(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 273 612 B1**

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication of patent specification : **09.08.95 Bulletin 95/32**

(51) Int. Cl.[6] : **G06F 9/46**, G06F 15/16

(21) Application number : **87310793.2**

(22) Date of filing : **09.12.87**

(54) Multiprocessor memory management method and apparatus.

(30) Priority : **22.12.86 US 941703**

(43) Date of publication of application : **06.07.88 Bulletin 88/27**

(45) Publication of the grant of the patent : **09.08.95 Bulletin 95/32**

(84) Designated Contracting States : **BE DE FR GB IT NL SE**

(56) References cited :
7TH ANNUAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 6th - 8th May 1980, pages 131-138; J.M. TOBIAS: "A single user multiprocessor incorporating processor manipulation facilities"
IEEE SOFTWARE, vol. 2, no. 4, July 1985, pages 30-37; P. EMRATH: "Xylem: An operating system for the cedar multiprocessor"

(73) Proprietor : **AT & T Corp.**
32 Avenue of the Americas
New York, NY 10013-2412 (US)

(72) Inventor : **Bishop, Thomas Patrick**
2505 Lincolnwood Court
Aurora Illinois 60505 (US)
Inventor : **Davis, Mark Henry**
29W320 Iroquois Court North
Warrenville Illinois 60555 (US)
Inventor : **Fish, Robert Wayne**
29W271 Canterbury Drive
West Chicago Illinois 60185 (US)
Inventor : **Peterson, James Stuart**
255 Peppertree Lane
Aurora Illinois 60505 (US)
Inventor : **Surratt, Grover Timothy**
28 West 716 Hickory Lane
West Chicago Illinois 60135 (US)

(74) Representative : **Watts, Christopher Malcolm Kelway, Dr. et al**
AT&T (UK) Ltd.
5, Mornington Road
Woodford Green Essex, IG8 0TU (GB)

EP 0 273 612 B1

Description

Technical Field

This invention concerns allocation of memory to processes in a multiprocessor system comprising a plurality of processors each having its own memory.

Background of the Invention

Multiprocessor computers are generally classified as being either "tightly-coupled" or "loosely-coupled", according to the memory arrangement which they use.

Tightly-coupled systems are shared-memory systems wherein a plurality of processors typically have no memory dedicated to their own use (other than cache memory) and share use of a centrally-administered memory. Central administration by a single control entity avoids the difficulty of coordinating the work of a plurality of control entities, and enables the central entity to manage the memory without conflict and easily to keep track of the status of every portion of that memory. Hence, memory management -- including the allocation of memory storage space to processes -- is easily implementable and may be made rather flexible. On the other hand, centralized memory management is a "bottleneck" that may restrict the performance of the multiprocessor. The central control entity also adversely affects the fault-tolerance of the multiprocessor, because it is a single point whose failure will generally incapacitate the whole system.

Loosely-coupled systems are non-shared memory systems wherein each processor has its own memory substantially dedicated to its own use and administered locally, i.e., by that processor's own control entity. Because each memory is locally administered by a different control entity that has no knowledge of, and no control over, memory of any other processor, extensive coordination and cooperation between the various processors is required for unified, system-wide, memory management. To avoid conflict between the plural control entities, assignment of processes to processors is generally predetermined and static, i.e., unvarying, and allocation of memory to processes is complex and rigidly limited.

Attempts at hybridizing these types of system have been made. An illustrative example thereof is disclosed by J.M. Tobias in 'A Single User Multiprocessor Incorporating Processor Manipulation Facilities', *Proceedings, 7th Annual Symposium on Computer Architecture (6-8 May 1980)*, pages 131-138.

Despite these attempts, in unsolved problem in the art is how to provide flexible and dynamic memory management, akin to that available in tightly-coupled multiprocessor systems, in non-shared memory multiprocessor systems.

Summary of the Invention

This invention is directed to solving this and other problems and disadvantages of the prior art. According to the invention, in a multiprocessor system that includes a plurality of processors each one of which has its own memory, each processor's memory is logically divided into a first and a second portion, referred to as a global and a local portion. A first control arrangement of the multiprocessor allocates to processes memory included in the first memory portions. Illustratively, the first control arrangement includes a process manager located on one of the processors, whose functions include assignment of processes to processors. A plurality of second control arrangements, a different one of which is associated with each processor, each allocate to processes memory included in the second memory portion of the associated processor. Illustratively, each second control arrangement includes an operating system kernel whose capabilities include conventional memory management functions. The kernel both allocates local memory to, and deallocates local memory from, processes assigned to its associated processor.

The invention has the desirable memory administration features of both the tightly-coupled and the loosely-coupled multiprocessor systems. On one hand, the global memory portions are administered in the manner of a tightly-coupled multiprocessor system, making memory management functions, such as process-to-processor assignment and allocation of initially-required memory to newly-created processes, flexible and easily implementable. Since these functions are centralized in a single control entity, there is no need to coordinate the activities of a plurality of control entities as in loosely-coupled systems. On the other hand, the local memory portions are administered in the manner of a loosely-coupled system. Because local memory administration pertains to intra-processor functions, such as allocation of memory to, and deallocation of memory from, processes assigned to the one processor, there is no need for coordinating the activities of a plurality of control entities with respect to these functions. Rather, each processor retains autonomous local control over its local memory. And because control of local memory is decentralized, there is no control bottleneck that may hamper the performance of the multiprocessor, or create a single point of failure for memory administration functions other than processor-to-processor assignment.

Preferably, the first control arrangement and the plurality of second control arrangements cooperate with each other to selectively transfer memory between the first and the second portion of memory of any processor. The transfers are made in response to

occurrence of predetermined conditions. For example, storage space is transferred from the global portion to the local portion when the amount of memory in the local portion that is not allocated to processes subceeds -- falls below -- a predetermined minimum, and memory is transferred from the unallocated local portion to the global portion when the amount of memory in the unallocated local portion exceeds a predetermined maximum. The size of the local and global memory portions is therefore not static, but may be dynamically varied to meet system needs. Because the local portion can "borrow" storage space from the global portion and supply excess storage space back to the global portion, memory is more efficiently utilized - and hence may be made smaller - than would be the case in a similar but statically-partitioned system.

In one illustrative embodiment of the invention, the first control arrangement does not allocate memory to process directly, but does so through the secondary control arrangements. The first control arrangement selects a processor for a process, among others on the basis of whether the processor has sufficient global memory to satisfy the process' initial memory requirements. The first control arrangement then transfers the required amount of memory from the global memory portion to the uncommitted local memory portion of the selected processor. The second control arrangement of the selected processor then allocates the required memory from the uncommitted local memory portion to the process.

This embodiment has the advantage that the first control arrangement is freed of most memory management functions and the record-keeping that accompanies them. For example, the first control arrangement need not keep records about how much, or which particular parts, of memory are allocated to processes, or even which particular parts of memory comprise the global portion. The first control arrangement only needs to know the amount of memory -- illustratively the number of pages -- that are within the global memory portion. Structure of the process manager is thus simplified and its performance is improved.

These and other advantages and features of the present invention will become apparent from the following description of an illustrative embodiment of the invention taken together with the drawing.

Brief Description of the Drawing

FIG. 1 is a block diagram of an illustrative multiprocessor system including an exemplary embodiment of the invention;

FIG. 2 is a block diagram of the global structure of the host processor of FIG. 1;

FIG. 3 is a block diagram of the tunable structure of an adjunct processor of FIG. 1;

FIG. 4 is a state transition diagram of memory of the system of FIG. 1;

FIG. 5 is a flow diagram of the operation of the system of FIG. 1 in response to a FORK call;

FIG. 6 is a flow diagram of the operation of the system of FIG. 1 in response to an EXEC call;

FIG. 7 is a flow diagram of the operation of the system of FIG. 1 in response to a dynamic memory allocation request;

FIG. 8 is a flow diagram of the operation of the system of FIG. 1 in response to an EXIT system call or to a dynamic memory deallocation request; and

FIG. 9 is a flow diagram of local-to-global memory transfer function of the system of FIG. 1.

Detailed Description

FIG. 1 shows a multiprocessor computing system comprising a plurality of processors 10-12 interconnected for communication by a communication medium such as a bus 15. Each processor includes at least three functional units: an interface 20 which enables the processor to communicate across bus 15 according to the protocol of bus 15, a central processing unit (CPU) 21 for executing processes including application and control processes, and a memory 22 which provides storage space for storing programs, data, and other information for use by processes assigned to the processor. For ease of reference, storage space within a memory will generally henceforth be referred to as memory.

Processors 10-12 include a host processor 10, which serves as the administrative and control center for the overall system of FIG. 1. Illustratively, host processor 10 is the 3B15 computer of AT&T-Information Systems Inc. Processors 10-12 also include a plurality of adjunct processors 11-12 each of which performs system functions specified by host processor 10. Illustratively, processors 11-12 each comprise a member of the WE^R 32100 family of microprocessors of AT&T, with attached memory. Bus 15 is, in this example, a high-speed packet bus, such as a version of the S/NET bus described in an article by S. R. Ahuja entitled "S/NET: A High-Speed Interconnect for Multiple Computers" in IEEE Journal on Selected Areas of Communications, Vol. SAC-1, No. 5 (Nov. 1983), and in U.S. Patent Nos. 4,384,323 and 4,514,728 to Ahuja.

The system of FIG. 1 operates under control of an operating system, illustratively a version of the UNIX^R operating system of AT&T. The operating system is distributed across the processors 10-12. Distributed operating systems are known in the art. For example, an adaptation of the UNIX operating system to multiprocessors is described in an article by P. Jackson entitled "UNIX Variant Opens a Path to Managing Multiprocessor Systems," in Electronics, (July 28, 1983),

pages 118-124. Program code of the operating system kernel 31 (the memory-resident portion of the operating system) is stored in memories 22 and executes as a series of processes on CPUs 21. Other portions of the operating system are stored on disk (not shown) that is illustratively attached to host processor 10 or one of processors 11-12 acting as a file server, and are brought into memory 22 of processor 10-12 when needed. Kernel 31 is replicated on each processor 10-12 and provides each processor 10-12 with the same basic operating system services. Kernel 31 of each processor 10-12 may additionally provide unique services for its processor. Included among the capabilities replicated on each processor 10-12 are memory management functions that perform allocation and deallocation of memory to/from processes assigned to the same one of the processors 10-12 on which the memory management function resides.

Further included in the operating system kernel is a process (or a collection of processes) referred to as a process manager 30. Process manager 30 is not replicated; it is located on host processor 10 only. Process manager 30 is a centralized control arrangement whose functions include the assignment of processes for execution to various ones of processors 10-12.

Memory is considered to be a local resource within the system of FIG. 1. That is, memory 22 of a processor 10-12 is allocated and deallocated to processes locally at each processor 10-12, by that processor's kernel 31 which has substantially no knowledge of, or control over, memory 22 of any other processor 10-12. However, processes are assigned for execution to processors 10-12 by process manager 30, which must known how much memory is available for initial assignment to processes on each processor 10-12, so that it can determine whether a particular processor 10-12 is capable of executing a given process.

For this reason, memory management is implemented in the system of FIG. 1 as follows. Memory 22 of each adjunct processor 11-12 is logically divided into two portions: local memory 41 and global memory 42. Local memories 41 together form a local memory pool, while global memories 42 together form a global memory pool. Local memories 41 of the local pool are managed by kernels 31 each one of which administers the local memory 41 of its associated processor, while the global pool is administered by process manager 30 in coordination with kernels 31. Since process manager 30 administers the global pool, it always knows the size of global memories 42 thereof. Process manager 30 uses this information to decide whether a processor 10-12 has sufficient memory available for allocation to processes to support assignment of any particular process thereto. Process manager 30 directly administers all memory 22 of host processor 10, using memory management functions of kernel 31 of processor 10 to do so. Process manager 30 administers memory 22 of host processor 10 in a conventional, uniprocessor, manner. It effectively views all memory 22 of processor 10 in the same manner as it views global memory of other processors.

Memory of each processor 10-12 is also logically divided into committed memory 46 and uncommitted memory 45. Uncommitted memory 45 is memory that has not been allocated for use to any process and is, therefore, free and available for allocation. Committed memory 46 is memory that has been allocated by a kernel 31 for use to a process and is therefore considered to be in use and unavailable for allocation. On adjunct processors 11-12, only local memory 41 may be committed. Global memory 42 is always uncommitted; this will be made clear further below.

To avoid disadvantages of static and inflexible memory partitioning, process manager 30 and kernels 31 of processors 11-12 may cooperatively transfer memory between local memory 41 and global memory 42 of a kernel's associated processor. Memory is thus allowed to "migrate" between local memory 41 and global memory 42, upon agreement between process manager 30 and kernel 31 of adjunct processor 11-12 to or form whose local portion 41 the memory is migrating.

Memory migration is graphically illustrated by FIG. 4. FIG. 4 shows that any portion -- illustratively any page -- of memory 22 of an adjunct processor 11-12 is capable of being in any one of three states: allocated to global memory state 400, allocated to uncommitted local memory state 401, and allocated to committed local memory state 402. The occurrence of various predetermined conditions, described further below and represented in FIG. 4 by state transitions 403-409, cause memory to migrate from state to state.

Process manager 30 keeps track of the global memory pool by means of global structure 50. Structure 50 is shown in greater detail in FIG. 2. Structure 50 is a data structure -- illustratively shown in FIG. 2 as a table -- stored in memory 22 of processor 10. Structure 50 includes a plurality of entries 211-213 each one of which corresponds to a different one of processors 10-12, respectively. Each entry stores information indicating the amount of memory included in global memory 42 of the associated processor. UNCOMMITTED CT. entry 211 associated with host processor 10 stores information indicating the amount of storage space included in unallocated memory 45 of processor 10.

Illustratively, memory 22 of each processor 10-12 is logically partitioned into pages -- each one being 2048 bytes in size, for example -- and each GLOBAL CT. entry 211-213 stores a count of the number of pages of memory included in unallocated memory 45 (entry 211) or in corresponding global memory 42 (entries 212-213). Nor particular pages are associat-

ed with the count, and the memory manager 30 does not care about the identity of pages - it cares only about numbers of pages. Hence, no information identifying any particular pages need be stored in entries 211-213 of global structure 50.

Each adjunct kernel 31 manages its corresponding local memory 41 substantially in a conventional manner. However, for purposes of efficient administration of the division of associated memory 22 into local memory 41 and global memory 42, each kernel 31 has its own tunable structure 51. An illustrative tunable structure 51 is shown in FIG. 3. Structure 51 is a data structure -- illustratively shown in FIG. 3 as a table -- stored in memory 22 of processors 11-12 on which kernels 31 reside. Structure 51 includes a plurality of entries 201-207 all pertaining to the memory 22 of the one adjunct processor on which it is located. GLOBAL CT. entry 201 stores information indicating the amount of memory included in global memory 42. This memory is represented by state 400 in FIG. 4. LOCAL CT. entry 202 stores information indicating the amount of memory included in uncommitted local memory 45. This memory is represented by state 401 in FIG. 4. INITLOCAL entry 203 stores information indicating the amount of memory to be included in unallocated local memory 45 upon initialization of the processor; remaining unallocated memory is included upon initialization in global memory 42. MINLOCAL entry 204 stores information indicating the miniumum amount of memory that uncommitted local memory 45 should desirably contain. If and when the amount of memory in uncommitted local memory 45 subceeds the minimum specified by entry 204, kernel 31 attempts to have additional storage space migrate to uncommitted local memory 45 from global memory 42. This migration is represented by state transition 404. MINGTOL entry 205 stores information indicating the minimum amount of memory that should desirably be migrated at any one time from global memory 42 to local memory 41, to avoid "thrashing". Less memory than the minimum specified by entry 205 is migrated generally only when global memory 42 contains less memory than that minimum. MAXLOCAL entry 206 stores information indicating the maximum amount of memory that uncommitted local memory 45 should desirably contain. If and when the amount of memory in uncommitted local memory 45 exceeds the maximum specified by entry 206, kernel 31 migrates, with approval of memory manager 30, some of that memory from local memory 41 to global memory 42. This migration is represented in FIG. 4 by state transition 403. MINLTOG entry 207 stores information indicating the minimum amount of memory that should desirably be migrated at any one time from uncommitted local memory 45 to global memory 42, to avoid "thrashing". Less memory than the minimum specified by entry 207 is migrated generally only when migration of that much memory would reduce the amount of memory in uncommitted local memo ry 45 below the minimum specified by entry 204.

Certain relationships exist among entries 203-207, as follows. The amount of memory indicated by MAXLOCAL entry 206 is greater than the amount of memory indicated by MINLOCAL entry 204. The amount of memory indicated by INITLOCAL entry 203 is bounded by the amounts of memory indicated by MAXLOCAL entry 206 and MINLOCAL entry 203. The difference between the amounts of memory indicated by MAXLOCAL entry 206 and MINLOCAL entry 203 is equal to or greater that the amount of memory indicated by MINGTOL entry 205 and by MINLTOG entry 207. And preferably, the amounts of memory indicated by MINLTOG entry 207 and by MINGTOL entry 205 are not equal to each other.

Illustratively, if memory 22 is logically partitioned into pages, each entry 201-207 stores a count of some number of pages; no information identifying particular pages need be stored in tunable structure 51. In one illustrative embodiment of the system of FIG. 1, wherein each memory 22 of processors 11-12 has one or two thousand pages of storage space and wherein the operating system kernel occupies approximately 200 to 500 pages of each memory 22, entry 203 indicates 5 pages, entry 204 indicates 5 pages, entry 205 indicates 40 pages, entry 206 indicates 75 pages, and entry 207 indicates 50 pages.

The UNIX operating system includes three process creation and termination primitives, which require allocation and deallocation of memory to/from processes.

A FORK primitive duplicates an existing process. The parent and child processes are identical in program -- they share the text portion of memory allocated to them. However, each duplicated process has its own data portion in memory, so the processes can execute independently of each other.

An EXEC primitive transforms an existing process into a different process, by causing the process issuing the EXEC primitive to stop executing its present program and to begin executing a specified new program. Hence, the old and the new process each must have its own text and data portions in memory. As part of the EXEC primitive call, generally there is specified the minimum amount of memory that must initially be allocated to the transformed, new, process.

Generally, one process creates a new process by means of execution of the FORK primitive, immediately followed by execution of the EXEC primitive on the child process.

An EXIT primitive terminates a process and releases -- deallocates -- its allocated memory. The EXIT primitive releases both the test and the data portions of a process not sharing the text memory portion with another process; it realeases only the data memory portion of a process sharing the text memory por-

tion with another process.

Additionally, the UNIX operating system includes four system calls by means of which an existing process can dynamically, i.e., during its execution, request that either additional memory be allocated thereto, or that some of the memory allocated thereto be deallocated. These system calls are BRK, SBRK, SHMCTL, and an EXECLNUP portion of the EXEC system call. Arguments accompanying the first three system calls specify whether memory allocation or deallocation is being requested (EXECLNUP only deallocates memory), and the amount of memory being requested. The BRK and SBRK system calls are for memory dedicated to a process, while SHMCTL is a call for memory shared by a plurality of processes. EXECLNUP is a call for deallocation of memory of an old process that has been successfully transformed into a new process.

The activities relevant to this invention that are undertaken by memory manager 30 and kernels 31 in response to the above-described primitives and system calls are diagramed in FIGS. 5-9.

FIG. 5 shows the system response to a FORK call. Since the parent and child processes will share a common text memory portion, they will be colocated on the same processor 10-12 on which the parent process is currently located. Hence, no choice of process-to-processor assignment is required.

For purposes of illustration, assume that the FORK primitive is called by a process on adjunct processor 11. In response to the FORK call, at step 500, kernel 31 of processor 11 determines the memory requirements of the child process, at step 501. Kernel 31 knows the amount of memory being used by any process on processor 11, and the child process is merely a copy of the parent. Hence, kernel 31 determines the memory requirements of the child from those of the parent. Next, at step 502, kernel 31 of processor 11 examines entry 201 of is tunable structure 51 to determine whether there is sufficient memory available in global memory 42 to satisfy that memory requirement. Illustratively, it does so by comparing the memory requirement of the child process with global count entry 201 of its tunable structure 51 to determine whether global memory 42 includes the amount of memory required by the child process. If sufficient global memory 42 is not available -- the child process' memory requirement exceeds entry 201 -- kernel 31 proceeds to step 506.

If kernel 31 determines at step 502 that sufficient global memory is not available for the process to FORK, kernel 31 fails the FORK, at step 507. The activities undertaken by kernel 31 at step 507 are conventional. But before failing the FORK, kernel 31 performs the local-to-global memory transfer function of FIG. 9 to transfer an amount of memory from uncommitted local memory 45 to global memory 42, at step 506. This transfer is represented by state transition 403 in FIG. 4. Kernel 31 does so in anticipation of the failed FORK being attempted again in the very near future, and tries to ensure thereby that the next FORK attempt will not fail due to lack of sufficient global memory 42. After failing the FORK at step 507, involvement of kernel 31 in the FORK comes to an end, at step 508.

If kernel 31 determines at step 502 that sufficient global memory is available for the process to FORK, kernel 31 requests host processor 10 to allow the parent process to FORK, at step 503. Illustratively, kernel 31 does so by causing a packet carrying information to that effect to be sent over bus 15 to process manager 30. The transferred information includes the memory requirement of the child process. Kernel 31 then awaits a response from process manager 30, at step 504.

The communication between processor 11 and processor 10 is kernel synchronous: processor 11 cannot dispatch any new process until processor 10 answers the request. It is logically a remote subroutine call. While requesting kernel 31 waits, it can handle local interrupts and also remote requests that it may receive, but no other process can be dispatched.

Process manager 30 responds to receipt of the FORK request, at step 510, by examining entry 212 of global structure 50 that corresponds to adjunct processor 11 to determine whether there is sufficient memory available in global memory 42 of adjunct processor 11, at step 511. Process manager 30 must do this because entry 212 of global structure 50 and entry 201 of tunable structure 51 of adjunct processor 11 are maintained by separate control mechanisms 30 and 31 and hence may at certain times by out of synch and not have identical contents.

If process manager 30 determines at step 511 that sufficient global memory is not available, it notifies adjunct processor 11 of its refusal to allow the FORK, at step 515. Process manager 30 does this illustratively by causing a packet having information to this effect to be sent to processor 11 over bus 15. Involvement of process manager 30 in the FORK thus ends, at step 516.

In response to receipt at step 505 of the refusal to allow the FORK, kernel 31 fails the FORK, at step 507, and ends its involvement in the FORK, at step 508.

If sufficient global memory is found to be available to processor 11, at step 511, process manager 30 decrements entry 212 of global structure 50 by the amount of requested memory, at step 512, thereby initiating migration of that amount of global memory 42 of processor 11 to local memory 41 of processor 11. This migration is represented in FIG. 4 by state transition 407. Process manager 30 then updates information that it keeps on all processes within the system of FIG. 1 to include the child process, at step 513. The activities undertaken by process manager 30 at

step 513 are substantially conventional, but also include storing of information regarding which processor 10-12 the child process is located on. Process manager 30 then notifies adjunct processor 11 of grant of the FORK request, at step 514. Illustratively, process manager 30 does so by causing a packet carrying information to that effect to be transmitted to adjunct processor 11 over bus 15. Involvement of process manager 30 in allocating memory to the child process is thus ended, at step 516.

Kernel 31 of processor 11 responds to receipt of notice of the FORK grant, at step 520, by initially blocking local-to-global (L-to-G) memory transfers from being made, at step 521. The transfers are blocked by means of incrementing a count semaphore 52 (see FIG. 1) associated with tunable structure 51 of processor 11 and located on processor 11. Count semaphores are well known in the art. Kernel 31 then transfers the amount of memory required by the child process from global memory 42 to uncommitted local memory 45, at step 522, to complete the migration represented in FIG. 4 by state transition 407. Kernel 31 accomplishes the transfer by decrementing entry 201 of its tunable structure 51 by the amount of requested memory, and incrementing entry 202 by the same amount.

LOCAL CT. entry 202 of structure 51 may now exceed MAXLOCAL entry 206 of structure 51. This result would generally trigger a migration, represented in FIG. 4 by state transition 403, of some of uncommitted local memory 46 to global memory 42. It was for the purpose of preventing this migration from occurring that kernel 31 blocked local-to-global memory transfers at step 521.

Following the memory transfer at step 522, kernel 31 allocates the requested amount of uncommitted local memory 45 to the child process, in a conventional manner, and decrements entry 202 of its tunable structure 51 to reflect this fact, at step 523.

This allocation migrates the allocated memory from uncommitted local memory 45 to committed local memory 46. This migration is represented in FIG. 4 by state transition 408. The net result of the FORK on memory 22 of processor 11 thus is the migration of the required amount of memory from global memory 42 to committed local memory 46. This net migration is indicated in FIG. 4 by state pseudo-transition 409, which represents the net effect of state transitions 407 and 408.

Kernel 31 completes the FORK, at step 525, in a conventional manner. Activities performed by kernel 31 at step 525 include updating of records that indicate which processes are assigned to processor 11, updating of records that indicate what memory is assigned to what process, and notifying the other processes affected by the FORK, including the parent process, of the success of the FORK. This being done, kernel 31 unblocks local-to-global memory transfers, at step 526, by decrementing counting semaphore 52. Involvement of kernel 31 in the FORK is thus ended, at step 527.

FIG. 6 shows the system response to an EXEC call. The transformed process, referred to as the new process, may end up being located on either the same processor 10-12 or on a different processor 10-12 than the original process, referred to as the old process. A choice of process-to-processor assignment is therefore required, and must be made by process manager 30. If the new process is not assigned by process manager 30 to the old process' processor, no memory need be allocated on the old process' processor as a consequence of the EXEC; only the old process' memory need be deallocated.

For purposes of illustration, assume that the EXEC primitive is called by a process on processor 12. In response to the EXEC call, at step 600, kernel 31 of processor 12 determines the memory requirements of the new process, at step 601. As was indicated above, the new process' initial memory requirements are specified directly by the EXEC call, for example. Alternatively, the memory requirements are specified in an object file that is to be EXEC'd, and are picked up therefrom by kernel 31. Next, kernel 31 of processor 12 requests process manager 30 to assign memory of a processor to the new process, at step 602. Illustratively, kernel 31 does so by causing a packet having information about the EXEC, including initial memory requirements of the new process, to be sent to host processor 10. Kernel 31 of processor 12 then awaits response from process manager 30, at step 603. Once again, the communication between processor 12 and processor 10 is kernel synchronous.

Process manager 30 responds to receipt of the EXEC request, at step 610, by selecting a processor 10-12 for the new process to run on and assigning the new process thereto, at step 611. Process manager 30 bases the selection on any suitable criteria applicable to the particular configuration of the system of FIG. 1 and the use to which the system is being put. These criteria include, for example, which processor 10-12 has the computing capabilities and peripherals required by the new process, and the current load distribution or processes across processors 10-12.

One of the criteria used by process manager 30 is the availability of memory for assignment to the new process on a processor 10-12. If sufficient memory is not available to a process, the process usually cannot execute. Hence, as part of the process-to-processor assignment performed at step 611, process manager 30 compares the initial memory requirement of the new process with a processor's corresponding GLOBAL CT entry in global structure 50, and assigns the new process to that processor 10-12 only if and when global memory 42 available on that processor 10-12 at least equals the initial memory re-

quirement of the new process.

For purposes of illustration, assume that processor 11 is selected at step 611. Following selection, process manager 30 decrements entry 212 of global structure 50 by the amount of initial memory requirement of the new process, at step 612, thereby initiating migration of that amount of global memory 42 of processor 11 to local memory 41 of processor 11. This migration is represented in FIG. 4 by state transition 407. Process manager 30 then notifies EXEC-requesting processor 12 of the assignment of processor 11 to the new process, at step 613, illustratively by causing a packet to be sent to processor 12 over bus 15. Barring occurrence of any difficulties with performance of the EXEC, involvement of process manager 30 therein is ended thereby, at step 614.

In response to the notice of assignment of processor 11 to the new process, at step 620, kernel 31 of processor 12 processes the EXEC, at step 621, substantially in a conventional manner. The further processing of the EXEC at step 621 leads to either success or failure of the EXEC, as suggested by step 622. If the EXEC fails, kernel 31 notifies process manager 30 of the failure, at step 623. Involvement of kernel 31 of processor 12 in the EXEC then ends, at step 626.

Process manager 30 responds to receipt of the EXEC failure notification, at step 630, by incrementing entry 212 of processor 11 by the initial memory requirement of the new process, at step 631, thereby restoring that memory to global memory portion 42 of processor 11 and terminating the migration represented in FIG. 4 by state transition 407. Involvement of process manager 30 in the EXEC thus ends, at step 632.

Since the EXEC failed, assigned processor 11 has not been actively involved in performing the EXEC.

Returning to step 622, if the EXEC succeeds, kernel 31 of processor 12 notifies kernel 31 of processor 11 of its assignment to the new process, at step 624. Illustratively, notification takes place by means of a packet being transmitted from processor 12 to processor 11 via bus 15. The packet contains information about the EXEC. Since the transformation was successful, the old process ceases to exist and hence its allocated memory must be deallocated. For this purpose, kernel 31 calls the EXECLNUP function of FIG. 6, at step 625. Involvement of kernel 31 of processor 12 in the EXEC then ends, at step 626.

Kernel 31 of processor 11 responds to being notified of the EXEC, at step 650, by initially blocking local-to-global memory transfers from being made, at step 651, by incrementing count semaphore 52. Kernel 31 then transfers the amount of memory required by the new process from global memory 42 to uncommitted local memory 45, at step 652, in the manner described for step 502. Kernel 31 thus completes the migration represented in FIG. 4 by state transition 407. Kernel 31 then allocates the requested amount of uncommitted local memory 45 to the new process, in a conventional manner, and decrements entry 202 of its tunable structure 51 to reflect this fact, at step 653.

This allocation migrates the allocated memory from uncommitted local memory 45 to committed local memory 46. This migration is represented in FIG. 4 by state transition 408. The net result of the EXEC on memory 22 of processor 11 thus is the migration of the required amount of memory from global memory 42 to committed local memory 46. This net migration is indicated in FIG. 4 by state pseudo-transition 409, which represents the net effect of state transitions 407 and 408.

Kernel 31 otherwise completes the EXEC, a step 655, in a conventional manner. Activities performed by kernel 31 at step 655 include those described for step 525. This being done, kernel 31 unblocks local-to-global memory transfers, at step 656, by decrementing counting semaphore 52. Involvement of kernel 31 in the EXEC is thus ended, at step 657.

FIG. 7 shows the system response to a system call by a process for dynamic allocation of memory. For purposes of this invention, this response is the same for the BRK, SBRK, and SHMCTL allocation calls. Since these calls comprise a request by a process assigned to and executing on a particular processor 10-12 for more memory on that one processor, no process-to-processor assignment is involved therein.

Assume that the call occurs on processor 11. In response to the allocation call, at step 700, kernel 31 of processor 11 determines the amount of memory being called for by the calling process, at step 701. Typically, the call will directly specify the required memory amount. Kernel 31 then examines LOCAL CT. entry 202 to determine whether uncommitted local memory 45 includes an amount of memory suitable to satisfy the call. Suitability may be determined on the basis of any desirable criteria, such as sufficiency of available memory. Illustratively, kernel 31 then compares the required amount against entries 202 and 204 of tunable structure 201, at step 702, to determine whether the required amount would reduce uncommitted local memory 45 below the MINLOCAL amount.

If so, kernel 31 compares the required amount against entry 205 of tunable structure 201 to determine whether the required amount exceeds the MINGTOL amount, at step 703. If the required amount does not exceed the MINGTOL amount, kernel 31 requests process manager 30 to transfer the MINGTOL amount of global memory 42 to local memory 41, at step 704. If the required amount does exceed the MINGTOL amount, kernel 31 requests process manager 30 to transfer the required amount of global memory 42 to local memory 41, at step 705. Illustra-

tively, kernel 31 makes the request by causing a packet to that effect to be sent from processor 11 to processor 10 via bus 15. Kernel 31 then awaits response to the global-to-local transfer request from process manager 30, at step 706. Once again, the communication between processor 11 and processor 10 is kernel synchronous.

Process manager 30 responds to receipt of the global-to-local transfer requests, at step 1100, by examining entry 212 of global structure 50 that corresponds to adjunct processor 11 to determine whether there is sufficient memory available in global memory 42 of adjunct processor 11 to satisfy the transfer request, at step 1101.

If process manager 30 determines at step 1101 that the requested amount of memory exceeds the amount of global memory 42, it decrements entry 212 of global structure 50 to zero, at step 1102, thereby initiating migration of all available global memory 42 of processor 11 to uncommitted local memory 45 of processor 11. This migration is represented in FIG. 4 by state transition 404.

If process manager 30 determines at step 1101 that the requested amount of memory does not exceed the amount of global memory 42, it decrements entry 212 of global structure 50 by the amount of requested memory, at step 1103, thereby initiating migration of that amount of global memory 42 of processor 11 to local memory 41 of processor 11. This migration is likewise represented in FIG. 4 by state transition 404.

Following decrementing of global memory 42 at step 1102 or 1103, process manager 30 notifies adjunct processor 11 of how much global memory 42 is being transferred to local memory 41, at step 1104. Process manager 30 illustratively does so by causing a packet carrying the information to be transmitted from processor 10 to processor 11 via bus 15. Involvement of process manager 30 in the global-to-local memory transfer thus ends, at step 1105.

Kernel 31 of processor 11 responds to receipt of the notice of result of the transfer request, at step 710, by transferring the indicated amount of memory from global memory 42 to uncommitted local memory 45, at step 711, to complete the migration represented in FIG. 4 by state transition 404. Kernel 31 accomplishes the transfer in the manner described for step 522.

Next, kernel 31 compares entry 202 of tunable structure 51 of processor 11 against the amount of memory required by the calling process to determine whether there is sufficient memory available in uncommitted local memory 45 to satisfy the requirement, at step 712.

If the required amount of memory exceeds the amount of uncommitted local memory 45, kernel 31 fails the allocation call, at step 716, in a conventional manner. Kernel 31 then ends its response to the memory allocation call, at step 717.

If the required amount of memory does not exceed the amount of uncommitted local memory 45 at step 712, kernel 31 allocates the requested amount of uncommitted local memory 45 to the process, in a conventional manner, and decrements entry 202 of its tunable structure 51 to reflect this fact, at step 713. This allocation migrates the allocated memory from uncommitted local memory 45 to committed local memory 46. This migration is represented in FIG. 4 by state transition 405.

Kernel 31 completes the BRK, SBRK, or SHMCTL call, at step 715, in a conventional manner. Involvement of kernel 31 in allocating memory to the calling process thus ends, at step 717.

If it is determined at step 702 that the required amount of memory would not reduce uncommitted local memory 45 below the MINLOCAL amount, kernel 31 proceeds to step 713 to perform the activities described above for steps 713, 715, and 717.

FIG. 8 shows the system response to a system call by a process for the dynamic deallocation of memory, or to an EXIT primitive call, and to an EXECLNUP call. For purposes of this invention, this response is the same for the BRK, SBRK, SHMCTL, EXIT, and EXECLNUP calls. Since these calls comprise a request concerning a process assigned to a particular processor 10-12 to release some or all of the memory allocated to the process on that one processor, no process-to-processor assignment is involved therein.

Assume that the call occurs on processor 11. In response to the call, at step 800, kernel 31 of processor 11 determines the amount of memory being released, at step 801. Typically, a deallocation call will directly specify the amount of memory being released by the process. And in response to an EXIT or an EXECLNUP call, the kernel 31 is able to determine the amount of memory allocated to a process from information that it stores about that process and every other process assigned to processor 11. Kernel 31 then deallocates the released amount of committed local memory 46, in a conventional manner, and increments entry 202 of its tunable structure 51 to reflect this fact, at step 802. The deallocation migrates the released memory from committed local memory 46 to uncommitted local memory 45. This migration is represented in FIG. 4 by state transition 406.

Kernel 31 completes the BRK, SBRK, SHMCTL, EXIT, or EXECLNUP call, at step 805, in a conventional manner. Kernel 31 then performs the local-to-global transfer function of FIG. 9, at step 804, in order to transfer excess uncommitted local memory 45 to global memory 42. Kernel 31 otherwise completes the call, at step 805, in a conventional manner, and ends its involvement in the deallocation, at step 806.

FIG. 9 shows the local-to-global ^(L-to-G) memory transfer function that is performed on each processor 11-12 periodically -- illustratively once every

second -- or as part of a FORK or a deallocation call (see step 506 of FIG. 5 and step 804 of FIG. 8). The purpose of this function is to transfer excess uncommitted local memory 45 to global memory 42.

Assume that the L-to-G transfer function is being performed on processor 11. Upon commencing the function, at step 900, kernel 31 of processor 11 checks the value of counting semaphore 52 to determine whether local-to-global transfers are blocked, at step 901. If the value of semaphore 52 is not zero, it indicates that transfers are blocked, for example because a FORK or an EXEC call is in progress (see step 521 of FIG. 5 and step 651 of FIG. 6). Kernel 31 therefore ends the function, at step 903.

If the value of semaphore 52 is found to be zero at step 901, indicating that local-to-global transfers are not blocked, kernel 31 compares entry 202 of its tunable structure 51 with entry 206 to determine whether the size of uncommitted local memory 45 exceeds the MAXLOCAL size, at step 902. If the size of memory 45 is below the specified maximum, kernel 31 again ends the function of FIG. 9 without performing any transfers, at step 903.

If the size of memory 45 is found to exceed the specified maximum at step 902, kernel 31 checks entries 202, 206, and 207 of tunable structure 51 against each other to determine whether reducing size of uncommitted local memory 45 by the MINLTOG amount would bring it below the specified maximum, at step 904. If so, kernel 31 requests process manager 30 to transfer the MINLTOG amount of uncommitted local memory 45 to global memory 42, at step 905. If not, kernel 31 requests process manager 30 to transfer from uncommitted local memory 45 to global memory 42 that amount which will reduce the size of memory 45 to the specified maximum, at step 906. Kernel 31 illustratively places the request to memory manager 30 by means of causing a packet having information to that effect to be sent from processor 11 to processor 10 over bus 15. Following placing of the request with memory manager 30, kernel 31 awaits response therefrom, at step 907. Once again, the communication between processor 11 and processor 10 is kernel synchronous.

In response to receipt of the request, at step 910, process manager 30 increments entry 212 of global structure 50 by the amount specified in the request, at step 911, thereby initiating migration of memory from state 401 of FIG. 4 to state 400, represented by state transition 403. Process manager 30 then notifies kernel 31 of processor 11 of this result, at step 912, illustratively by sending an acknowledging packet to processor 11 over bus 15. Involvement of process manager 30 in the transfer then ends, at step 913.

In response to receiving from process manager 30 notice of the result of the request, at step 920, kernel 31 of processor 11 transfers the indicated amount of memory from uncommitted local memory 45 to global memory 42, at step 921, thereby completing the migration represented in FIG. 4 by state transition 403. Kernel 31 accomplishes the transfer by incrementing entry 201 of its tunable structure 51 by the indicated amount of memory, and decrementing entry 202 by the same amount. The local-to-global transfer then ends, at step 922.

Of course, it should be understood that various changes and modifications to the illustrative embodiment described above will be apparent to those skilled in the art. For example, a global-to-local memory transfer may be performed periodically to maintain the size of uncommitted local memory 45 at or above a predetermined minimum.

Claims

1. Multiprocessor apparatus having a bus connecting a plurality of processors (11-12) for executing processes that are assigned to the processors, each processor having its own non-shared memory (22)

   CHARACTERIZED BY

   each of the memories being logically divided into a first global portion (42) and a second local portion (41);

   first control means (30) for allocating to individual ones of the processes an initially-required amount of memory from said first memory portion in the processor to which the individual one of the processes is assigned; and

   a plurality of second control means (31), a different one associated with each processor, each for allocating memory included in the second memory portion of the associated processor to processes assigned to said associated processor;

   the first control means cooperating with the second control means of any processor to transfer memory as needed between the first and second memory portion associated with the second control means.

2. The apparatus or claim 1

   CHARACTERIZED IN THAT

   the first control means is for assigning (611) processes to the processors.

3. The apparatus of claim 1

   CHARACTERIZED IN THAT

   the first control means comprises

   means (610-614, 630-632) for assigning processes to the processors and further for transferring memory included in the first memory portion to the second memory portion for each assigned process, both portions being of the processor to which the process is assigned; and

each of the second control means comprises

means (650-657) for allocating memory included in the second memory portion to processes assigned to the associated processor.

4. The apparatus of claim 1
CHARACTERIZED IN THAT
the first control means and the second control means of any processor cooperate to transfer (FIGS. 7, 9) memory between the first memory portion and the second memory portion of the processor associated with the second control means, on the basis of amount of memory included in the second portion and unallocated to the processes.

5. The apparatus of claim 1
CHARACTERIZED IN THAT
each second control means is further for deallocating (800-806) allocated memory from processes to the second memory portion of the associated processor; and
the first control means and the second control means of any processor cooperate to transfer (FIG. 9) memory from the second memory portion to the first memory portion, both portions being of the processor associated with the cooperating second control means, when an amount of unallocated memory in the second memory portion exceeds a predetermined maximum, and further transfer (FIG. 7) memory from the first memory portion to the second memory portion when an amount of unallocated memory in the second memory portion is exceeded by a predetermined minimum.

6. The apparatus of claim 1
CHARACTERIZED IN THAT
each memory is logically divided into a global, centrally-managed, memory portion (42), a local, locally-managed, memory portion unallocated to processes (45), and a local, locally-managed, memory portion allocated to processes (46);
the first control means is for assigning (611) processes to the processors and further for transferring (612-614, 630-632), for each assigned process, memory included in the global memory portion to the unallocated local memory portion of the processor to which the process is assigned; and
the second control means are each for transferring (650-657) memory from the unallocated local memory portion to the allocated local memory portion of the associated processor, to allocate the transferred memory to processes assigned to the associated processor, and further for transferring (FIG. 8) memory from the allocated local memory portion to the unallocated local memory portion of the associated processor, to deallocate the transferred memory from processes assigned to the associated processor;
the first control means and the second control means cooperating (FIGS. 5-9) to transfer memory between the global memory portion and the unallocated local memory portion, both memory portions being of the processor associated with the second control means, in response to occurrence of predetermined conditions.

7. The apparatus of claim 6
CHARACTERIZED IN THAT
each second control means comprises
its own second table means including a first entry (201) for storing an indication of an amount of memory included in the global memory portion of the associated processor and a second entry (202) for storing an indication of an amount of memory included in the unallocated local memory portion of the associated processor, and
operating system kernel means (31), responsive to a call for duplication of a process assigned to the processor associated with the second control means, for examining (500-502) the first entry of the second table means to determine whether the global memory portion includes an amount of memory required by the duplicated process, and for requesting (503-504) the first control means to allow the duplication if the global memory portion includes the required amount of memory;
the first control means comprises
first table means including a plurality of entries (212-213) each associated with a different one of the plurality of processors and for storing an indication of an amount of memory included in the global memory portion of the associated processor, and
operating system process managing means (30), responsive to the request to allow the duplication, for examining (510, 511) the first table means' entry associated with the processor of the requesting second control means to determine whether the global memory portion of the processor includes the required amount of memory, and for both (a) decrementing (512-513) the processor's associated entry in the first table means by the required amount to initiate transfer of the required amount of memory from the global memory portion to the unallocated local memory portion of the processor and (b) notifying (514) the second control means of a grant of the request for duplication, if the global memory portion includes the required amount of memory; and

the kernel means of the requesting second control means further responsive to the notice of the grant, for both (a) incrementing (520-522) the second entry and decrementing the first entry of the second table means by the required amount to complete transfer of the required amount of memory from the global memory portion to the unallocated local memory portion, and (b) decrementing (523-527) the second entry of the second table means by the required amount to transfer the required amount of memory from the unallocated local memory portion to the allocated local memory portion for allocation to the duplicated process.

8. The apparatus of claim 6
CHARACTERIZED IN THAT
the first control means comprises
first table means including a plurality of entries (212-213) each associated with a different one of the plurality of processors and for storing an indication of an amount of memory in the global memory portion of the associated processor, and
operating system process managing means (30), responsive to a call from a calling processor for transformation of a process, for selecting (610-611) a processor for assignment to the transformed process, the selected processor being of the plurality of processors and having a global memory portion that includes an amount of memory required by the transformed process, and for both (a) decrementing (612) the first table means' entry associated with the selected processor by the required amount to initiate transfer of the required amount of memory from the global memory portion to the unallocated local memory portion of the selected processor and (b) notifying (613-614) the calling processor of the selection; and
each second control means comprises
its own second table means including a first entry (201) for storing an indication of an amount of memory in the global memory portion of the associated processor and a second entry (202) for storing an indication of an amount of memory in the unallocated local memory portion of the associated processor, and
operating system kernel means (31), responsive to notice from the calling processor of selection of the associated processor, for both (a) incrementing (650-652) the second entry and decrementing the first entry of the second table means by the required amount to complete transfer of the required amount of memory from the global memory portion to the unallocated local memory portion, and (b) decrementing (653-657) the second entry of the second table means by the required amount to transfer the required amount of memory from the unallocated local memory portion to the allocated local memory portion for allocation to the transformed process.

9. The apparatus of claim 6
CHARACTERIZED IN THAT
each second control means comprises
its own second table means (FIG. 3) including a first entry (201) for storing an indication of an amount of memory in the global memory portion and a second entry (202) for storing an indication of an amount of memory in the unallocated local memory portion, both memory portions being of the associated processor, and
operating system kernel means (31), responsive to a call for allocation of a first amount of memory to a process assigned to the associated processor, for examining (700-702) the second entry of the second table means to determine whether the unallocated local memory portion includes an amount of memory suitable to satisfy the call, for requesting (703-706) the first control means to transfer a second amount of memory from the global memory portion to the local memory portion if the unallocated local memory portion does not include a suitable amount of memory, and for decrementing (713-717) the second entry of the second table means by the first amount to transfer the first amount of memory from the unallocated local memory portion to the allocated local memory portion for allocation to the calling process, if the unallocated local memory portion includes a suitable amount of memory; and
the first control means comprises
first table means (FIG. 2) including a plurality of entries (212-213) each associated with a different one of the plurality of processors and for storing an indication of an amount of memory in the global memory portion of the associated processor, and
operating system process manager means (30), responsive to the transfer request, for decrementing (1100-1103) the first table means' entry associated with the processor of the requesting second control means by the smaller of the second amount and the full amount of memory in the global memory portion of the associated processor to initiate transfer of that smaller amount of memory from the global memory portion to the unallocated local memory portion, and notifying (1104-1105) the requesting second control means of the result;
the kernel means of the requesting second control means further responsive to the notice for both incrementing (710-711) the second entry and decrementing the first entry of the second ta-

ble means by the amount being transferred to complete transfer of that amount of memory from the global memory portion to the unallocated local memory portion, for examining (712) the second entry of the second table means to determine whether the unallocated local memory portion includes the first amount of memory, and for decrementing (712-717) the second entry by the first amount to transfer the first amount of memory from the unallocated local memory portion to the allocated local memory portion for allocation to the calling process if the unallocated local memory portion includes the first amount of memory.

10. The apparatus of claim 6
CHARACTERIZED IN THAT
each second control means comprises
its own second table means (FIG. 3) including a first entry (201) for storing an indication of an amount of memory in the global memory portion of the associated processor, and a second entry (202) for storing an indication of amount of memory in the unallocated local memory portion of the associated processor, and
operating system kernel means (31), responsive to a call for deallocation from a process assigned to the associated processor of a first amount of memory allocated thereto, for incrementing (800-802) the second entry of the second table means by the first amount to transfer the first amount of memory from the allocated local memory portion to the unallocated local memory portion to deallocate the first amount of memory from the process, thereafter for examining (804, 900-903) the second entry of the second table means to determine whether the unallocated local memory portion includes an amount of memory in excess of a predetermined amount, and for requesting (904, 907) the first control means to transfer a second amount of memory from the local memory portion to the global memory portion if the unallocated local memory portion exceeds the predetermined amount of memory; and
the first control means comprises
first table means including a plurality of entries (212-213) each associated with a different one of the plurality of processors and for storing an indication of an amount of memory in the global memory portion of the associated processor, and
operating system process managing means (30), responsive to the transfer request, for incrementing (910-911) the first table means' entry associated with the processor of the requesting second control means by the second amount to initiate transfer of that amount of memory from the unallocated local memory portion to the global memory portion and notifying (912-913) thereof the requesting second control means;
the kernel means of the requesting second control means (920-922) further responsive to the notice, for both decrementing the second entry and incrementing the first entry of the second table means by the second amount to complete transfer of that amount of memory from the unallocated local memory portion to the global memory portion.

11. A method of allocating memory to processes in a system (FIG. 1) having a bus connecting a plurality of processors (11-12) for executing processes that are assigned thereto, each processor having its own non-shared memory (22)
CHARACTERIZED BY the steps of
requesting (503; 602) a first control arrangement (30) to allocate to a process an initially-required amount of memory from a first global memory portion (42) of the memory of the one of said processors to which said process is assigned;
requesting (700; 800) a second control arrangement (31), dedicated to said processor to which said process is assigned, to allocate a required amount of memory from a second local memory portion (41) of its memory to said process; and
the first and second control arrangements cooperatively transferring memory as needed between the first and second memory portion of said processor (704-706, 710-711, 1100-1105, FIG. 9).

12. The method of claim 11
CHARACTERIZED IN THAT
the step of allocating the first amount of memory comprises the steps of
the first control arrangement assigning (611) the process to a first processor having at least the first amount of memory in the first memory portion,
the first control arrangement transferring (612-632) the first amount of memory from the first memory portion to the second memory portion of the first processor, and
in response to the assignment, the second control arrangement dedicated to the first processor allocating (650-657) to the process the first amount of memory from the second memory portion of the first processor, and
the step of requesting a second control arrangement to allocate a second amount of memory comprises the step of
requesting (700) the second control arrangement dedicated to the first processor to al-

locate to the process a second amount of memory, or

requesting (800) the second control arrangement to deallocate a third amount of memory allocated to a process which is assigned to the first processor; and

the step of the second control arrangement allocating memory comprises the steps of

in response to the allocation request, the second control arrangement allocating (701-717) to the process the second amount of memory from the second memory portion of the first processor, and

in response to the deallocation request, the second control arrangement deallocating (801-806) the third amount of memory from the process to the second memory portion of the first processor.

Patentansprüche

1. Mehrprozessoreinrichtung mit einem eine Mehrzahl von Prozessoren (11-12) verbindenden Bus zur Ausführung von den Prozessoren zugewiesenen Prozessen, wobei jeder Prozessor seinen eigenen nicht gemeinsam benutzten Speicher (22) aufweist,
gekennzeichnet dadurch,
daß jeder der Speicher logisch in einen ersten Globalteil (42) und einen zweiten Lokalteil (41) eingeteilt ist;
durch erste Steuermittel (30) zum Zuweisen zu einzelnen der Prozesse eines anfänglich erforderlichen Speicherumfangs aus dem besagten ersten Speicherteil im Prozessor, dem der einzelne der Prozesse zugewiesen wird; und
eine Mehrzahl von zweiten Steuermitteln (31), von denen jeweils ein anderes mit jedem Prozessor verbunden ist, jeweils zur Zuweisung von im zweiten Speicherteil des zugehörigen Prozessors enthaltenem Speicherraum zu dem besagten zugehörigen Prozessor zugewiesenen Prozessen;
wobei die ersten Steuermittel mit den zweiten Steuermitteln jedes Prozessors zusammenwirken, um nach Bedarf Speicherraum zwischen den ersten und zweiten mit dem zweiten Steuermittel verbundenen Speicherteilen zu übertragen.

2. Einrichtung nach Anspruch 1,
dadurch gekennzeichnet,
daß das erste Steuermittel für die Zuweisung (611) von Prozessen zu den Prozessoren bestimmt ist.

3. Einrichtung nach Anspruch 1,
dadurch gekennzeichnet,
daß das erste Steuermittel folgendes umfaßt:
Mittel (610-614, 630-632) zum Zuweisen von Prozessen zu den Prozessoren und weiterhin zum Übertragen von im ersten Speicherteil enthaltenem Speicherraum zum zweiten Speicherteil für jeden zugewiesenen Prozeß, wobei beide Teile zu dem Prozessor gehören, dem der Prozeß zugewiesen ist; und
jedes der zweiten Steuermittel
Mittel (650-657) zum Zuweisen von im zweiten Speicherteil enthaltenen Speicherraum zu dem zugehörigen Prozessor zugewiesenen Prozessen umfaßt.

4. Einrichtung nach Anspruch 1,
dadurch gekennzeichnet,
daß die ersten Steuermittel und die zweiten Steuermittel jedes Prozessors zusammenwirken, um Speicherraum zwischen dem ersten Speicherteil und dem zweiten Speicherteil des mit dem zweiten Steuermittel verbundenen Prozessors zu übertragen (Figuren 7, 9), auf Grundlage des im zweiten Teil enthaltenen und den Prozessen nicht zugewiesenen Speicherumfangs.

5. Einrichtung nach Anspruch 1,
dadurch gekennzeichnet,
daß jedes zweite Steuermittel weiterhin für die Freigabe (800-806) von zugewiesenem Speicherraum von Prozessen zum zweiten Speicherteil des zugehörigen Prozessors bestimmt ist; und
die ersten Steuermittel und die zweiten Steuermittel jedes Prozessors zusammenwirken, um Speicherraum vom zweiten Speicherteil zum ersten Speicherteil zu übertragen (Figur 9), wobei beide Teile zu dem mit den zusammenwirkenden zweiten Steuermitteln verbundenen Prozessor gehören, wenn ein Umfang nicht zugewiesenen Speicherraums im zweiten Speicherteil einen vorbestimmten Höchstwert überschreitet, und weiterhin um Speicherraum vom ersten Speicherteil zum zweiten Speicherteil zu übertragen (Figur 7), wenn ein Umfang von nicht zugewiesenem Speicherraum im zweiten Speicherteil um einen vorbestimmten Mindestwert überschritten wird.

6. Vorrichtung nach Anspruch 1,
dadurch gekennzeichnet,
daß jeder Speicher logisch in einen globalen, zentral verwalteten Speicherteil (42), einen lokalen, örtlich verwalteten Speicherteil (45), der keinen Prozessen zugewiesen ist, und einen lokalen, örtlich verwalteten Speicherteil (46), der Prozessen zugewiesen ist, eingeteilt ist;
das erste Steuermittel für die Zuweisung (611)

von Prozessen zu den Prozessoren und weiterhin zur Übertragung (612-614, 630-632) für jeden zugewiesenen Prozeß von im globalen Speicherteil enthaltenen Speicherraum zum nicht zugewiesenen lokalen Speicherteil des Prozessors, dem der Prozeß zugewiesen ist, bestimmt ist; und
die zweiten Steuermittel jeweils zur Übertragung (650-657) von Speicherraum vom nicht zugewiesenen lokalen Speicherteil zum zugewiesenen lokalen Speicherteil des zugehörigen Prozessors zur Zuweisung des übertragenen Speicherraums zu dem zugehörigen Prozessor zugewiesenen Prozessen und weiterhin zur Übertragung (Figur 8) von Speicherraum vom zugewiesenen lokalen Speicherteil zum nicht zugewiesenen lokalen Speicherteil des zugehörigen Prozessors zur Freigabe des übertragenen Speicherraums von dem zugehörigen Prozessor zugewiesenen Prozessen bestimmt sind;
wobei die ersten Steuermittel und die zweiten Steuermittel zusammenwirken (Figuren 5-9), um Speicherraum zwischen dem globalen Speicherteil und dem nicht zugewiesenen lokalen Speicherteil zu übertragen, wobei beide Speicherteile zu dem mit den zweiten Steuermitteln verbundenen Prozessor gehören, als Reaktion auf das Auftreten von vorbestimmten Zuständen.

7. Einrichtung nach Anspruch 6,
dadurch gekennzeichnet,
daß jedes zweite Steuermittel folgendes umfaßt:
sein eigenes zweites Tabellenmittel einschließlich eines ersten Eintrags (201) zum Speichern einer Anzeige eines im globalen Speicherteil des zugehörigen Prozessors enthaltenen Speicherumfangs und eines zweiten Eintrags (202) zum Speichern einer Anzeige eines im nicht zugewiesenen lokalen Speicherteil des zugehörigen Prozessors enthaltenen Speicherumfangs, und
auf einen Aufruf zur Duplizierung eines dem mit dem zweiten Steuermittel verbundenen Prozessors zugewiesenen Prozesses reagierende Betriebssystemkernmittel (31) zum Untersuchen (500-502) des ersten Eintrags des zweiten Tabellenmittels zur Bestimmung, ob der globale Speicherteil einen vom duplizierten Prozeß benötigten Speicherumfang enthält und zum Anfordern (503-504), daß das erste Steuermittel die Duplizierung zuläßt, wenn der globale Speicherteil den benötigten Speicherumfang enthält;
das erste Steuermittel folgendes umfaßt:
erste Tabellenmittel einschließlich einer Mehrzahl von jeweils mit einem anderen der Mehrzahl von Prozessoren verbundenen Einträgen (212-213) zum Speichern einer Anzeige eines im globalen Speicherteil des zugehörigen Prozessors enthaltenen Speicherumfangs, und
auf die Anforderung zum Zulassen der Duplizierung reagierende Betriebssystemprozeßverwaltungsmittel (30) zum Untersuchen (510, 511) des mit dem Prozessor des anfordernden zweiten Steuermittels verbundenen Eintrags im ersten Tabellenmittel zur Bestimmung, ob der globale Speicherteil des Prozessors den benötigten Speicherumfang enthält, und zum sowohl (a) Erniedrigen (512-513) des mit dem Prozessor verbundenen Eintrags im ersten Tabellenmittel um die erforderliche Größe zur Einleitung der Übertragung des erforderlichen Speicherumfangs vom globalen Speicherteil zum nicht zugewiesenen Speicherteil des Prozessors als auch (b) Benachrichtigen (514) des zweiten Steuermittels von der Gewährung der Anforderung nach Duplizierung, wenn der globale Speicherteil den benötigten Speicherumfang enthält; und
das Kernmittel des anfordernden zweiten Steuermittels weiterhin auf die Gewährungsbenachrichtigung reagiert, um sowohl (a) den zweiten Eintrag um die benötigte Größe zu erhöhen (520-522) und den ersten Eintrag im zweiten Tabellenmittel um die benötigte Größe zu erniedrigen, um die Übertragung des benötigten Speicherumfangs vom globalen Speicherteil zum nicht zugewiesenen lokalen Speicherteil zu vollenden, als auch (b) den zweiten Eintrag im zweiten Tabellenmittel um die benötigte Größe zu erniedrigen (523-527), um den benötigten Speicherumfang zur Zuweisung zum duplizierten Prozeß vom nicht zugewiesenen lokalen Speicherteil zum zugewiesenen lokalen Speicherteil zu übertragen.

8. Einrichtung nach Anspruch 6,
dadurch gekennzeichnet,
daß das erste Steuermittel folgendes umfaßt:
erste Tabellenmittel einschließlich einer Mehrzahl von jeweils mit einem anderen der Mehrzahl von Prozessoren verbundenen Einträgen (212-213) zum Speichern einer Anzeige eines Speicherumfangs im globalen Speicherteil des zugehörigen Prozessors, und
auf einen Aufruf von einem aufrufenden Prozessor zur Transformation eines Prozesses reagierende Betriebssystemprozeßverwaltungsmittel (30) zum Auswählen (610-611) eines Prozessors zur Zuweisung zum transformierten Prozeß, wobei der ausgewählte Prozessor zur Mehrzahl von Prozessoren gehört und einen globalen Speicherteil aufweist, der einen vom transformierten Prozeß benötigten Speicherumfang enthält, und zum sowohl (a) Erniedrigen (612) des mit dem ausgewählten Prozessor verbundenen Eintrags im ersten Tabellenmittel um die erforderliche Größe zur Einleitung der Übertragung des erforderlichen Speicherumfangs vom globalen Speicherteil zum nicht zugewiesenen lokalen Spei-

cherteil des ausgewählten Prozessors als auch (b) Benachrichtigen (613-614) des aufrufenden Prozessors von der Auswahl; und
wobei jedes zweite Steuermittel folgendes umfaßt:
sein eigenes zweites Tabellenmittel einschließlich eines ersten Eintrags (201) zum Speichern einer Anzeige eines Speicherumfangs im globalen Speicherteil des zugehörigen Prozessors und eines zweiten Eintrags (202) zum Speichern einer Anzeige eines Speicherumfangs im nicht zugewiesenen örtlichen Speicherteil des zugehörigen Prozessors, und auf Benachrichtigung vom aufrufenden Prozessor über die Auswahl des zugehörigen Prozessors reagierende Betriebssystemkernmittel (31) zum sowohl (a) Erhöhen (650-652) des zweiten Eintrags und Erniedrigen des ersten Eintrags im zweiten Tabellenmittel um die erforderliche Größe zur Vollendung der Übertragung des benötigten Speicherumfangs vom globalen Speicherteil zum nicht zugewiesenen lokalen Speicherteil, als auch (b) Erniedrigen (653-657) des zweiten Eintrags im zweiten Tabellenmittel um die benötigte Größe zur Übertragung des erforderlichen Speicherumfangs vom nicht zugewiesenen lokalen Speicherteil zum zugewiesenen lokalen Speicherteil für Zuweisung zum transformierten Prozeß.

9. Einrichtung nach Anspruch 6,
dadurch gekennzeichnet,
daß jedes zweite Steuermittel folgendes umfaßt:
sein eigenes zweites Tabellenmittel (Figur 3) einschließlich eines ersten Eintrags (201) zum Speichern einer Anzeige eines Speicherumfangs im globalen Speicherteil und eines zweiten Eintrags (202) zum Speichern einer Anzeige eines Speicherumfangs im nicht zugewiesenen lokalen Speicherteil, wobei beide Speicherteile zum zugehörigen Prozessor gehören, und
auf einen Aufruf zur Zuweisung eines ersten Speicherumfangs zu einem dem zugehörigen Prozessor zugewiesenen Prozeß reagierende Betriebssystemkernmittel (31) zum Untersuchen (700-702) des zweiten Eintrags des zweiten Tabellenmittels zur Bestimmung, ob der nicht zugewiesene lokale Speicherteil einen zum Erfüllen des Aufrufs geeigneten Speicherumfang enthält, zum Anfordern (703-706), daß das erste Steuermittel einen zweiten Speicherumfang vom globalen Speicherteil zum lokalen Speicherteil überträgt, wenn der nicht zugewiesene lokale Speicherteil nicht einen geeigneten Speicherumfang enthält, und zum Erniedrigen (713-717) des zweiten Eintrags im zweiten Tabellenmittel um die erste Größe zur Übertragung des ersten Speicherumfangs vom nicht zugewiesenen lokalen Speicherteil zum zugewiesenen lokalen Speicherteil zur Zuweisung zum aufrufenden Prozeß, wenn der nicht zugewiesene lokale Speicherteil einen geeigneten Speicherumfang enthält; und
das erste Steuermittel folgendes umfaßt:
erste Tabellenmittel (Figur 2) einschließlich einer Mehrzahl von jeweils mit einem anderen der Mehrzahl von Prozessoren verbundenen Einträgen (212-213) zum Speichern einer Anzeige eines Speicherumfangs im globalen Speicherteil des zügehörigen Prozessors, und
auf die Übertragungsanforderung reagierende Betriebssystemprozeßverwaltermittel (30) zum Erniedrigen (1100-1103) des mit dem Prozessor des anfragenden zweiten Steuermittels verbundenen Eintrags des ersten Tabellenmittels um den kleineren der zweiten Größe und des vollen Speicherumfangs im globalen Speicherteil des zugehörigen Prozessors zur Einleitung der Übertragung dieses kleineren Speicherumfangs vom globalen Speicherteil zum nicht zugewiesenen lokalen Speicherteil und Benachrichtigung (1104-1105) des anfordernden zweiten Steuermittels von dem Ergebnis;
das Kernmittel des anfordernden zweiten Steuermittels weiterhin auf die Benachrichtigung sowohl zur Erhöhung (710-711) des zweiten Eintrags als auch Erniedrigung des ersten Eintrags im zweiten Tabellenmittel um die übertragene Größe reagiert, um die Übertragung dieses Speicherumfangs vom globalen Speicherteil zum nicht zugewiesenen Speicherteil zu vollenden, um den zweiten Eintrag des zweiten Tabellenmittels zu untersuchen (712), um zu bestimmen, ob der nicht zugewiesene lokale Speicherteil den ersten Speicherumfang enthält, und zum Erniedrigen (712-717) des zweiten Eintrags um die erste Größe zur Übertragung des ersten Speicherumfangs vom nicht zugewiesenen lokalen Speicherteil zum zugewiesenen lokalen Speicherteil zur Zuweisung zum aufrufenden Prozeß, wenn der nicht zugewiesene lokale Speicherteil den ersten Speicherumfang enthält.

10. Einrichtung nach Anspruch 6,
dadurch gekennzeichnet,
daß jedes zweite Steuermittel folgendes umfaßt:
sein eigenes zweites Tabellenmittel (3) einschließlich eines ersten Eintrags (201) zum Speichern einer Anzeige eines Speicherumfangs im globalen Speicherteil des zugehörigen Prozessors und einen zweiten Eintrag (202) zum Speichern einer Anzeige des Speicherumfangs im nicht zugewiesenen lokalen Speicherteil des zugehörigen Prozessors, und
auf einen Aufruf zur Freigabe von einem dem zugehörigen Prozessor Zugewiesenen Prozeß eines diesem zugewiesenen ersten Speicherumfangs reagierende Betriebssystemkernmittel

(31) zum Erhöhen (800-802) des zweiten Eintrags im zweiten Tabellenmittel um die erste Größe zur Übertragung des ersten Speicherumfangs vom zugewiesenen lokalen Speicherteil zum nicht zugewiesenen lokalen Speicherteil zur Freigabe des ersten Speicherumfangs vom Prozeß, danach zur Untersuchung (804, 900-903) des zweiten Eintrags im zweiten Tabellenmittel zur Bestimmung, ob der nicht zugewiesene lokale Speicherteil einen eine vorbestimmte Größe überschreitenden Speicherumfang enthält, und zum Anfordern (904, 907), daß das erste Steuermittel einen zweiten Speicherumfang vom lokalen Speicherteil zum globalen Speicherteil überträgt, wenn der nicht zugewiesene lokale Speicherteil den vorbestimmten Speicherumfang überschreitet; und
wobei das erste Steuermittel folgendes umfaßt:
erste Tabellenmittel einschließlich einer Mehrzahl von jeweils mit einem anderen der Mehrzahl von Prozessoren verbundenen Einträgen (212-213) zum Speichern einer Anzeige eines Speicherumfangs im globalen Speicherteil des zugehörigen Prozessors, und
auf die Übertragungsanforderung reagierende Betriebssystemprozeßverwaltungsmittel (30) zum Erhöhen (910-911) des mit dem Prozessor des anfordernden zweiten Steuermittels verbunden Eintrags im ersten Tabellenmittel um die zweite Größe zur Einleitung der Übertragung dieses Speicherumfangs vom nicht zugewiesenen lokalen Speicherteil zum globalen Speicherteil und Benachrichtigung (912-913) des anfordernden zweiten Steuermittels darüber;
das Kernmittel des anfordernden zweiten Steuermittels (920-922) weiterhin sowohl zur Erniedrigung des zweiten Eintrags als auch Erhöhung des ersten Eintrags im zweiten Tabellenmittel um die zweite Größe reagiert, um die Übertragung dieses Speicherumfangs vom nicht zugewiesenen lokalen Speicherteil zum globalen Speicherteil zu vollenden.

11. Verfahren zur Zuweisung von Speicherraum zu einem Prozeß in einem System (Figur 1) mit einer Mehrzahl von Prozessoren (11-12), die jeweils ihren eigenen nicht gemeinsam genutzten logisch in einen ersten (42) und einen zweiten Teil (51) eingeteilten Speicher (22) aufweisen, gekennzeichnet durch folgende Schritte:
Anfordern (503; 602) einer ersten Steueranordnung (30) zum Zuweisen von in den ersten Speicherteilen enthaltenen Speicherraum zu den Prozessen zur Zuweisung eines ersten Speicherumfangs zu einem Prozeß;
als Reaktion auf die Anforderung, Zuweisen (510-527; 610-675) zum Prozeß des ersten Speicherumfangs von einem ersten Speicherteil eines ersten Prozessors durch die erste Steueranordnung;
Anfordern (700; 800), daß eine dem ersten Prozessor fest zugeordnete zweite Steueranordnung (31) zur Zuweisung von im zweiten Speicherteil des ersten Prozessors enthaltenen Speicherraum zu Prozessen dem Prozeß eine zweiten Speicherumfang zuweist;
als Reaktion auf die Anforderung, Zuweisen (701-717; 801-806) zum Prozeß des zweiten Speicherumfangs vom zweiten Speicherteil des ersten Prozessors durch die zweite Steueranordnung; und
mitwirkendes Übertragen von Speicherraum zwischen dem ersten und zweiten Speicherteil des ersten Prozessors (704-706, 710-711, 1100-1105, Figur 9) durch die erste und zweite Steueranordnung.

12. Verfahren nach Anspruch 11,
dadurch gekennzeichnet,
daß der Schritt des Zuweisens des ersten Speicherumfangs folgende Schritte umfaßt:
Zuweisen (611) des Prozesses zu einem ersten Prozessor mit mindestens dem ersten Speicherumfang im ersten Speicherteil durch die erste Steueranordnung,
Übertragen (612-632) des ersten Speicherumfangs vom ersten Speicherteil zum zweiten Speicherteil des ersten Prozessors durch die erste Steueranordnung, und
als Reaktion auf die Zuweisung, Zuweisen (650-657) zum Prozeß des ersten Speicherumfangs vom zweiten Speicherteil des ersten Prozessors durch die dem ersten Prozessor fest zugeordnete zweite Steueranordnung; und der Schritt des Anforderns, daß eine zweite Steueranordnung einen zweiten Speicherumfang zuweist, folgenden Schritt umfaßt:
Anfordern (700) des Zuweisens eines zweiten Speicherumfangs zum Prozeß durch die dem ersten Prozessor fest zugeordnete zweite Steueranordnung, oder
Anfordern (800), daß die zweite Steueranordnung einen einem dem ersten Prozessor zugewiesenen Prozeß zugewiesenen dritten Speicherumfang freigibt; und
der Schritt des Zuweisens von Speicherraum durch die zweite Steueranordnung folgende Schritte umfaßt:
als Reaktion auf die Zuordnungsanforderung, Zuweisen (701-717) zum Prozeß des zweiten Speicherumfangs vom zweiten Speicherteil des ersten Prozessors durch die zweite Steueranordnung, und
als Reaktion auf die Freigabeanforderung, Freigeben (801-806) des dritten Speicherumfangs vom Prozeß zum zweiten Speicherteil des ersten

Prozessors durch die zweite Steueranordnung.

Revendications

1. Appareil multiprocesseur ayant un bus connectant une pluralité de processeurs (11-12) pour exécuter des procédés qui sont assignés aux processeurs, chaque processeur ayant sa propre mémoire non partagée (22)
CARACTERISE PAR
chacune des mémoires étant logiquement divisée en une première partie globale (42) et une seconde partie locale (41);
un premier moyen de commande (30) pour assigner à des procédés individuels une quantité initialement requise de mémoire de ladite première partie de mémoire dans le processeur auquel le procédé individuel est assigné; et
une pluralité de seconds moyens de commande (31), un moyen différent étant associé à chaque processeur, chacun pour assigner de la mémoire incluse dans la seconde partie de mémoire du processeur associé à des procédés assignés audit processeur associé;
le premier moyen de commande coopérant avec le second moyen de commande de n'importe quel processeur pour transférer de la mémoire au fur et à mesure des besoins entre la première partie de mémoire et la seconde partie de mémoire associée aux seconds moyens de commande.

2. Appareil de la revendication 1
CARACTERISE EN CE QUE
le premier moyen de commande sert à assigner (611) des procédés aux processeurs.

3. Appareil de la revendication 1
CARACTERISE EN CE QUE
le premier moyen de commande comprend
un moyen (610-614, 630-632) pour assigner des procédés aux processeurs et en outre transférer de la mémoire incluse dans la première partie de mémoire à la seconde partie de mémoire pour chaque procédé assigné, les deux parties appartenant au processeur auquel le procédé est assigné; et
chacun des seconds moyens de commande comprend
un moyen (650-657) pour assigner de la mémoire incluse dans la seconde partie de mémoire à des procédés assignés au processeur associé.

4. Appareil de la revendication 1
CARACTERISE EN CE QUE
le premier moyen de commande et le second moyen de commande de n'importe quel processeur coopèrent pour transférer (figures 7,9) de la mémoire entre la première partie de mémoire et la seconde partie de mémoire du processeur associé au second moyen de commande, en fonction de la quantité de mémoire incluse dans la seconde partie et non assignée aux procédés.

5. Appareil de la revendication 1
CARACTERISE EN CE QUE
chaque second moyen de commande sert en outre à désassigner (800-806) de la mémoire assignée à des procédés à la seconde partie de mémoire du processeur associé; et
le premier moyen de commande et le second moyen de commande de n'importe quel processeur coopèrent pour transférer (figure 9) de la mémoire depuis la seconde partie de mémoire vers la première partie de mémoire, les deux parties appartenant au processeur associé au second moyen de commande coopérant, quand une quantité de mémoire non assignée dans la seconde partie de mémoire dépasse un maximum prédéterminé, et en outre transférer (figure 7) de la mémoire depuis la première partie de mémoire vers la seconde partie de mémoire quand une quantité de mémoire non assignée dans la seconde partie de mémoire est dépassée par un minimum prédéterminé.

6. Appareil de la revendication 1
CARACTERISE EN CE QUE
chaque mémoire est divisée logiquement en une partie de mémoire globale, gérée centralement (42), une partie de mémoire locale, gérée localement, non assignée à des procédés (45), et une partie de mémoire locale, gérée localement, assignée à des procédés (46);
le premier moyen de commande sert à assigner (611) des procédés aux processeurs et en outre à transférer (612-614, 630-632), pour chaque procédé assigné, de la mémoire incluse dans la partie de mémoire globale à la partie de mémoire locale non assignée du processeur auquel le procédé est assigné; et
les seconds moyens de commande servent chacun à transférer (650-657) de la mémoire depuis la partie de mémoire locale non assignée vers la partie de mémoire locale assignée du processeur associé, de façon à assigner la mémoire transférée à des procédés assignés au processeur associé, et en outre à transférer (figure 8) de la mémoire depuis la partie de mémoire locale assignée vers la partie de mémoire locale non assignée du processeur associé, de façon à désassigner la mémoire transférée depuis des procédés assignés au processeur associé;

le premier moyen de commande et les seconds moyens de commande coopérant (figures 5 à 9) pour transférer de la mémoire entre la partie de mémoire globale et la partie de mémoire locale non assignée, les deux parties de mémoire appartenant au processeur associé au second moyen de commande, en réponse à la présence de conditions prédéterminées.

7. Appareil de la revendication 6
CARACTERISE EN CE QUE
chaque second moyen de commande comprend
son propre second moyen de table comportant une première entrée (201) pour mémoriser une indication d'une quantité de mémoire incluse dans la partie de mémoire globale du processeur associé et une seconde entrée (202) pour mémoriser une indication d'une quantité de mémoire incluse dans la partie de mémoire locale non assignée du processeur associé, et
un moyen de noyau de système d'exploitation (31), sensible à un appel de reproduction d'un procédé assigné au processeur associé au second moyen de commande, pour examiner (500-502) la première entrée du second moyen de table afin de déterminer si la partie de mémoire globale comporte ou non une quantité de mémoire requise par le procédé reproduit, et pour demander (503-504) au premier moyen de commande de permettre la reproduction si la partie de mémoire globale comporte la quantité requise de mémoire;
le premier moyen de commande comprend
un premier moyen de table comportant une pluralité d'entrées (212-213) associées chacune à un processeur différent de la pluralité de processeurs et servant à mémoriser une indication d'une quantité de mémoire incluse dans la partie de mémoire globale du processeur associé, et
un moyen de gestion de procédés de système d'exploitation (30), sensible à la demande d'autorisation de reproduction, pour examiner (510,511) l'entrée du premier moyen de table associée au processeur du second moyen de commande demandeur de façon à déterminer si la partie de mémoire globale du processeur comporte ou non la quantité requise de mémoire, et pour à la fois (a) décrémenter (512-513) l'entrée associée du processeur dans le premier moyen de table par la quantité requise de façon à déclencher le transfert de la quantité requise de mémoire depuis la partie de mémoire globale vers la partie de mémoire locale non assignée du processeur et (b) notifier (514) le second moyen de commande d'un accord à la demande de reproduction, si la partie de mémoire globale comporte la quantité requise de mémoire; et
le moyen de noyau du second moyen de commande demandeur étant en outre sensible à l'avis de l'accord, pour à la fois (a) incrémenter (520-522) la seconde entrée et décrémenter la première entrée du second moyen de table par la quantité requise de façon à terminer le transfert de la quantité requise de mémoire depuis la partie de mémoire globale vers la partie de mémoire locale non assignée, et (b) décrémenter (523-527) la seconde entrée du second moyen de table par la quantité requise pour transférer la quantité requise de mémoire depuis la partie de mémoire locale non assignée vers la partie de mémoire locale assignée pour l'assignation au procédé reproduit.

8. Appareil de la revendication 6
CARACTERISE EN CE QUE
le premier moyen de commande comprend
un premier moyen de table comportant une pluralité d'entrées (212-213) associées chacune à un processeur différent de la pluralité de processeurs et servant à mémoriser une indication d'une quantité de mémoire incluse dans la partie de mémoire globale du processeur associé et
un moyen de gestion de procédés de système d'exploitation (30), sensible à un appel d'un processeur demandeur pour la transformation d'un procédé, pour sélectionner (610-611) un processeur pour l'assignation au procédé transformé, le processeur sélectionné appartenant à la pluralité de processeurs et ayant une partie de mémoire globale qui comporte une quantité de mémoire requise par le procédé transformé, et pour à la fois (a) décrémenter (612) l'entrée du premier moyen de table associée au processeur associé par la quantité requise de façon à déclencher le transfert de la quantité requise de mémoire depuis la partie de mémoire globale vers la partie de mémoire locale non assignée du processeur sélectionné et (b) notifier (613-614) le processeur demandeur de la sélection; et
chaque second moyen de commande comprend
son propre second moyen de table comportant une première entrée (201) pour mémoriser une indication d'une quantité de mémoire incluse dans la partie de mémoire globale du processeur associé et une seconde entrée (202) pour mémoriser une indication d'une quantité de mémoire dans la partie de mémoire locale non assignée du processeur associé, et
un moyen de noyau de système d'exploitation (31), sensible à un avis provenant du pro-

cesseur demandeur de la sélection du processeur associé, pour à la fois (a) incrémenter (650-652) la seconde entrée et décrémenter la première entrée du second moyen de table par la quantité requise de façon à terminer le transfert de la quantité requise de mémoire depuis la partie de mémoire globale vers la partie de mémoire locale non assignée, et (b) décrémenter (653-657) la seconde entrée du second moyen de table par la quantité requise de façon à transférer la quantité requise de mémoire depuis la partie de mémoire locale non assignée vers la partie de mémoire locale assignée pour l'assignation au procédé transformé.

9. Appareil de la revendication 6
CARACTERISE EN CE QUE
chaque second moyen de commande comprend
son propre second moyen de table (figure 3) comportant une première entrée (201) pour mémoriser une indication d'une quantité de mémoire dans la partie de mémoire globale et une seconde entrée (202) pour mémoriser une indication d'une quantité de mémoire dans la partie de mémoire locale non assignée, les deux parties de mémoire appartenant au processeur associé, et
un moyen de noyau de système d'exploitation (31), sensible à un appel d'assignation d'une première quantité de mémoire à un procédé assigné au processeur associé, pour examiner (700-702) la seconde entrée du second moyen de table afin de déterminer si la partie de mémoire locale non assignée comporte ou non une quantité de mémoire pouvant satisfaire l'appel, pour demander (703-706) au premier moyen de commande de transférer une seconde quantité de mémoire depuis la partie de mémoire globale vers la partie de mémoire locale si la partie de mémoire locale non assignée ne comporte pas une quantité convenable de mémoire, et pour décrémenter (713-717) la seconde entrée du second moyen de table par la première quantité pour transférer la première quantité de mémoire depuis la partie de mémoire locale non assignée vers la partie de mémoire locale assignée pour l'assignation au procédé demandeur, si la partie de mémoire locale non assignée comporte une quantité convenable de mémoire; et
le premier moyen de commande comprend
un premier moyen de table (figure 2) comportant une pluralité d'entrées (212-213) associées chacune à un processeur différent de la pluralité de processeurs et servant à mémoriser une indication d'une quantité de mémoire incluse dans la partie de mémoire globale du processeur associé, et
un moyen de gestion de procédés de système d'exploitation (30), sensible à la demande de transfert, pour décrémenter (1100-1103) l'entrée du premier moyen de table associé au processeur du second moyen de commande demandeur par la plus petite de la seconde quantité et de la quantité complète de mémoire dans la partie de mémoire globale du processeur associé de façon à déclencher le transfert de cette plus petite quantité de mémoire depuis la partie de mémoire globale vers la partie de mémoire locale non assignée du processeur et, notifier (1104-1105) le second moyen de commande demandeur du résultat;
le moyen de noyau du second moyen de commande demandeur étant en outre sensible à l'avis pour à la fois incrémenter (710-711) la seconde entrée et décrémenter la première entrée du second moyen de table par la quantité transférée de façon à terminer le transfert de cette quantité de mémoire depuis la partie de mémoire globale vers la partie de mémoire locale non assignée, pour examiner (712) la seconde entrée du second moyen de tableau pour déterminer si la partie de mémoire locale non assignée comporte ou non la première quantité de mémoire, et pour décrémenter (712-717) la seconde entrée par la première quantité pour transférer la première quantité de mémoire depuis la partie de mémoire locale non assignée vers la partie de mémoire locale assignée pour l'assignation au procédé demandeur si la partie de mémoire locale non assignée comporte la première quantité de mémoire.

10. Appareil de la revendication 6
CARACTERISE EN CE QUE
chaque second moyen de commande comprend
son propre second moyen de table (figure 3) comportant une première entrée (201) pour mémoriser une indication d'une quantité de mémoire dans la partie de mémoire globale du processeur associé, et une seconde entrée (202) pour mémoriser une indication d'une quantité de mémoire dans la partie de mémoire locale non assignée du processeur associé, et
un moyen de noyau de système d'exploitation (31), sensible à un appel de désassignation d'un procédé assigné au processeur associé d'une première quantité de mémoire assignée à celui-ci, pour incrémenter (800-802) la seconde entrée du second moyen de table par la première quantité de façon à transférer la première quantité de mémoire depuis la partie de mémoire locale assignée vers la partie de mémoire locale non assignée de façon à désassigner la première quantité de mémoire du procédé, puis pour exa-

miner (804, 900-903) la seconde entrée du second moyen de table de façon à déterminer si la partie de mémoire locale non assignée comporte ou non une quantité de mémoire dépassant une quantité prédéterminée, et pour demander (904,907) au premier moyen de commande de transférer une seconde quantité de mémoire depuis la partie de mémoire locale vers la partie de mémoire globale si la partie de mémoire locale non assignée dépasse la quantité prédéterminée de mémoire; et

le premier moyen de commande comprend

un premier moyen de table comportant une pluralité d'entrées (212-213) associées chacune à un processeur différent de la pluralité de processeurs et servant à mémoriser une indication d'une quantité de mémoire dans la partie de mémoire globale du processeur associé, et

un moyen de gestion de procédés de système d'exploitation (30), sensible à la demande de transfert, pour incrémenter (910,911) l'entrée du premier moyen de table associé au processeur du second moyen de commande demandeur par la seconde quantité de façon à déclencher le transfert de cette quantité de mémoire depuis la partie de mémoire locale non assigné vers la partie de mémoire globale et pour en notifier (912-913) le second moyen de commande demandeur;

le moyen de noyau du second moyen de commande demandeur (920-922) étant en outre sensible à l'avis, pour à la fois décrémenter la seconde entrée et incrémenter la première entrée du second moyen de table par la seconde quantité de façon à terminer le transfert de cette quantité de mémoire depuis la partie de mémoire locale non assignée vers la partie de mémoire globale.

11. Procédé d'assignation de mémoire à un procédé dans un système (figure 1) ayant une pluralité de processeurs (11-12) ayant chacun sa propre mémoire (22) non partagée, divisée logiquement en une première (42) et une deuxième (41) partie, le procédé étant caractérisé par les étapes de:

demande (503;602) à un premier dispositif de commande (30), servant à assigner à des procédés de la mémoire incluse dans les premières parties de mémoire, d'assigner à un procédé une première quantité de mémoire;

en réponse à la demande, assignation par le premier dispositif de commande (510-527; 610-657) au procédé de la première quantité de mémoire depuis une première partie de mémoire d'un premier processeur;

demande (700;800) à un second dispositif de commande (31), dédié au premier processeur et servant à assigner à des procédés de la mémoire incluse dans la seconde partie de mémoire du premier processeur, d'assigner au procédé une seconde quantité de mémoire;

en réponse à la demande, assignation par le second dispositif de commande (701-717; 801-806) au procédé de la seconde quantité de mémoire depuis la seconde partie de mémoire du premier processeur; et

transfert par les premier et second dispositifs de commande de façon coopérative de mémoire entre les première et seconde parties de mémoire du premier processeur (704-706, 710-711, 1100-1105, figure 9).

12. Procédé de la revendication 11

CARACTERISE EN CE QUE

l'opération d'assignation de la première quantité de mémoire comprend les opérations de

assignation (611) par le premier dispositif de commande du procédé à un premier processeur ayant au moins la première quantité de mémoire dans la première partie de mémoire,

transfert (612-632) par le premier dispositif de commande de la première quantité de mémoire depuis la première partie de mémoire vers la seconde partie de mémoire du premier processeur, et

en réponse à l'assignation, l'assignation (650-657) par le second dispositif de commande dédié au premier processeur au procédé de la première quantité de mémoire depuis la seconde partie de mémoire du premier processeur; et

l'opération de demande à un second dispositif de commande d'assigner une seconde quantité de mémoire comprend les opérations de

demande (700) au second dispositif de commande dédié au premier processeur d'assigner au procédé une seconde quantité de mémoire, ou

demande (800) au second dispositif de commande de désassigner une troisième quantité de mémoire assignée à un procédé qui est assigné au premier processeur, et

l'opération d'assignation de mémoire par le second dispositif de commande comprend les opérations de

en réponse à la demande d'assignation, l'assignation (701-717) par le second dispositif de commande au procédé de la seconde quantité de mémoire depuis la seconde partie de mémoire du premier processeur, et

en réponse à la demande de désassignation, la désassignation (801-806) par le second dispositif de commande de la troisième quantité de mémoire du procédé vers la seconde partie de mémoire du premier processeur.

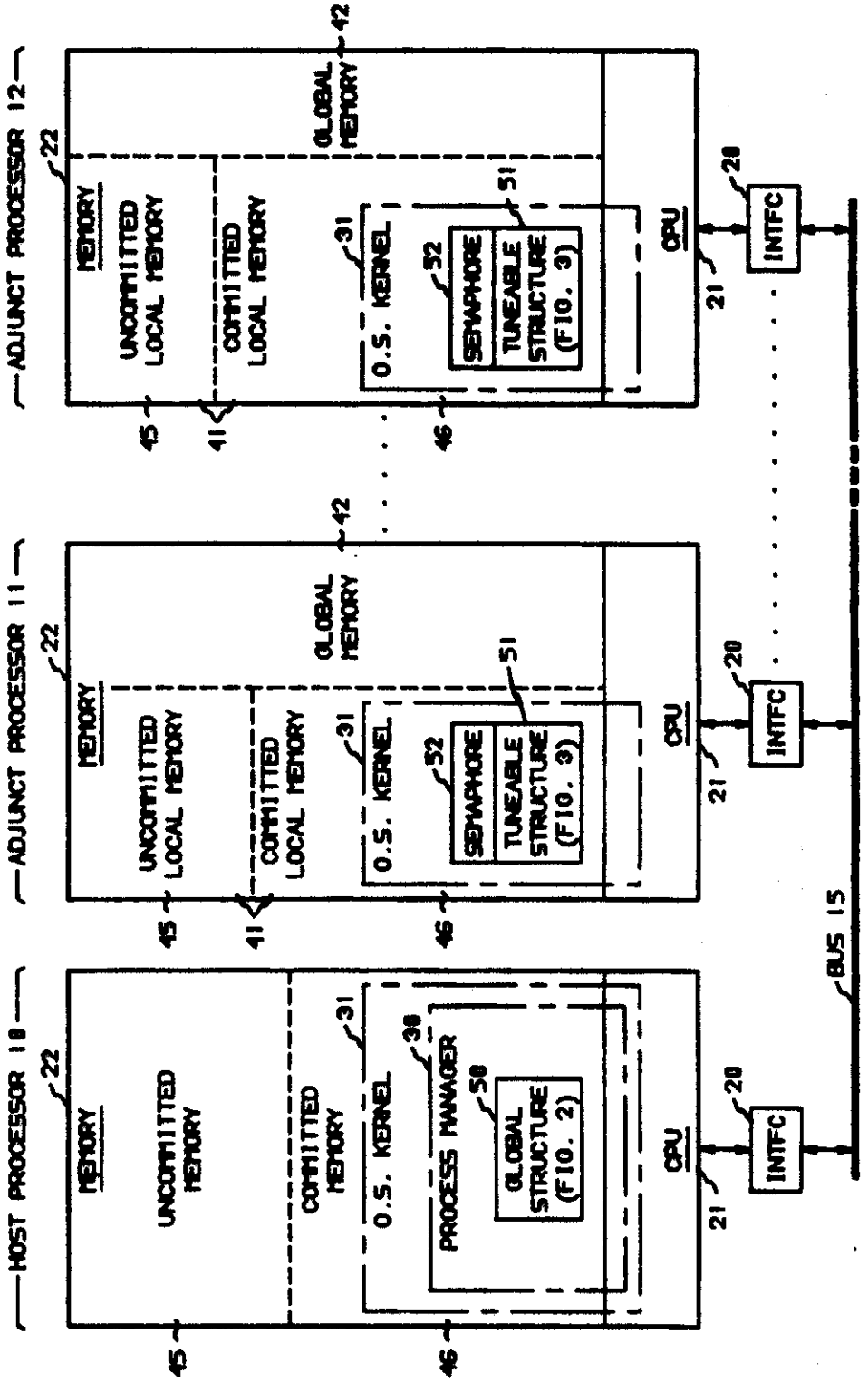

HOST PROCESSOR 10
ADJUNCT PROCESSOR 11
ADJUNCT PROCESSOR 12
MEMORY
UNCOMMITTED MEMORY
COMMITTED MEMORY
UNCOMMITTED LOCAL MEMORY
COMMITTED LOCAL MEMORY
GLOBAL MEMORY
O.S. KERNEL
PROCESS MANAGER
GLOBAL STRUCTURE (FIG. 2)
SEMAPHORE
TUNEABLE STRUCTURE (FIG. 3)
CPU
INTFC
BUS 15
22
31
30
50
42
51
52
20
21
28
45
41
46

FIG. 1

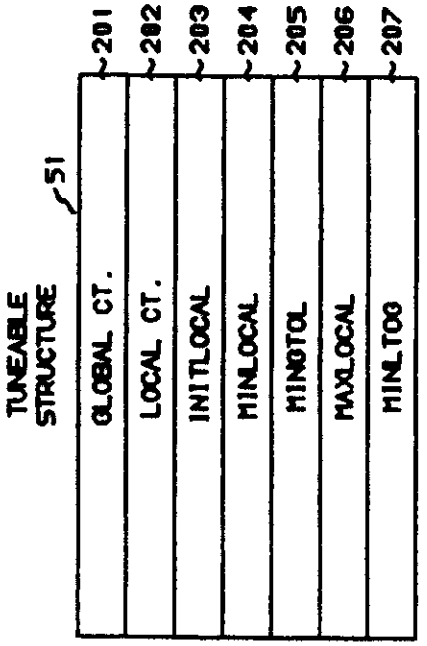

TUNEABLE STRUCTURE
51
GLOBAL CT.
LOCAL CT.
INITLOCAL
MINLOCAL
MINGTOL
MAXLOCAL
MINLTOO
201
202
203
204
205
206
207

FIG. 3

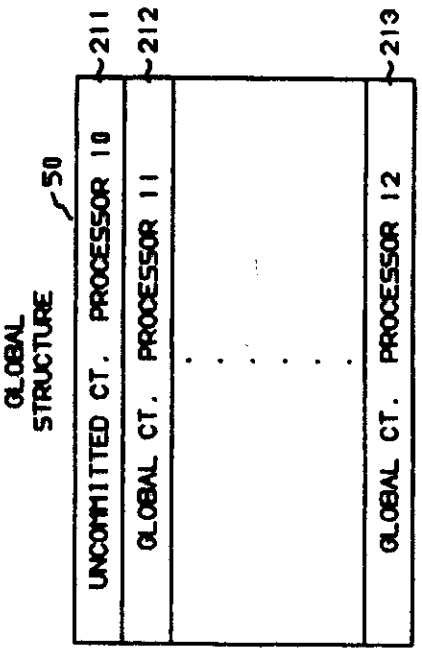
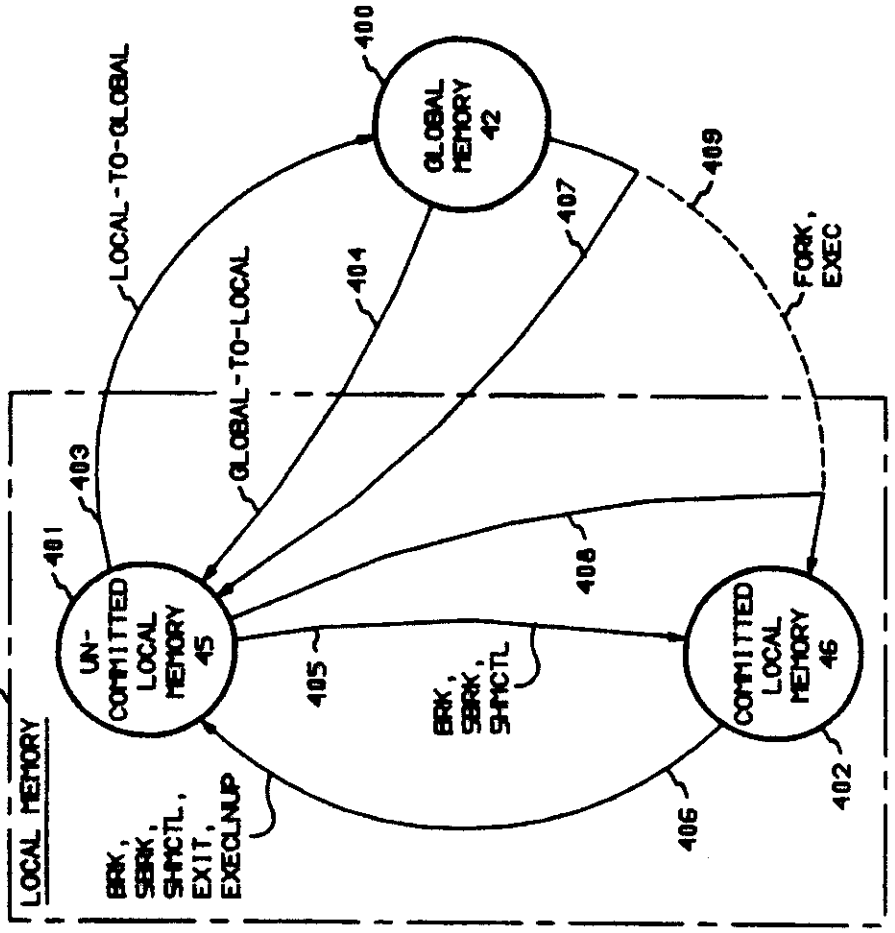

GLOBAL STRUCTURE
50
UNCOMMITTED CT. PROCESSOR 10
GLOBAL CT. PROCESSOR 11
GLOBAL CT. PROCESSOR 12
211
212
213

FIG. 2



MEMORY STATE DIAGRAM

FIG. 4

FIG. 5

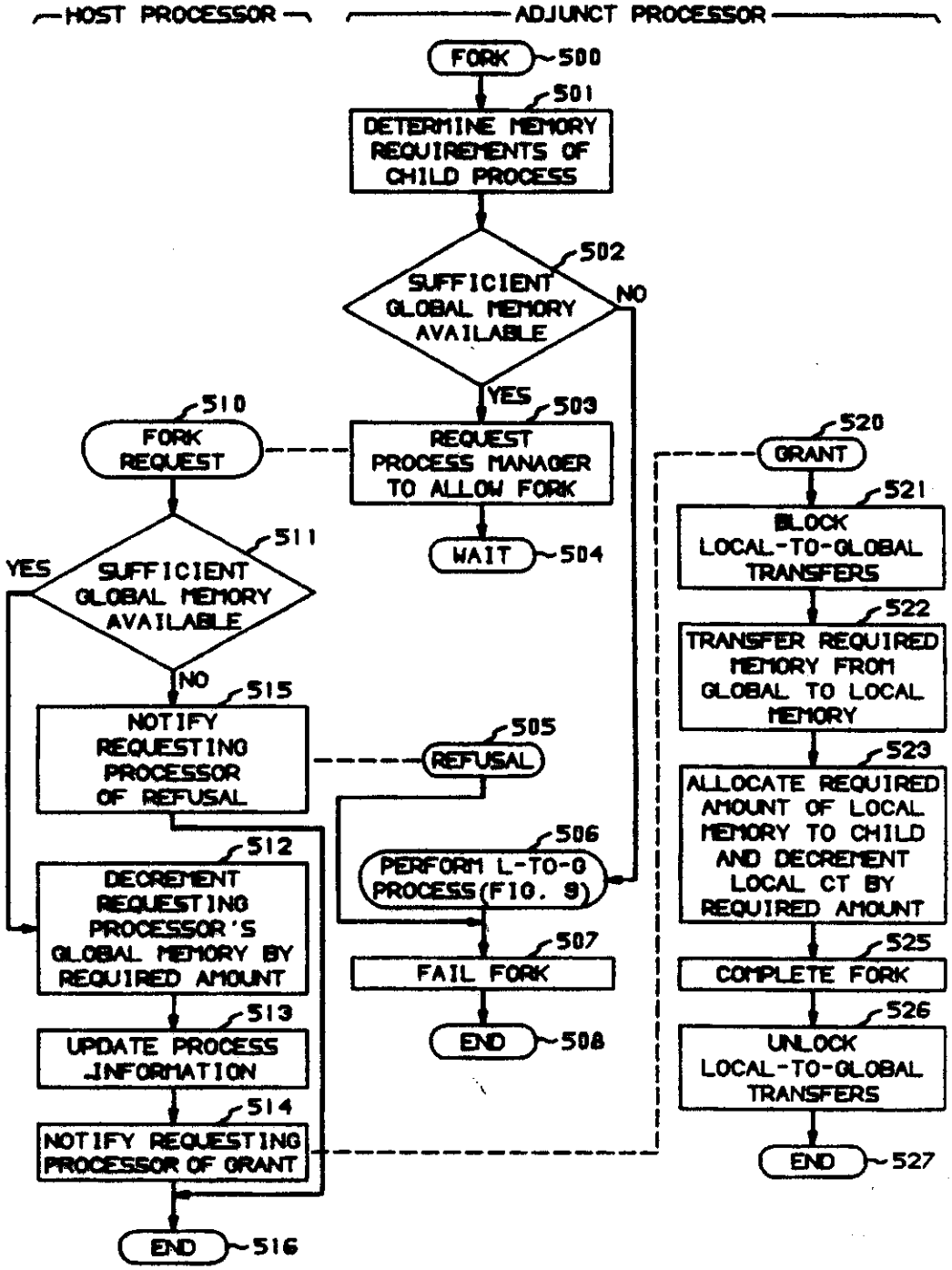

HOST PROCESSOR
ADJUNCT PROCESSOR
FORK
500
501
DETERMINE MEMORY REQUIREMENTS OF CHILD PROCESS
502
SUFFICIENT GLOBAL MEMORY AVAILABLE
NO
YES
503
REQUEST PROCESS MANAGER TO ALLOW FORK
WAIT
504
510
FORK REQUEST
511
SUFFICIENT GLOBAL MEMORY AVAILABLE
YES
NO
515
NOTIFY REQUESTING PROCESSOR OF REFUSAL
505
REFUSAL
506
PERFORM L-TO-G PROCESS (FIG. 9)
507
FAIL FORK
END
508
512
DECREMENT REQUESTING PROCESSOR'S GLOBAL MEMORY BY REQUIRED AMOUNT
513
UPDATE PROCESS INFORMATION
514
NOTIFY REQUESTING PROCESSOR OF GRANT
END
516
520
GRANT
521
BLOCK LOCAL-TO-GLOBAL TRANSFERS
522
TRANSFER REQUIRED MEMORY FROM GLOBAL TO LOCAL MEMORY
523
ALLOCATE REQUIRED AMOUNT OF LOCAL MEMORY TO CHILD AND DECREMENT LOCAL CT BY REQUIRED AMOUNT
525
COMPLETE FORK
526
UNLOCK LOCAL-TO-GLOBAL TRANSFERS
END
527

FIG. 6

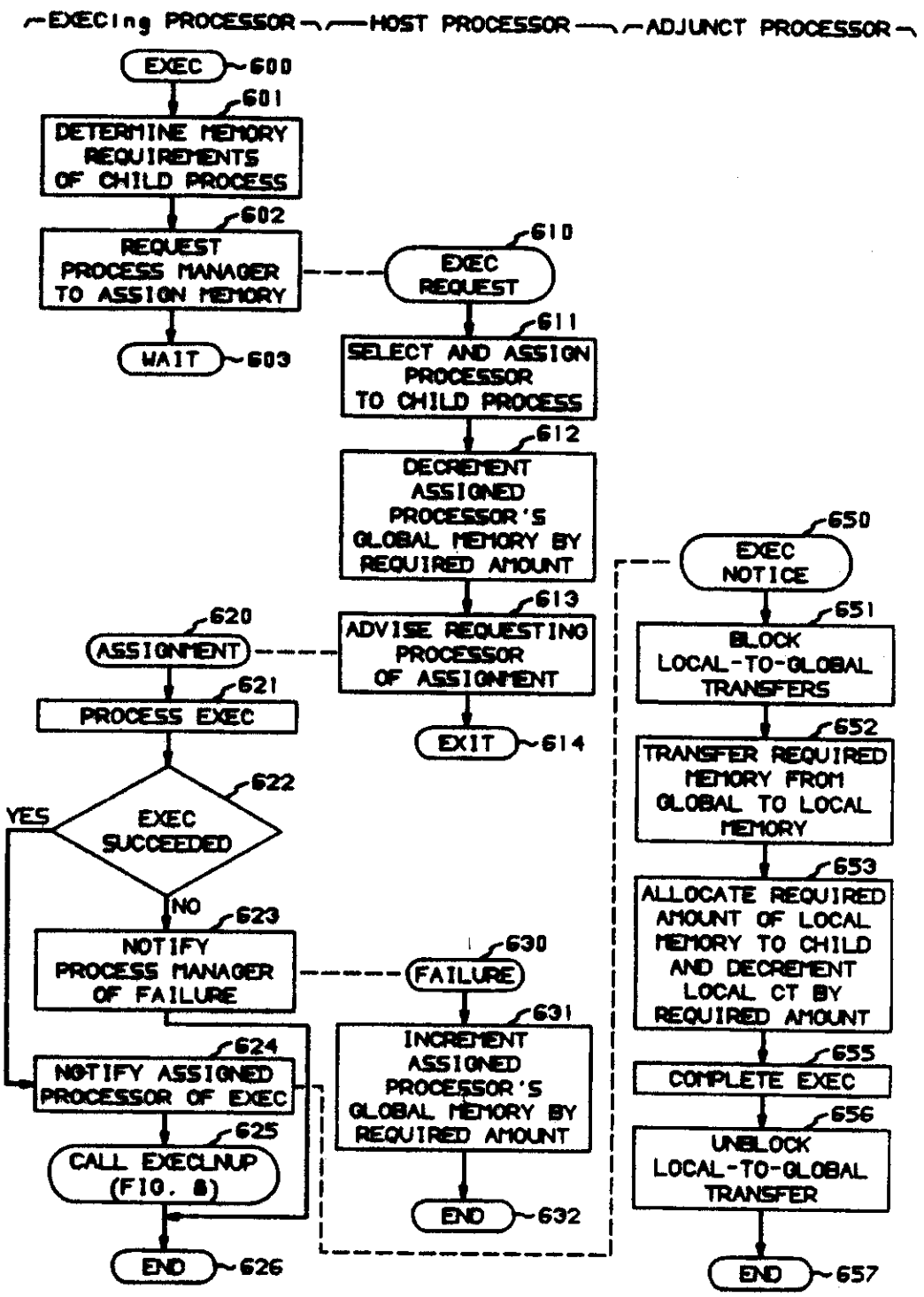

EXECing PROCESSOR
HOST PROCESSOR
ADJUNCT PROCESSOR
EXEC
600
601
DETERMINE MEMORY REQUIREMENTS OF CHILD PROCESS
602
REQUEST PROCESS MANAGER TO ASSIGN MEMORY
WAIT
603
610
EXEC REQUEST
611
SELECT AND ASSIGN PROCESSOR TO CHILD PROCESS
612
DECREMENT ASSIGNED PROCESSOR'S GLOBAL MEMORY BY REQUIRED AMOUNT
613
ADVISE REQUESTING PROCESSOR OF ASSIGNMENT
EXIT
614
620
ASSIGNMENT
621
PROCESS EXEC
622
EXEC SUCCEEDED
YES
NO
623
NOTIFY PROCESS MANAGER OF FAILURE
630
FAILURE
631
INCREMENT ASSIGNED PROCESSOR'S GLOBAL MEMORY BY REQUIRED AMOUNT
END
632
624
NOTIFY ASSIGNED PROCESSOR OF EXEC
625
CALL EXECLNUP (FIG. 8)
END
626
650
EXEC NOTICE
651
BLOCK LOCAL-TO-GLOBAL TRANSFERS
652
TRANSFER REQUIRED MEMORY FROM GLOBAL TO LOCAL MEMORY
653
ALLOCATE REQUIRED AMOUNT OF LOCAL MEMORY TO CHILD AND DECREMENT LOCAL CT BY REQUIRED AMOUNT
655
COMPLETE EXEC
656
UNBLOCK LOCAL-TO-GLOBAL TRANSFER
END
657

FIG. 7

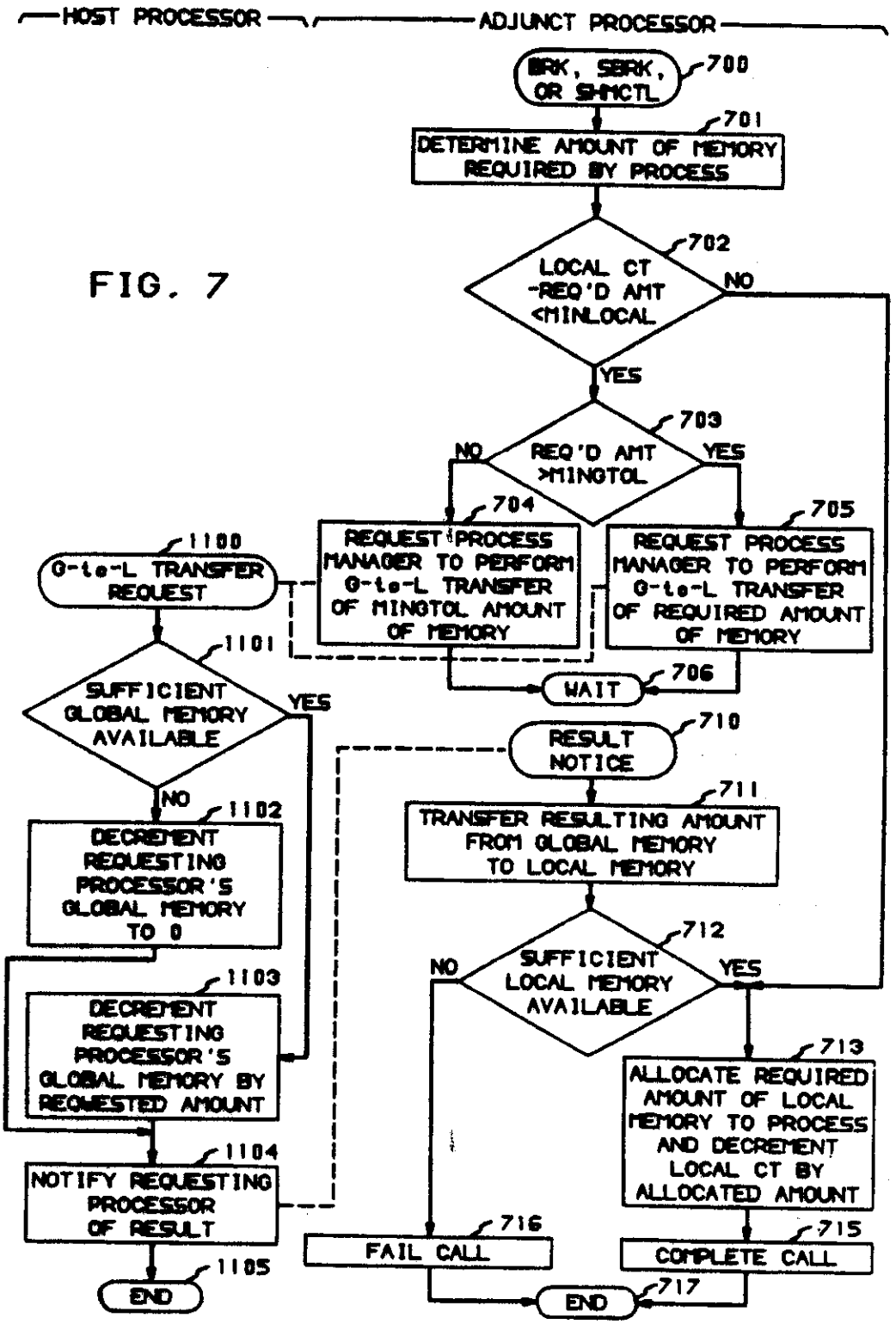

HOST PROCESSOR
ADJUNCT PROCESSOR
SHRK, SSRK, OR SHMCTL
700
DETERMINE AMOUNT OF MEMORY REQUIRED BY PROCESS
701
LOCAL CT -REQ'D AMT <MINLOCAL
702
NO
YES
REQ'D AMT >MINGTOL
703
NO
YES
REQUEST PROCESS MANAGER TO PERFORM G-to-L TRANSFER OF MINGTOL AMOUNT OF MEMORY
704
REQUEST PROCESS MANAGER TO PERFORM G-to-L TRANSFER OF REQUIRED AMOUNT OF MEMORY
705
WAIT
706
1100
G-to-L TRANSFER REQUEST
1101
SUFFICIENT GLOBAL MEMORY AVAILABLE
YES
NO
1102
DECREMENT REQUESTING PROCESSOR'S GLOBAL MEMORY TO 0
1103
DECREMENT REQUESTING PROCESSOR'S GLOBAL MEMORY BY REQUESTED AMOUNT
1104
NOTIFY REQUESTING PROCESSOR OF RESULT
1105
END
RESULT NOTICE
710
TRANSFER RESULTING AMOUNT FROM GLOBAL MEMORY TO LOCAL MEMORY
711
SUFFICIENT LOCAL MEMORY AVAILABLE
712
NO
YES
ALLOCATE REQUIRED AMOUNT OF LOCAL MEMORY TO PROCESS AND DECREMENT LOCAL CT BY ALLOCATED AMOUNT
713
FAIL CALL
716
COMPLETE CALL
715
END
717

FIG. 8

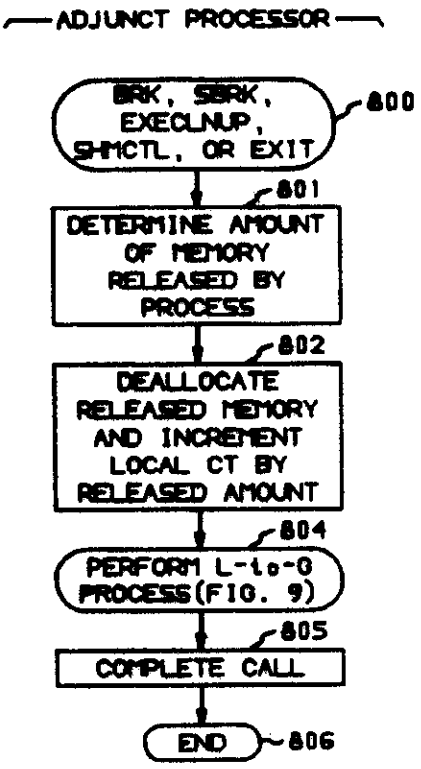
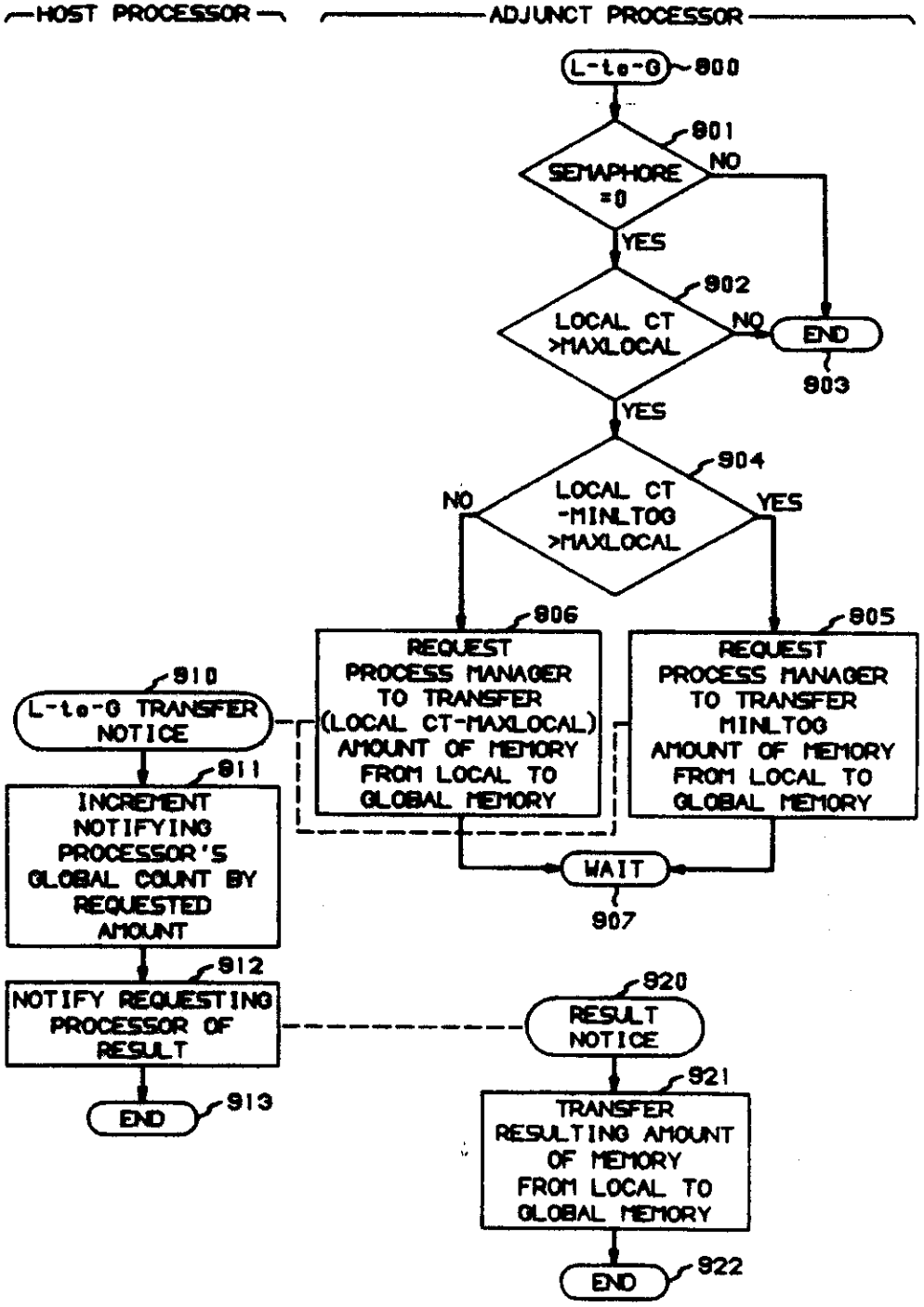

ADJUNCT PROCESSOR
BRK, SBRK, EXECLNUP, SHMCTL, OR EXIT
800
801
DETERMINE AMOUNT OF MEMORY RELEASED BY PROCESS
802
DEALLOCATE RELEASED MEMORY AND INCREMENT LOCAL CT BY RELEASED AMOUNT
804
PERFORM L-to-G PROCESS(FIG. 9)
805
COMPLETE CALL
END
806

FIG. 9


HOST PROCESSOR
ADJUNCT PROCESSOR
L-to-G
800
801
SEMAPHORE =0
NO
YES
902
LOCAL CT >MAXLOCAL
NO
END
903
YES
904
NO
LOCAL CT -MINLTOG >MAXLOCAL
YES
906
REQUEST PROCESS MANAGER TO TRANSFER (LOCAL CT-MAXLOCAL) AMOUNT OF MEMORY FROM LOCAL TO GLOBAL MEMORY
905
REQUEST PROCESS MANAGER TO TRANSFER MINLTOG AMOUNT OF MEMORY FROM LOCAL TO GLOBAL MEMORY
910
L-to-G TRANSFER NOTICE
911
INCREMENT NOTIFYING PROCESSOR'S GLOBAL COUNT BY REQUESTED AMOUNT
WAIT
907
912
NOTIFY REQUESTING PROCESSOR OF RESULT
920
RESULT NOTICE
913
END
921
TRANSFER RESULTING AMOUNT OF MEMORY FROM LOCAL TO GLOBAL MEMORY
END
922

REGISTER ENTRY FOR EP0273612

European Application No EP87310793.2 filing date 09.12.1987

Priority claimed:
22.12.1986 in United States of America - doc: 941703

Designated States BE DE FR GB IT NL SE

Title MULTIPROCESSOR MEMORY MANAGEMENT METHOD AND APPARATUS

Applicant/Proprietor
AMERICAN TELEPHONE AND TELEGRAPH COMPANY, 550 Madison Avenue, New York, NY 10022, United States of America [ADP No. 50209329002]

Inventors
THOMAS PATRICK BISHOP, 2505 Lincolnwood Court, Aurora Illinois 60505, United States of America [ADP No. 55792766001]

MARK HENRY DAVIS, 29W320 Iroquois Court North, Warrenville Illinois 60555, United States of America [ADP No. 55792774001]

ROBERT WAYNE FISH, 29W271 Canterbury Drive, West Chicago Illinois 60185, United States of America [ADP No. 55792782001]

JAMES STUART PETERSON, 255 Peppertree Lane, Aurora Illinois 60505, United States of America [ADP No. 55792790001]

GROVER TIMOTHY SURRATT, 28 West 716 Hickory Lane, West Chicago Illinois 60135, United States of America [ADP No. 55792808001]

Classified to
G4A
G06F

Address for Service
AT&T CORP., 32 Avenue of the Americas, New York, NY 10013-2412, United States of America [ADP No. 50209329002]

EPO Representative
DR. CHRISTOPHER MALCOLM KELWAY WATTS, Western Electric Company Limited 5, Mornington Road, Woodford Green Essex, IG8 0TU, United Kingdom [ADP No. 50071455001]

Publication No EP0273612 dated 06.07.1988
Publication in English

Examination requested 23.11.1991

Patent Granted with effect from 09.08.1995 (Section 25(1)) with title
MULTIPROCESSOR MEMORY MANAGEMENT METHOD AND APPARATUS.

29.04.1991 EPO: Search report published on 29.05.1991
Entry Type 25.11 Staff ID. RD06 Auth ID. EPT

20.05.1994 Notification from EPO of change of Applicant/Proprietor details
from
AMERICAN TELEPHONE AND TELEGRAPH COMPANY, 550 Madison Avenue, New York, NY 10022, United States of America [ADP No. 50209329002]
to
AT&T CORP., 32 Avenue of the Americas, New York, NY 10013-2412, United States of America [ADP No. 50209329002]
Entry Type 25.14 Staff ID. RD06 Auth ID. EPT

07.07.1995 Notification from EPO of change of EPO Representative details from
DR. CHRISTOPHER MALCOLM KELWAY WATTS, Western Electric Company Limited 5, Mornington Road, Woodford Green Essex, IG8 0TU, United Kingdom [ADP No. 50071455001]
to
DR. CHRISTOPHER MALCOLM KELWAY WATTS, AT&T (UK) Ltd. 5, Mornington Road, Woodford Green Essex, IG8 0TU, United Kingdom [ADP No. 50071455001]
Entry Type 25.14 Staff ID. RD06 Auth ID. EPT

**** END OF REGISTER ENTRY ****

RENEWAL DETAILS

PUBLICATION NUMBER EP0273612

PROPRIETOR(S)

AT&T Corp., 32 Avenue of the Americas, New York, NY 10013-2412, United States of America

DATE FILED 09.12.1987

DATE GRANTED 09.08.1995

DATE NEXT RENEWAL DUE 09.12.1995

DATE NOT IN FORCE

DATE OF LAST RENEWAL

YEAR OF LAST RENEWAL 00

STATUS PATENT IN FORCE

**** END OF REPORT ****