Memory Management Basics

Memory is central to the operation of a modern computer System. Memory consists of a large array of words or bytes each with its own address. The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses. I have discussed a typical instruction-execution cycle in the previous article. In this article I will discuss on basic hardware issues, the binding of symbolic memory addresses to actual physical addresses and distinguishing between logical and physical addresses.

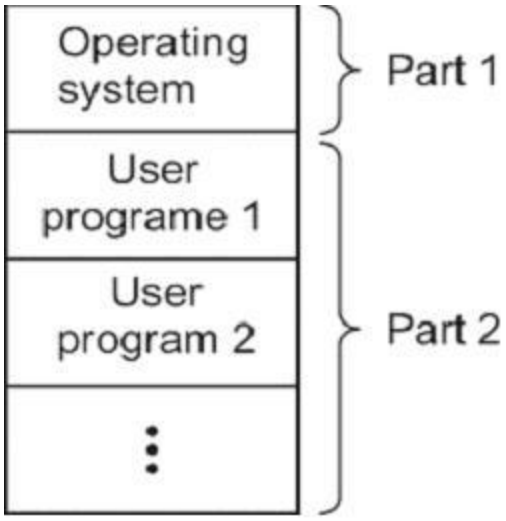
Basic Hardware

Main memory and the registers built into the processor itself are the only storage that the CPU can access directly. There are machine instructions that take memory addresses as arguments, but none that take disk addresses. Therefore, any instructions in execution, and any data being used by the instructions, must be in one of these direct-access storage devices. If the data are not in memory, they must be moved there before the CPU can operate on them.

Registers that are built into the CPU are generally accessible within one cycle of the CPU clock. Most CPU can decode instructions and perform simple operations on register contents at the rate of one or more operations per clock tick. The same cannot be said of main memory, which is accessed via a transaction on the memory bus. Memory access may take many cycles of the CPU clock to complete, in which case the processor normally needs to stall, since it does not have the data required to complete the instruction that it is executing. The situation is intolerable because of the frequency of memory accesses. The remedy is to add fast memory between the CPU and main memory. A memory buffer used to accommodate a speed differential is called a cache.

Main memory is divided into two parts. One part is to hold the operating system instructions, where as the other part will hold the program which is currently being executed by the user (i.e., in a uni-programming). In a multiprogramming system, the user part is further subdivided, for accommodating multiple processes. The task of subdivision is carried out dynamically by the operating system is known as memory management. Memory management module of an Operating System is concerned with organization of main memory. Memory management is concerned with:

- keeping track of the status of each memory location i.e. allocated or free.
- determining allocation policy for memory.
- reclaiming previously allocated memory.



Memory Management Schemes

There are two memory management schemes:

- **Single User Case-Mono Programming**
- **Multiple User Case-Multi Programming**

Single User-Mono Programming

It is a simple memory management scheme and associated with simple microcomputers. In this scheme memory is divided up between the OS and a single user process. The OS is protected from user programs (to prevent user program overwriting OS code).

Advantages

- Simplicity
- Small OS

Disadvantages

- Poor utilization of memory. It is wasteful.
- Poor utilization of processor.
- User address space may contain information that is never used.
- Low flexibility. User's job limited to the size of available memory.

Multiple Users-Multiprogramming

It is a memory management scheme in which a user can operate on more than one job at a

time. Resources are distributed among jobs. In this scheme there are two problems- relocation and protection. The solution to this problem is to use register-base and limit register. We need to maximize the degree of multiprogramming i.e. the number of processes in memory.

Memory-Management Requirements

Memory management should satisfy the following requirements:

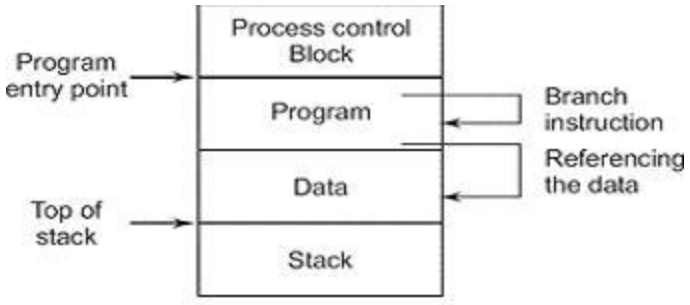
- Relocation
- Sharing
- Protection
- Logical organization
- Physical organization.

Relocation

Relocation is a basic requirement of memory management. For simplification purposes let us assume that the process image occupies a contiguous region of main memory. The operating system need to know the location of:

- Process control information
- Execution stack
- Entry point to begin the execution of a program.
  - ‘Processor’ must deal with memory references within the program.
  - ‘Branch instructions’ contain an address to reference the instruction to be executed next.
  - ‘Data reference instructions’ contain the address of the byte or word of data referenced.

The figure below depicts the above concept:



Sharing

Any protection mechanism must have the flexibility to allow several processes to access the same portion of main memory.

E.g., if numbers of processes are executing the same program then it is advantageous to allow each process to access the same copy of the program rather than its own separate copy.

Protection

Once we have two programs in memory at the same time there is a danger that one program can write to the address space of another program. Every process should be protected against unwanted interference by other processes. Satisfaction of the relocation requirement increases the difficulty of satisfying the protection requirement.CPU tend to support absolute addressing which means that code runs differently when loaded in different places.

It is not possible to check the absolute address at compile time. Most of the programming languages allow the dynamic calculation of address at run time.

Logical Organization

Main memory is organized as a linear or one-dimensional address space that consists of sequence of bytes or words. Secondary memory at its physical level is similarly organized. Most of the programs are organized into modules.

Advantages of Modules

- Modules can be written and compiled independently
- Protection (such as read-only, execute-only) is given to different modules.
- Introduce mechanism by which modules can be shared among processes.

Physical Organization

Computer memory is organized into two levels:

- **Main memory** - Main memory is a volatile memory and it provides fast access at relatively high cost.
- **Secondary memory** - Secondary memory is a non-volatile memory and it is slower and cheaper than main memory.

The organization of the flow of information between main memory and secondary memory is a major system concern. It is impractical for the programmer, because of the following reasons.

- Main memory available for a program plus its data, may be insufficient. In this case programmer must use the mechanism known as "overlaying". That is keep in memory only those instructions and data that are needed at any given time.
- In a multiprogramming environment, programmer does not know how much space is available and that space is at where at the time of coding.

There are two main ways to make code relocatable:

- We can have the compiler/linker mark all the address in the code and rewrite them all when the program is loaded. This adds start up overhead.
- Use only relative addressing and require the OS to set a base register to the beginning of the code. This is particularly useful in base/limit protected system because the same base register can be used for relocation and protection.
- Relocation without protection is dangerous. It is possible to create an absolute addressing instruction and use it to access some other processes memory.
- A solution which solves both the relocation and protection problem is to use two registers called base and limit registers. The base register stores the start address of the position and the limit register holds the length of the portions.
- Any address that is generated by the program has the base register added to it. In addition all addresses are checked to ensure that they are within the range of partition.
- An additional benefit of this scheme is that if a program is moved within memory only its base register need to be amended.

Address Binding

Usually, a program resides on a disk as a binary executable file. To be executed, the program must be brought into memory and placed within a process. Depending on the memory management in use, the process may be moved between disk and memory during its execution. The processes on the disk that are waiting to be brought into memory for execution from the input queue.

The normal procedure is to select one of the processes in the input queue and to load that process into memory. As the process is executed, it accesses instructions and data from memory. Eventually, the process terminates, and its memory space is declared available.

Most systems allow a user process to reside in any part of the physical memory. Thus, although the address space of the computer starts at 00000, the first address of the user process need not be 00000. This approach affects the addresses that the user program can use. In most cases, a user program will go through several steps- some of which may be optional- before being executed. Addresses in the source program are generally symbolic (such as count). a compiler will typically bind these symbolic addresses to relocatable addresses(such as "14 bytes from the beginning of this module"). The linkage editor or loader will in turn bind the relocatable addresses to absolute addresses (such as 74014). Each binding is mapping from one address space to another.

Address Binding is a process which fixes a physical address to the logical address of a process' address space. There are various types of binding as below:

- **Compile time binding** - If the program location is fixed and it is known at compile time where the process will reside in memory is called compile time binding. If at some later time the starting location changes then it will be necessary to recompile this code. The MS-DOS .COM format programs are bound at compile time.

- **Load time binding** - If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code. In this case, final binding is delayed until load time. If the starting address changes, we need only reload the user code to incorporate this changed value.
- **Execution time binding** - If the process can moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware must be available for this scheme to work. Most general-purpose operating systems use this method.

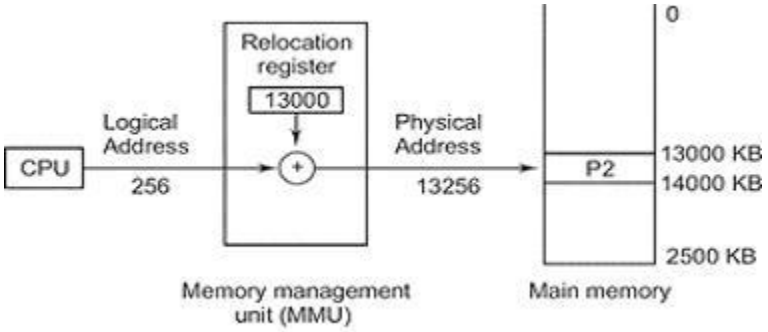
Logical & Physical Address Space

The concept of logical address space that is bound to a separate physical address space is central to proper memory management.

Logical address is the address that is generated by the CPU. It is also referred to as virtual address.

Physical address is the address as seen by the memory management unit that is the one that is loaded into the memory address register of the memory.

For example, P2 is a program, with size 256 KB. But program is loaded in the main memory from 13000 to 13256 KB; this address is called physical address.



Physical address space = Logical address space + relocation register value

13256 = 256 + 13000

In compile-time and load-time address-binding schemes logical and physical addresses are same. Whereas in execution-time the address-binding schemes will differ. Here logical address is virtual-address.

The set of all logical addresses (generated by a program) is referred ta as Logical address

space. The set of all Physical addresses corresponding to these logical addresses is referred to as physical address space. The run-time mapping from logical (virtual) to physical addresses is done by the MMU (memory-management unit), which is a hardware device.We can choose from many different methods to accomplish such mapping. This will be discussed in the next article.

Base register is called relocation register. User program never sees the real physical addresses. User program deals with the logical addresses only.

Source: http://www.go4expert.com/articles/memory-management-basics-t22350/