4.0 Memory Management

Memory management is responsible for allocating memory to processes and for protecting the memory allocated to each process from undesired access by other processes. It is also responsible for protecting the memory allocated to the OS from unauthorized access. Memory management is both software and hardware work.

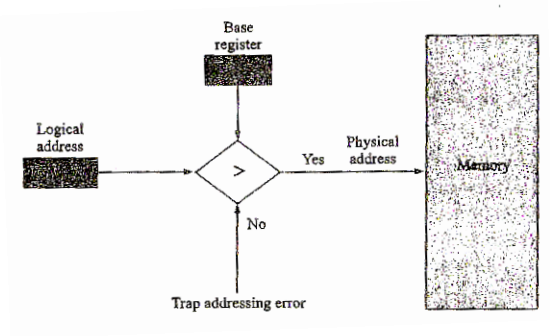
Memory Management Techniques

4.1 Single Absolute Partition

Simplest memory management technique where memory space is divide into **two partitions** by convention i.e. one partition is allocated to the OS and the other to an executing process. It requires no hardware support. Programs tend to corrupt the OS because there is no H/W support. Example: early DOS systems.

Mechanism

- When a program is loaded into a partition, the addresses in the loaded code must correspond to the appropriate addresses in the partition.
- Addresses in the code may be **bound** to particular memory addresses at either compile time or load time.
- Programs that have been bound to memory locations at compile time are said to contain an **absolute code.** If the binding occurs when the program is loaded into memory, the program is said to contain **relocatable code.**
- For a program to contain relocatable code, some mechanism must exist to identify the bytes in the program that refer to memory addresses. Although this adds complexity to the compiling and loading process, it has the obvious advantage of freeing the program to be loaded into any available memory locations.
- Protection can be incorporated into an absolute partition scheme by adding a hardware **base register.**
- The OS loads the base register with the lowest address accessible by the user program. The H/W then compares each address generated by the user program to the contents of the contents of the base register.
- Addresses less than the address stored in the base register cause a memory-fault trap into the OS.



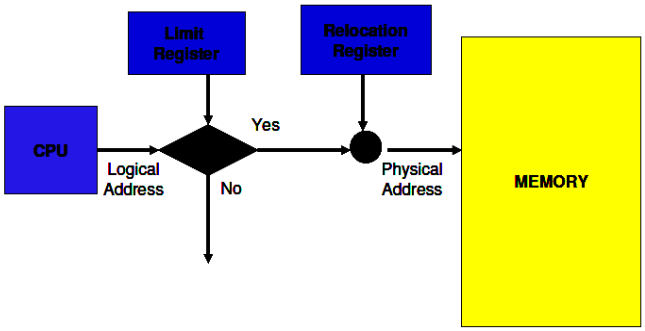
4.2 Single Relocatable Partition

- Far more useful than base register systems because they contain a **relocation register**.
- As with base register, the relocation register is loaded by the OS with the starting address of the process. But instead of comparing the relocation register's content's to the address generated by the process, its contents are added.
- The program now operates in a **logical** address space. It is compiled as if it would be allocated to memory starting at location 0.
- The memory management H/W converts logical addresses into actual **physical** addresses.

Note: Single-partition schemes limit a computer to executing one program at a time. Such mono-programming systems are increasingly rare. To load multiple processes, the OS must divide memory into multiple partitions for those processes.

4.3 Multiple Partitions

Multiple partitions are created to allow multiple user processes to be resident in memory simultaneously. A second H/W register, to be used in conjunction with the relocation register marks the end of a partition. Typically, the register contains the partition's size and is referred to as the **size register** or the **limit register**.



4.3.1 Multiple fixed partitions

- It reduces **internal fragmentation** i.e. wasted space at the end of a partition that is unused by a process.
- The OS must reset a process each time a process is allocated to the CPU. The partition must be either stored in the partition table or deduced from the partition number.
- Different size-partitions complicate process management. Each time a different process is given control of the CPU, the OS must reset the size register in addition to the relocation register. The OS must also decide which partition it should allocate to a process (of the right size).

4.3.2 Multiple variable partitions

- It is better than multiple fixed partitions.
- OS can choose to place processes into whichever memory locations are unused. The amount of space allocated to a process is the exact amount of space it requires, eliminating internal fragmentation. But the OS must keep more data.
- Memory usage evolves into alternating sections of allocated and unallocated space, or **checkerboarding.** As a result, although there may be more memory unallocated than the size of a waiting process. That memory can't be used for the process because it is scattered among a number of memory **holes**. This wasted space not allocated to any partition is called **external fragmentation.**

- A **bit map** can be used to keep track of what memory has been allocated. Memory is divided into allocation units and each bit in the map indicates whether or not the corresponding unit is allocated. Increasing the size of the allocation unit decreases the size of the bit map, but increases the amount of memory wasted when the process size is not a multiple of the size of the allocation unit.
- A **linked list** can also be used to keep track of free memory. Each hole will contain an entry indicating the hole's size and a pointer to the next hole in the list.
- **Compaction** may be used to make more efficient use of memory. By moving processes in memory, the memory holes can be collected into a single section of unallocated space.

Partition selection algorithms

- **First Fit:** the OS locks all the sectors of free memory. The process is allocated to the first hole found that is larger than the size of the process. Unless the size of the process matches the size of the hole, the hole continues to exist, reduced by the size of the process.
- **Next Fit:** attempts to improve performance by distributing its searches more evenly over the entire memory space. It does this by keeping track of which was last allocated. The next search starts at the last hole allocated, not at the beginning of memory.
- **Best Fit:** searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.
- **Worst Fit:** in situations where best fit finds an almost perfect match, the hole that remains is virtually useless because it is so small. To prevent the creation of these useless holes, worst fit works the opposite of first fit; it always picks the largest remaining hole.

4.4 Buddy systems

- The **buddy memory allocation** technique is a memory allocation technique that divides memory into partitions to try to satisfy a memory request as suitably as possible. This system makes use of splitting memory into halves to try to give a best-fit.
- Compared to the memory allocation techniques (such as paging) that modern operating systems such as Microsoft Windows and Linux use, the buddy memory allocation is relatively easy to implement, and does not have the hardware requirement of a memory management unit (MMU). Thus, it can be implemented, for example, on Intel 80286 and below computers.
- Memory is allocated in units that are a power of 2. Initially there is a single allocation unit that comprises all of memory.
- When memory must be allocated to a process, the process is allocated a unit memory whose size is the smallest power of 2 larger than the size of the process. For example, a 50K process would be placed in a 64K allocation unit. If no allocation units exist of that size, the smallest available allocation larger than the process will split into two "buddy" units half the size of the original. Splitting continues with one of the buddy units until an allocation unit of the appropriate size is created
- When a process releasing memory causes two buddy units to both be free, the units are combined to form a unit twice as large.
- Given a memory size of 2^N, a buddy system maintains a maximum of N lists of free blocks, one for each size.(given a memory size of 2^8, allocation units of size 2^8 through 2^1 are possible). with a separate list of available blocks for each size, the allocation or release of memory can be processes efficiently.
- Buddy system does not use memory efficiently because of both internal and external fragmentation.

Action	Memory					
Start	1M					
Request A, 150K	A	256K			512K	
Request B, 100K	A	B	128K		512K	
Request C, 50K	A	B	C	64K	512K	
Release B	A	128K	C	64K	512K	
Request D, 200K	A	128K	C	64K	D	256K
Request E, 60K	A	128K	C	E	D	256K
Release C	A	128K	64K	E	D	256K
Release A	256K	128K	64K	E	D	256K
Release E	512K				D	256K
Release D	1M					

Revision Questions

Q1. Explain the following terms: ***relocatable code, base register*** and ***logical address***.

Q2 Using a simple diagram, describe dynamic memory relocation scheme.

Q3. List **THREE** advantages of segmented paging over pure segmentation.

Q4. Describe the *Buddy* system of memory allocation.