8.2 PROCESS MIGRATION

Process migration is the relocation of a process from its current location (the *source node*) to another node (the *destination node*). The flow of execution of a migrating process is illustrated in Figure 8.1.

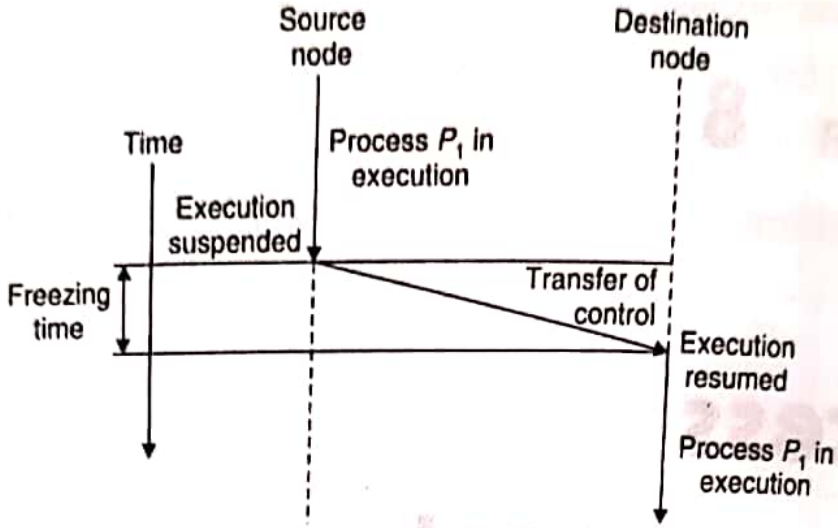

Fig. 8.1 Flow of execution of a migrating process.

A process may be migrated either before it starts executing on its source node or during the course of its execution. The former is known as *non-preemptive* process migration and the latter is known as *preemptive* process migration. Preemptive process migration is costlier than non-preemptive process migration since the process environment must also accompany the process to its new node for an already executing process.

Process migration involves the following major steps:

1. Selection of a process that should be migrated
2. Selection of the destination node to which the selected process should be migrated
3. Actual transfer of the selected process to the destination node

The first two steps are taken care of by the process migration policy and the third step is taken care of by the process migration mechanism. The policies for the selection of a source node, a destination node, and the process to be migrated have already been described in the previous chapter on resource management. This chapter presents a description of the process migration mechanisms used by the existing distributed operating systems.

8.2.1 Desirable Features of a Good Process Migration Mechanism

A good process migration mechanism must possess transparency, minimal interferences, minimal residue dependencies, efficiency, robustness, and communication between coprocesses.

Transparency

Transparency is an important requirement for a system that supports process migration. The following levels of transparency can be identified:

1. *Object access level.* Transparency at the object access level is the minimum requirement for a system to support non-preemptive process migration facility. If a system supports transparency at the object access level, access to objects such as files and devices can be done in a location-independent manner. Thus, the object access level transparency allows free initiation of programs at an arbitrary node. Of course, to support transparency at object access level, the system must provide a mechanism for transparent object naming and locating.

2. *System call and interprocess communication level.* So that a migrated process does not continue to depend upon its originating node after being migrated, it is necessary that all system calls, including interprocess communication, are location independent. Thus, transparency at this level must be provided in a system that is to support preemptive process migration facility. However, system calls to request the physical properties of a node need not be location independent.

Transparency of interprocess communication is also desired for the transparent redirection of messages during the transient state of a process that recently migrated. That is, once a message has been sent, it should reach its receiver process without the need for resending it from the sender node in case the receiver process moves to another node before the message is received.

Minimal Interference

Migration of a process should cause minimal interference to the progress of the process involved and to the system as a whole. One method to achieve this is by minimizing the freezing time of the process being migrated. *Freezing time* is defined as the time period for which the execution of the process is stopped for transferring its information to the destination node.

Minimal Residual Dependencies

No residual dependency should be left on the previous node. That is, a migrated process should not in any way continue to depend on its previous node once it has started executing on its new node since, otherwise, the following will occur:

- The migrated process continues to impose a load on its previous node, thus diminishing some of the benefits of migrating the process.
- A failure or reboot of the previous node will cause the process to fail.

Efficiency

Efficiency is another major issue in implementing process migration. The main sources of inefficiency involved with process migration are as follows:

- The time required for migrating a process
- The cost of locating an object (includes the migrated process)
- The cost of supporting remote execution once the process is migrated

All these costs should be kept to minimum as far as practicable.

Robustness

The process migration mechanism must also be robust in the sense that the failure of a node other than the one on which a process is currently running should not in any way affect the accessibility or execution of that process.

Communication between Coprocesses of a Job

One further exploitation of process migration is the parallel processing among the processes of a single job distributed over several nodes. Moreover, if this facility is supported, to reduce communication cost, it is also necessary that these coprocesses be able to directly communicate with each other irrespective of their locations.

8.2.2 Process Migration Mechanisms

Migration of a process is a complex activity that involves proper handling of several sub-activities in order to meet the requirements of a good process migration mechanism listed above. The four major subactivities involved in process migration are as follows:

1. Freezing the process on its source node and restarting it on its destination node
2. Transferring the process's address space from its source node to its destination node
3. Forwarding messages meant for the migrant process
4. Handling communication between cooperating processes that have been separated (placed on different nodes) as a result of process migration

The commonly used mechanisms for handling each of these subactivities are described below.

Mechanisms for Freezing and Restarting a Process

In preemptive process migration, the usual process is to take a "snapshot" of the process's state on its source node and reinstate the snapshot on the destination node. For this, at some point during migration, the process is frozen on its source node, its state information is transferred to its destination node, and the process is restarted on its destination node using this state information. By freezing the process, we mean that the execution of the process is suspended and all external interactions with the process are deferred. Although the freezing and restart operations differ from system to system, some general issues involved in these operations are described below.

Immediate and Delayed Blocking of the Process. Before a process can be frozen, its execution must be blocked. Depending upon the process's current state, it may be blocked immediately or the blocking may have to be delayed until the process reaches a state when it can be blocked. Some typical cases are as follows [Kingsbury and Kline 1989]:

1. If the process is not executing a system call, it can be immediately blocked from further execution.
2. If the process is executing a system call but is sleeping at an interruptible priority (a priority at which any received signal would awaken the process) waiting for a kernel event to occur, it can be immediately blocked from further execution.
3. If the process is executing a system call and is sleeping at a noninterruptible priority waiting for a kernel event to occur, it cannot be blocked immediately. The process's blocking has to be delayed until the system call is complete. Therefore, in this situation, a flag is set, telling the process that when the system call is complete, it should block itself from further execution.

Note that there may be some exceptions to this general procedure of blocking a process. For example, sometimes processes executing in certain kernel threads are immediately blocked, even when sleeping at noninterruptible priorities. The actual mechanism varies from one implementation to another.

Fast and Slow I/O Operations. In general, after the process has been blocked, the next step in freezing the process is to wait for the completion of all fast I/O operations (e.g., disk I/O) associated with the process. The process is frozen after the completion of all fast I/O operations. Note that it is feasible to wait for fast I/O operations to complete before freezing the process. However, it is not feasible to wait for slow I/O operations, such as those on a pipe or terminal, because the process must be frozen in a timely manner for the effectiveness of process migration. Thus proper mechanisms are necessary for continuing these slow I/O operations correctly after the process starts executing on its destination node.

Information about Open Files. A process's state information also consists of the information pertaining to files currently open by the process. This includes such information as the names or identifiers of the files, their access modes, and the current

positions of their file pointers. In a distributed system that provides a network transparent execution environment, there is no problem in collecting this state information because the same protocol is used to access local as well as remote files using the systemwide unique file identifiers. However, several UNIX-based network systems uniquely identify files by their full pathnames [Mandelberg and Sunderam 1988, Alonso and Kyrimis 1988]. But in these systems it is difficult for a process in execution to obtain a file's complete pathname owing to UNIX file system semantics. A pathname loses significance once a file has been opened by a process because the operating system returns to the process a file descriptor that the process uses to perform all I/O operations on the file. Therefore, in such systems, it is necessary to somehow preserve a pointer to the file so that the migrated process could continue to access it. The following two approaches are used for this:

1. In the first approach [Mandelberg and Sunderam 1988], a link is created to the file and the pathname of the link is used as an access point to the file after the process migrates. Thus when the snapshot of the process's state is being created, a link (with a special name) is created to each file that is in use by the process.

2. In the second approach [Alonso and Kyrimis 1988], an open file's complete pathname is reconstructed when required. For this, necessary modifications have to be incorporated in the UNIX kernel. For example, in the approach described in [Alonso and Kyrimis 1988], each file structure, where information about open files is contained, is augmented with a pointer to a dynamically allocated character string containing the absolute pathname of the file to which it refers.

Another file system issue is that one or more files being used by the process on its source node may also be present on its destination node. For example, it is likely that the code for system commands such as *nroff*, *cc* is replicated at every node. It would be more efficient to access these files from the local node at which the process is executing, rather than accessing them across the network from the process's previous node [Agrawal and Ezzat 1987]. Another example is temporary files that would be more efficiently created at the node on which the process is executing, as by default these files are automatically deleted at the end of the operation. Therefore, for performance reasons, the file state information collected at the source node should be modified properly on the process's destination node in order to ensure that, whenever possible, local file operations are performed instead of remote file operations. This also helps in reducing the amount of data to be transferred at the time of address space transfer because the files already present on the destination node need not be transferred.

Reinstating the Process on its Destination Node. On the destination node, an empty process state is created that is similar to that allocated during process creation. Depending upon the implementation, the newly allocated process may or may not have the same process identifier as the migrating process. In some implementations, this newly created copy of the process initially has a process identifier different from the migrating process in order to allow both the old copy and the new copy to exist and be accessible at the same time. However, if the process identifier of the new copy of the process is different from its old copy, the new copy's identifier is changed to the original identifier

in a subsequent step before the process starts executing on the destination node. The rest of the system cannot detect the existence of two copies of the process because operations on both of them are suspended. Once all the state of the migrating process has been transferred from the source node to the destination node and copied into the empty process state, the new copy of the process is unfrozen and the old copy is deleted. Thus the process is restarted on its destination node in whatever state it was in before being migrated.

It may be noted here that the method described above to reinstate the migrant process is followed in the most simple and straightforward case. Several special cases may require special handling and hence more work. For example, when obtaining the snapshot, the process may have been executing a system call since some calls are not atomic. In particular, as described before, this can happen when the process is frozen while performing an I/O operation on a slow device. If the snapshot had been taken under such conditions, correct process continuation would be possible only if the system call is performed again. Therefore normally a check is made for these conditions and, if required, the program counter is adjusted as needed to reissue the system call.

Address Space Transfer Mechanisms

A process consists of the program being executed, along with the program's data, stack, and state. Thus, the migration of a process involves the transfer of the following types of information from the source node to the destination node:

- Process's state, which consists of the execution status (contents of registers), scheduling information, information about main memory being used by the process (memory tables), I/O states (I/O queue, contents of I/O buffers, interrupt signals, etc.), a list of objects to which the process has a right to access (capability list), process's identifier, process's user and group identifiers, information about the files opened by the process (such as the mode and current position of the file pointer), and so on
- Process's address space (code, data, and stack of the program)

For nontrivial processes, the size of the process's address space (several megabytes) overshadows the size of the process's state information (few kilobytes). Therefore the cost of migrating a process is dominated by the time taken to transfer its address space. Although it is necessary to completely stop the execution of the migrant process while transferring its state information, it is possible to transfer the process's address space without stopping its execution. In addition, the migrant process's state information must be transferred to the destination node before it can start its execution on that node. Contrary to this, the process's address space can be transferred to the destination node either before or after the process starts executing on the destination node.

Thus in all the systems, the migrant process's execution is stopped while transferring its state information. However, due to the flexibility in transferring the process's address space at any time after the migration decision is made, the existing distributed systems use one of the following address space transfer mechanisms: total freezing, pretransferring, or transfer on reference.

Total Freezing. In this method, a process's execution is stopped while its address space is being transferred (Fig. 8.2). This method is used in DEMOS/MP [Powell and Miller 1983], Sprite [Douglis and Ousterhout 1987], and LOCUS [Popek and Walker 1985] and is simple and easy to implement. Its main disadvantage is that if a process is suspended for a long time during migration, timeouts may occur, and if the process is interactive, the delay will be noticed by the user.

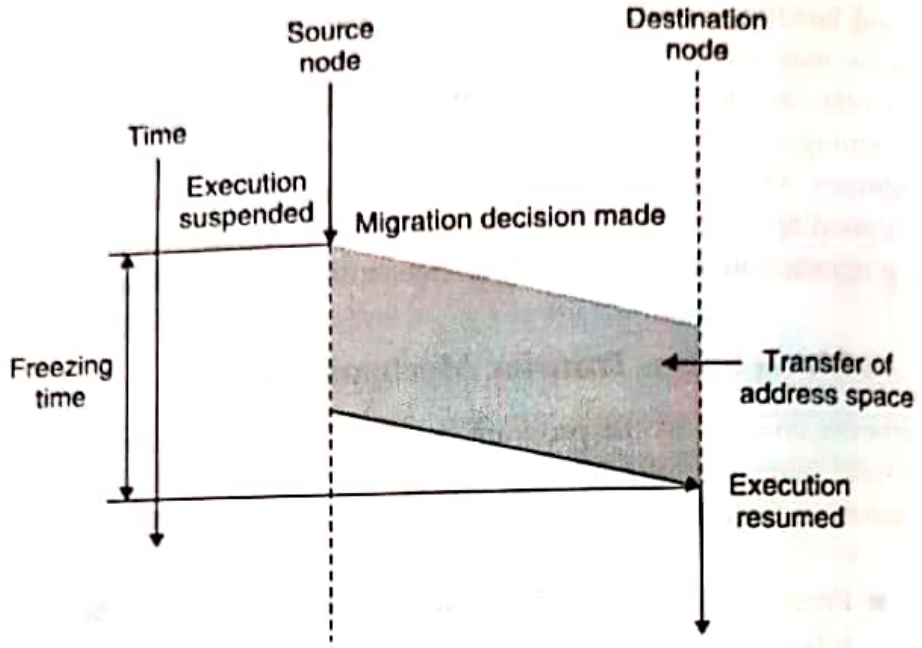

Fig. 8.2 Total freezing mechanism.

Pretransferring. In this method, the address space is transferred while the process is still running on the source node (Fig. 8.3). Therefore, once the decision has been made to migrate a process, it continues to run on its source node until its address space has been transferred to the destination node. Pretransferring (also known as *precopying*) is done as an initial transfer of the complete address space followed by repeated transfers of the pages modified during the previous transfer until the number of modified pages is relatively small or until no significant reduction in the number of modified pages (detected using dirty bits) is achieved. The remaining modified pages are retransferred after the process is frozen for transferring its state information [Theimer et al. 1985].

In the pretransfer operation, the first transfer operation moves the entire address space and takes the longest time, thus providing the longest time for modifications to the program's address space to occur. The second transfer moves only those pages of the address space that were modified during the first transfer, thus taking less time and presumably allowing fewer modifications to occur during its execution time. Thus subsequent transfer operations have to move fewer and fewer pages, finally converging to zero or very few pages, which are then transferred after the process is frozen. It may be noted here that the pretransfer operation is executed at a higher priority than all other

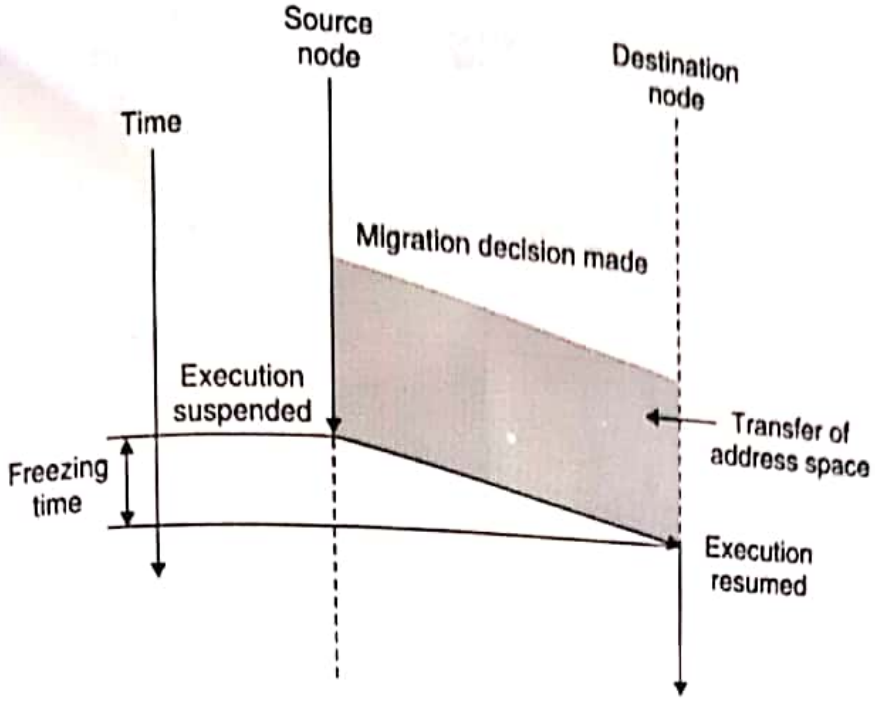

Fig. 8.3 Pretransfer mechanism.

programs on the source node to prevent these other programs from interfering with the progress of the pretransfer operation.

This method is used in the V-System [Theimer et al. 1985]. In this method, the freezing time is reduced so migration interferes minimally with the process's interaction with other processes and the user. Although pretransferring reduces the freezing time of the process, it may increase the total time for migration due to the possibility of redundant page transfers. *Redundant pages* are pages that are transferred more than once during pretransferring because they become dirty while the pretransfer operation is being performed.

Transfer on Reference. This method is based on the assumption that processes tend to use only a relatively small part of their address spaces while executing. In this method, the process's address space is left behind on its source node, and as the relocated process executes on its destination node, attempts to reference memory pages results in the generation of requests to copy in the desired blocks from their remote locations. Therefore in this demand-driven copy-on-reference approach, a page of the migrant process's address space is transferred from its source node to its destination node only when referenced (Fig. 8.4). However, Zayas [1987] also concluded through his simulation results that prefetching of one additional contiguous page per remote fault improves performance.

This method is used in Accent [Zayas 1987]. In this method, the switching time of the process from its source node to its destination node is very short once the decision about migrating the process has been made and is virtually independent of the size of the address space. However, this method is not efficient in terms of the cost of supporting remote execution once the process is migrated, and part of the effort saved in the lazy transfer of an address space must be expended as the process accesses its memory

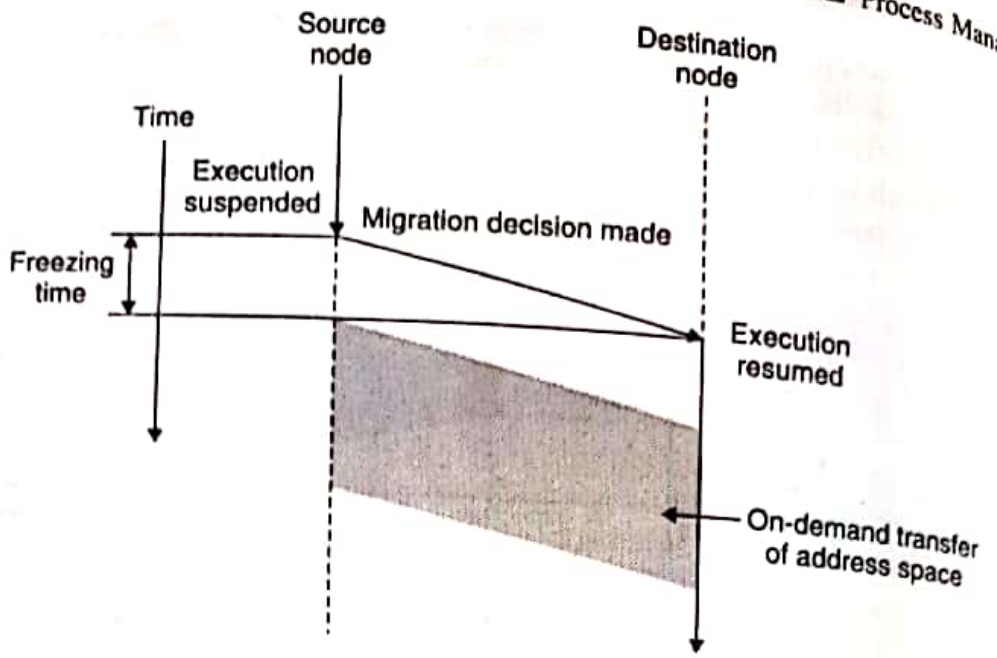

Fig. 8.4 Transfer-on-reference mechanism.

remotely. Furthermore, this method imposes a continued load on the process's source node and results in failure of the process if the source node fails or is rebooted.

Message-Forwarding Mechanisms

In moving a process, it must be ensured that all pending, en-route, and future messages arrive at the process's new location. The messages to be forwarded to the migrant process's new location can be classified into the following:

Type 1: Messages received at the source node after the process's execution has been stopped on its source node and the process's execution has not yet been started on its destination node

Type 2: Messages received at the source node after the process's execution has started on its destination node

Type 3: Messages that are to be sent to the migrant process from any other node after it has started executing on the destination node

The different mechanisms used for message forwarding in existing distributed systems are described below.

Mechanism of Resending the Message. This mechanism is used in the V-System [Cheriton 1988, Theimer et al. 1985] and Amoeba [Mullender et al. 1990] to handle messages of all three types. In this method, messages of types 1 and 2 are returned to the sender as not deliverable or are simply dropped, with the assurance that the sender of the message is storing a copy of the data and is prepared to retransmit it.

For example, in V-System, a message of type 1 or 2 is simply dropped and the sender is prompted to resend it to the process's new node. The interprocess communication mechanism of V-System ensures that senders will retry until successful receipt of a reply. Similarly in Amoeba, for all messages of type 1, the source node's kernel sends a "try again later, this process is frozen" message to the sender. After the process has been deleted from the source node, those messages will come again at some point of time, but this time as type 2 messages. For all type 2 messages, the source node's kernel sends a "this process is unknown at this node" message to the sender.

In this method, upon receipt of a negative reply, the sender does a *locate* operation to find the new whereabouts of the process, and communication is reestablished. Both V-System and Amoeba use the broadcasting mechanism to locate a process (object locating mechanisms are described in Chapter 10). Obviously, in this mechanism, messages of type 3 are sent directly to the process's destination node.

This method does not require any process state to be left behind on the process's source node. However, the main drawback of this mechanism is that the message-forwarding mechanism of process migration operation is nontransparent to the processes interacting with the migrant process.

Origin Site Mechanism. This method is used in AIX's TCF (Transparent Computing Facility) [Walker and Mathews 1989] and Sprite [Douglis and Ousterhout 1987]. The process identifier of these systems has the process's *origin site* (or *home node*) embedded in it, and each site is responsible for keeping information about the current locations of all the processes created on it. Therefore, a process's current location can be simply obtained by consulting its origin site. Thus, in these systems, messages for a particular process are always first sent to its origin site. The origin site then forwards the message to the process's current location. This method is not good from a reliability point of view because the failure of the origin site will disrupt the message-forwarding mechanism. Another drawback of this mechanism is that there is a continuous load on the migrant process's origin site even after the process has migrated from that node.

Link Traversal Mechanism. In DEMOS/MP [Powell and Miller 1983], to redirect the messages of type 1, a message queue for the migrant process is created on its source node. All the messages of this type are placed in this message queue. Upon being notified that the process is established on the destination node, all messages in the queue are sent to the destination node as a part of the migration procedure.

To redirect the messages of types 2 and 3, a forwarding address known as *link* is left at the source node pointing to the destination node of the migrant process. The most important part of a link is the message process address that has two components. The first component is a systemwide, unique, process identifier. It consists of the identifier of the node on which the process was created and a unique local identifier generated by that node. The second component is the last known location of the process. During the lifetime of a link, the first component of its address never changes; the second, however, may. Thus to forward messages of types 2 and 3, a migrated process is located by traversing a series of links (starting from the node where the process was created) that form a chain ultimately leading to the process's current location. The second component of a link is updated when the corresponding

process is accessed from a node. This is done to improve the efficiency of subsequent locating operations for the same process from that node.

The link traversal mechanism used by DEMOS/MP for message forwarding suffers from the drawbacks of poor efficiency and reliability. Several links may have to be traversed to locate a process from a node, and if any node in the chain of links fails, the process cannot be located.

Link Update Mechanism. In Charlotte [Artsy and Finkel 1989], processes communicate via location-independent links, which are capabilities for duplex communication channels. During the transfer phase of the migrant process, the source node sends link-update messages to the kernels controlling all of the migrant process's communication partners. These link update messages tell the new address of each link held by the migrant process and are acknowledged (by the notified kernels) for synchronization purposes. This task is not expensive since it is performed in parallel. After this point, messages sent to the migrant process on any of its links will be sent directly to the migrant process's new node. Communication requests postponed while the migrant process was being transferred are buffered at the source node and directed to the destination node as a part of the transfer process. Therefore, messages of types 1 and 2 are forwarded to the destination node by the source node and messages of type 3 are sent directly to the process's destination node.

Mechanisms for Handling Coprocesses

In systems that allow process migration, another important issue is the necessity to provide efficient communication between a process (parent) and its subprocesses (children), which might have been migrated and placed on different nodes. The two different mechanisms used by existing distributed operating systems to take care of this problem are described below.

Disallowing Separation of Coprocesses. The easiest method of handling communication between coprocesses is to disallow their separation. This can be achieved in the following ways:

1. By disallowing the migration of processes that wait for one or more of their children to complete
2. By ensuring that when a parent process migrates, its children processes will be migrated along with it

The first method is used by some UNIX-based network systems [Alonso and Kyrimis 1988, Mandelberg and Sunderam 1988] and the second method is used by V-System [Theimer et al. 1985]. To ensure that a parent process will always be migrated along with its children processes, V-System introduced the concept of *logical host*. V-System address spaces and their associated processes are grouped into logical hosts. A V-System process identifier is structured as a (*logical-host-id*, *local-index*) pair. In the extreme, each program can be run in its own logical host. There may be multiple logical hosts associated with a single node; however, a logical host is local to a single node. In V-System, all subprocesses

of a process typically execute within a single logical host. Migration of a process is actually migration of the logical host containing that process. Thus, typically, all subprocesses of a process are migrated together when the process is migrated [Theimer et al. 1985].

The main disadvantage of this method is that it does not allow the use of parallelism within jobs, which is achieved by assigning the various tasks of a job to the different nodes of the system and executing them simultaneously on these nodes. Furthermore, in the method employed by V-System, the overhead involved in migrating a process is large when its logical host consists of several associated processes.

Home Node or Origin Site Concept. Sprite [Douglis and Ousterhout 1987] uses its home node concept (previously described) for communication between a process and its subprocess when the two are running on different nodes. Unlike V-System, this allows the complete freedom of migrating a process or its subprocesses independently and executing them on different nodes of the system. However, since all communications between a parent process and its children processes take place via the home node, the message traffic and the communication cost increase considerably. Similar drawbacks are associated with the concept of origin site of LOCUS [Popek and Walker 1985].