MODULE 3 :MEMORY MANAGEMENT

Main Memory Management Strategies

Every program to be executed has to be executed must be in memory. The instruction must be fetched from memory before it is executed.

In multi-tasking OS memory management is complex, because as processes are swapped in and out of the CPU, their code and data must be swapped in and out of memory.

Basic Hardware

Main memory, cache and CPU registers in the processors are the only storage spaces that CPU can access directly.

The program and data must be bought into the memory from the disk, for the process to run. Each process has a separate memory space and must access only this range of legal addresses. Protection of memory is required to ensure correct operation. This prevention is provided by hardware implementation.

Two registers are used - a base register and a limit register. The base register holds the smallest legal physical memory address; the limit register specifies the size of the range.

For example, The base register holds the smallest legal physical memory address; the limit register specifies the size of the range. For example, if the base register holds 300040 and limit register is 120900, then the program can legally access all addresses from 300040 through 420940 (inclusive).

Figure: A base and a limit-register define a logical-address space

The base and limit registers can be loaded only by the operating system, which uses a special privileged instruction. Since privileged instructions can be executed only in kernel mode only the operating system can load the base and limit registers.

Figure: Hardware address protection with base and limit-registers

Address Binding

User programs typically refer to memory addresses with symbolic names. These symbolic names
must be mapped or bound to physical memory addresses.

Address binding of instructions to memory-addresses can happen at 3 different stages.

1. Compile Time - If it is known at compile time where a program will reside in physical memory, then absolute code can be generated by the compiler, containing actual physical addresses. However, if
the load address changes at some later time, then the program will have to be recompiled.
2. Load Time - If the location at which a program will be loaded is not known at compile time, then
the compiler must generate relocatable code, which references addresses relative to the start of the
program. If that starting address changes, then the program must be reloaded but not recompiled.
3. Execution Time - If a program can be moved around in memory during the course of its execution,
then binding must be delayed until execution time.

Figure: Multistep processing of a user program

Logical Versus Physical Address Space

The address generated by the CPU is a logical address, whereas the memory address where
programs are actually stored is a physical address.

The set of all logical addresses used by a program composes the logical address space, and the set of
all corresponding physical addresses composes the physical address space.

The run time mapping of logical to physical addresses is handled by the memorymanagement unit
(MMU).

One of the simplest is a modification of the base-register scheme.

The base register is termed a relocation register

The value in the relocation-register is added to every address generated by a user-process at the time
it is sent to memory.

The user-program deals with logical-addresses; it never sees the real physicaladdresses.

Figure: Dynamic relocation using a relocation-register

Dynamic Loading

This can be used to obtain better memory-space utilization.

A routine is not loaded until it is called.

This works as follows:

1. Initially, all routines are kept on disk in a relocatable-load format.
2. Firstly, the main-program is loaded into memory and is executed.
3. When a main-program calls the routine, the main-program first checks to see whether the routine has
been loaded.
4. If routine has been not yet loaded, the loader is called to load desired routine into memory.
5. Finally, control is passed to the newly loaded-routine.

Advantages:

1. An unused routine is never loaded.
2. Useful when large amounts of code are needed to handle infrequently occurring cases.
3. Although the total program-size may be large, the portion that is used (and hence loaded) may be
much smaller.
4. Does not require special support from the OS.

Dynamic Linking and Shared Libraries

With static linking library modules get fully included in executable modules, wasting both disk space and main memory usage, because every program that included a certain routine from the library
would have to have their own copy of that routine linked into their executable code.

With dynamic linking, however, only a stub is linked into the executable module, containing references to the actual library module linked in at run time.

The stub is a small piece of code used to locate the appropriate memory-resident library-routine.

This method saves disk space, because the library routines do not need to be fully included in the
executable modules, only the stubs.

An added benefit of dynamically linked libraries (DLLs, also known as shared libraries or shared
objects on UNIX systems) involves easy upgrades and updates.

Shared libraries

A library may be replaced by a new version, and all programs that reference the library will automatically use the new one.

Version info. is included in both program & library so that programs won't accidentally execute incompatible versions.

Figure: Swapping of two processes using a disk as a backing store

Example:

Assume that the user process is 10 MB in size and the backing store is a standard hard disk with a
transfer rate of 40 MB per second.

The actual transfer of the 10-MB process to or from main memory takes 10000 KB/40000 KB per
second = 1/4 second = 250 milliseconds.