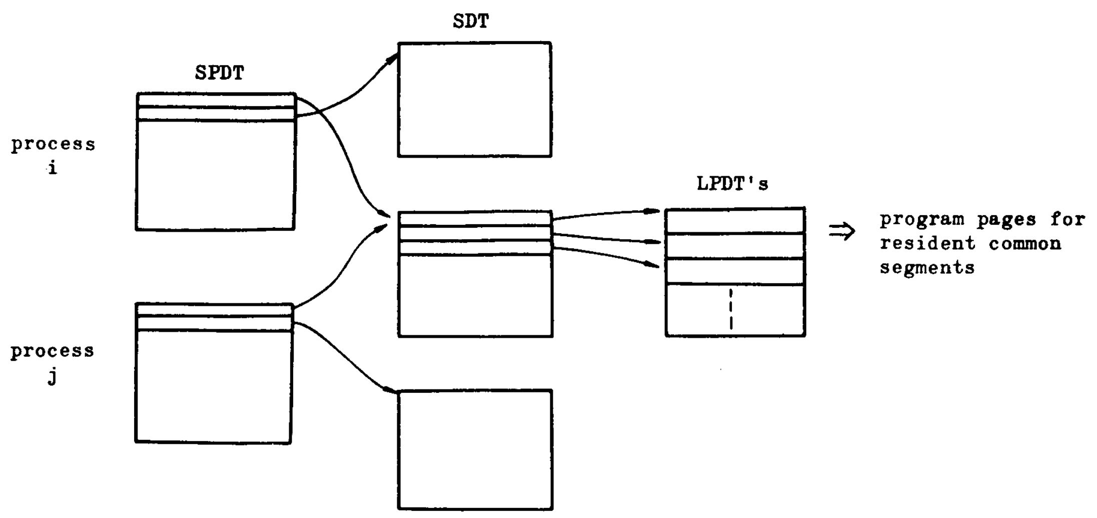

Fig. 8. Wired-down Common Segment

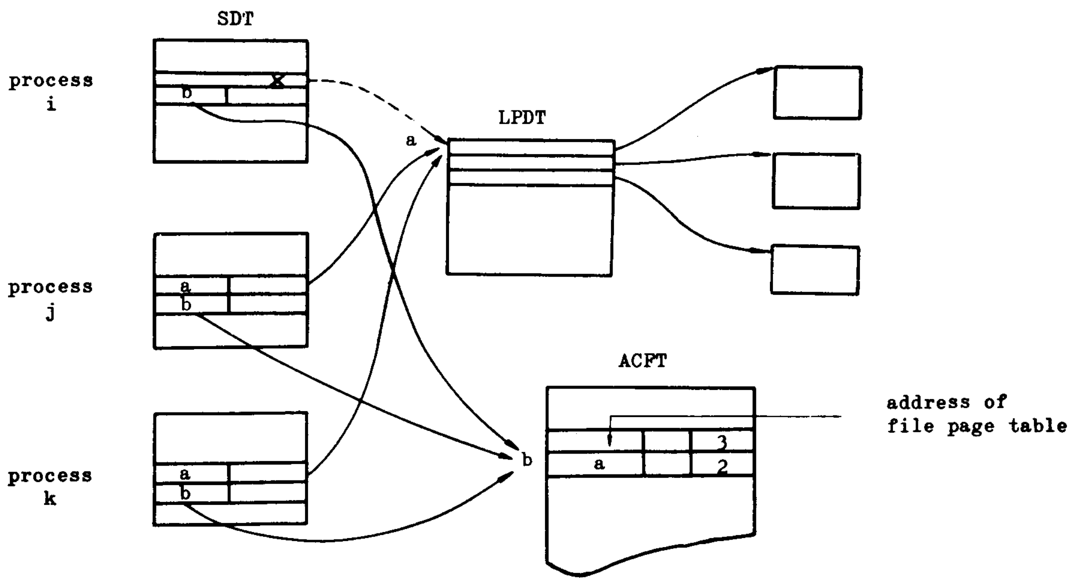

Fig. 9. Relation Between Segment Descriptors and ACFT Entry for Common Segments

LPDT is stored if it is not a common segment; or the information in the pointer to ACFT entry of the second word is used in the case of a common segment.
Entries in the ACFT contain useful information on assignment, swapping and deletion of any common segment which is activated. Brief descriptions of them follow:

File page table address -- Address of the file page table on the disc for this common file. It is used to examine whether this common file is already in use by another process or not when a new file is called by dynamic linking.

- Current LPDT address -- To show where the LPDT for this common segment is presently stored. It is a core address if this common segment is in core; othersise it is a secondary memory address.
- α -- To show whether the LPDT for this common segment is presently in core or not.
- β -- To show, when the LPDT for this common segment is not in core, whether it is on the disc or drum.
- γ -- To indicate, when the LPDT for this common segment is swapped in or out, swapping of it until end of I/O.
- N -- To indicate the number of processes which use this common segment at present.
- n -- To indicate, when the LPDT for this common segment is in core, the number of processes which are linked to the LPDT at present.

In the following, the procedure of common segment assignment, swapping and deletion based on the information contained in the segment descriptors and the ACFT mentioned above will be explained.

(1) Common segment assignment

When a symbolic call is made from a given program, control transfers to the BFS, and the process is examined as to whether the file is already in use by it or not. If the file is in use, the usage is continued without a new segment assignment. If it is not in use, a file search is made according to given rules until the file page table address for the file is obtained. (This procedure is the same whether the file is a common file or not.) If the file is a common file and in order to determine whether it is already in use by other process, it is necessary to check for any entry in the ACFT which corresponds to the file page table address just obtained. If there is a corresponding entry, a segment number is assigned and the missing-fault bit is set when the segment descriptor is registered. At the same time, a pointer to the corresponding ACFT entry is stored. With these settings and when a missing fault is found in the segment descriptor, the current LPDT address in the ACFT is obtained based on this pointer to the ACFT entry and linking to the common LPDT is accomplished. In response to the increase in number of processes using the common segment, N in the ACFT entry is increased by 1. If there is no corresponding entry in the ACFT, this is interpreted to mean that the common file is called first by this process, and a new entry is stored on the ACFT.

(2) Missing page fault in the segment descriptor

A missing page fault may occur in the segment descriptor during execution of a program. If it is a page belonging to a common segment, the ACFT is examined with the help of the pointer to the ACFT entry stored in the second word of the segment descriptor in which the fault occurs and a determination is made where the corresponding LPDT is located at present. If it is not in core, the process is blocked until swapping in of the LPDT is completed. If it is in core, the current LPDT address indicates its core address. This address is stored in the segment descriptor and the missing fault bit is turned off, whereby linking to the LPDT is accomplished. In response to the increase in number of processes connected to the LPDT in core, n in the ACFT entry is increased by 1. If the LPDT is not in core, it means that this is a new common segment reference or that the LPDT is being swapped out. In either case, swapping in from the second memory indicated by the current LPDT address is accomplished.

(3) Swapping of a common segment

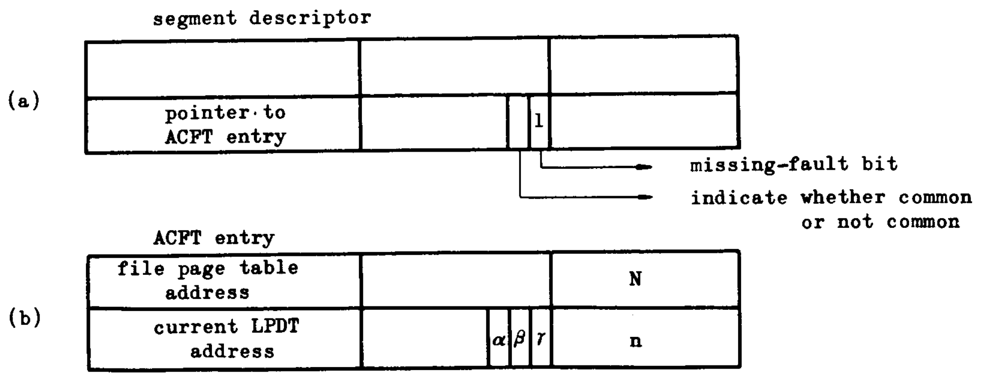

Fig. 10. Contents of Segment Descriptor and ACFT entry

Swapping of a program or mapping tables for any segment which is not a common segment causes no significant problems. When a program page for a common segment is swapped, there is a possibility that the referenced segment is also used by another process during swapping. In this case, an indicator is set up in each page descriptor to indicate swapping or no-swapping; while this indicator is used the processes referring to this page are blocked until swapping is completed.
In the case of swapping of a LPDT for any common segment, linkages to other processes must be checked with the help of the information contained in the ACFT. To swap out the LPDT, a first check is made to see whether the LPDT is actually linked to several processes. If $n = 1$, it is linked to the present process only. Then, n is set to 0, and the LPDT is swapped out. While the LPDT is swapped, the swap indicator in the ACFT is set accordingly. If $n > 1$, the LPDT is linked also to other processes and can not be swapped out. Then, n is set to $n - 1$, the missing fault bit in the segment descriptor is set, and linking to this process only is cut off.

(4) Deletion of common segment

Deleting all the segments used by a process at the end of a user's job or deleting any segment at any given time becomes a problem if these are common segments. The deletion of any common segment can not be attempted until it is requested by all the processes which use it. The information to control this is N, the number of using processes contained in the ACFT. At each request for segment deletion, N in the corresponding ACFT entry is decreased by 1; when it reaches zero, programs, LPDT, etc. for the segment are deleted from the main or secondary memories. When a common segment is deleted, the corresponding entry in the ACFT is also removed.

Scheduling and Swapping Algorithms

One of the most important functions of the time-sharing supervisor programs is to control the swapping mechanism on whose efficiency the total system greatly depends, especially when the number of processes is great. At the initial release of 5020TSS in June, 1968, frequent use of very large programs such as PL/1W compiler (a subset of PL/1) caused the paging input/output traffic to be significantly larger than expected, and significant performance degradation occurred. To resolve these problems we made further studies from various points of view, resulting in successive improvements of our swapping algorithms. Consequently, system performance improved with subsequent release.
In this section, we first describe a brief summary of process scheduling, and then describe the swapping algorithms which are very closely associated with the process scheduling.
In the 5020TSS, a process may be in one of four execution states:

(1) Running
(2) Ready
(3) Pending
(4) Blocked

A process is in the running state if it has currently control of the machine. If a process is in ready or pending states it is ready to run but is waiting for service. A process is in the blocked state if it is waiting for a "wakeup" signal, for example an terminate signal of input/output from or to a remote terminal.
An important difference between ready state and pending state is that while the programs belonging to the ready processes are not basically swapped out, the programs belonging to the pending processes may be swapped out when needed.
Transitions among these states are caused for the following reasons and they are handled as described below (see Fig. 11):

(1) A running process required to swap-in through a missing-page-fault is placed at the end of the ready queue. In this case, a bit in the corresponding process table entry is set to indicate that the process is waiting for the "swap-in" terminate signal.
(2) A running process is placed in the pending queue when it has used up a time slice or when available core pages were not given to the process although it requested them.
(3) A running process goes to the blocked queue when it must wait for a wakeup signal such as the end of terminal input or output. A blocked process is transfered to the pending queue when it is re-initiated by a wakeup signal.
(4) A pending process is put in the ready queue if the controlling number of ready processes permits this.

We can now begin to describe the swapping mechanism that is associated with process scheduling. Experience with the 5020TSS operations proved that the swapping mechanism is effective even in severe circumstances.
The swapping algorithms are constructed as follows:

(1) Swapping-in is performed on a page basis as requested, while swapping out is performed in units of several pages when it occurs. Core memory is always controlled to keep the number of available pages optimum so that they can be used immediately upon receipt of a request for "swap-in". When available core spaces becomes smaller than the optimum number of pages, swapping out is started to make room in core memory.

(2) When swapping out of pages is required, programs belonging to blocked processes are first selected. If no pages of blocked processes can be swapped out, or if there are no blocked processes, programs belonging to the pending processes are swapped out next. Programs belonging to the ready processes can not be swapped out if core memory is fully

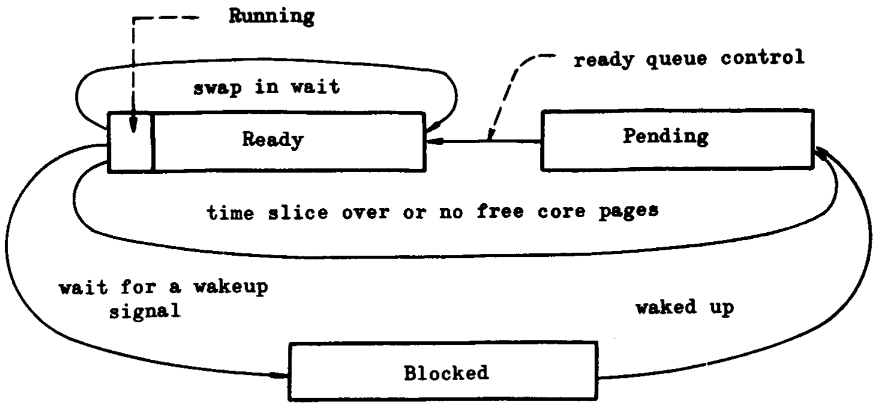

Fig. 11. Execution State Transitions.

occupied by ready processes only; therefore, the running process is temporarily put in the pending queue so that its status can be changed to that of supplying pages.

(3) As indicated above, programs belonging to blocked or pending processes may be swapped out at any time, but the pages on which each pending process was previously running should not be swapped out, since the pending processes will be put back in the ready queue and run before long.

(4) Although swapping out is performed in units of several pages, these pages can be swapped out by a single instruction using a command chainning technique. By this technique, the time wasted to each trap handling is considerably reduced. Furthermore, to take even greater advantage of command chainning, free pages on the drum needed for this swap are selected in such a way that the access time will be as small as possible.

(5) In case of swapping out a common segment which belongs not only to blocked or pending processes but also to some ready processes, there is a possibility that the pages currently used by the ready processes may be swapped out. To prevent this, a "use table" is set up to store page numbers in recent use. This table is a sort of push down table of proper size controlled in a first-in first-out fashion. A new entry is put in the table when a page belonging to a common segment is swapped in core from drum or disc. When a page of a common segment is to be swapped out and the corresponding page number is found in this table, the swap is not executed. By this strategy, many vain attempts to swap out common programs are avoided.

(6) When a page belonging to a pure procedure segment (which may or may not be common) is swapped in, the correspondence between the core page and the drum page is established by placing an entry in a table without releasing the drum page. When swapping out of a page belonging to the pure procedure becomes necessary and the corresponding entry of the page is found in the table, the core page is only released but without swapping.

(7) Very large or malicious programs which request too much core memory without consuming much computer time will badly affect other processes. One important policy algorithm refuses to allow too many pages to be loaded while the process is in ready queue by assigning an upper limit to the number of loaded pages. When the limit is reached, the malicious process is forced into the pending state so that the pages belonging to the process may be swapped out.

Conclusions

The first version of the two-dimensional addressing 5020 TSS, the neucleus of the supervisory system, was completed in June, 1968. A second version is now being developed with the following considerations:

(1) Evaluation of the two-dimensional addressing scheme

The two-dimensional feature can be criticized in various respects, but the scheme itself seems to lead to a highly refined architecture. We are in the process of gathering enough data so that we may proceed with extensive analysis including system overhead, on the system we developed.

(2) Problem of software self-proliferation and software productivity improvement

It is a very interesting and important problem to consider how the self-proliferation and productivity improvement of software systems are made and should be made.

(3) Practicality of the time-sharing system

At present, the time-sharing system in this country may be considered as being still in an experimental stage. This system, however, in the near future, should become a basic tool for creating a new field of computer utility.

Acknowledgement

In performing this study, the authors were given many useful ideas in discussions and meetings by professor Hidetosi Takahasi, professor Eiichi Goto, and professor Eiichi Wada, all of Tokyo University. The authors also wish to thank professor Mamoru Hosaka and professor Setsuo Osuga of the Tokyo University's Institute of Space and Aeronautical Science for useful advice given. Finally the authors wish to thank professor Akira Inatomi of Meiji University, Dr. Shozo Shimada, Dr. Kisaburo Nakazawa, and Mr. Ikuo Nakata, all of Hitachi Ltd., for cooperation in the stage of system design.

References

1) F. J. Corbato and V. A. Vyssotsky, "Introduction and Overview of the MULTICS System", AFIPS Conference Preceedings, Vol. 27 (1965 FJCC).

2) J. H. Saltzer, "Traffic Control in a Multiplexed Computer System", MAC-TR-30 (1966, Project, MAC).

3) J. B. Dennis, "Segmentation and the Design of Multiprogramed Computer Systems", JACM 12, 4, pp. 589 - 602 (1965).