

Table 7.1 Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual-Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual-Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Table 7.2 Address Binding

(a) Loader

Binding Time	Function
Programming time	All actual physical addresses are directly specified by the programmer in the program itself.
Compile or assembly time	The program contains symbolic address references, and these are converted to actual physical addresses by the compiler or assembler.
Load time	The compiler or assembler produces relative addresses. The loader translates these to absolute addresses at the time of program loading.
Run time	The loaded program retains relative addresses. These are converted dynamically to absolute addresses by processor hardware.

(b) Linker

Linkage Time	Function
Programming time	No external program or data references are allowed. The programmer must place into the program the source code for all subprograms that are referenced.
Compile or assembly time	The assembler must fetch the source code of every subroutine that is referenced and assemble them as a unit.
Load module creation	All object modules have been assembled using relative addresses. These modules are linked together and all references are restated relative to the origin of the final load module.
Load time	External references are not resolved until the load module is to be loaded into main memory. At that time, referenced dynamic link modules are appended to the load module, and the entire package is loaded into main or virtual memory.
Run time	External references are not resolved until the external call is executed by the processor. At that time, the process is interrupted and the desired module is linked to the calling program.