Operating Systems

Karim Lounis

Queen’s University, Canada

Winter’19, Jan-Apr 2019
Memory management
Logical and physical addresses

Every byte in the memory has a specific address that may range from 0 to some maximum, known as **physical address**.

  e.g., if the address size \( M = 32\text{bits} \) then addresses range from \( 0x00000000 \) to \( 0xFFFFFFFF \)

Whenever a program is brought into memory, it occupies a certain number of memory locations, and the set of all physical addresses used by that program is called **physical address space**.

  e.g., from \( 0xCD18B235 \) to \( 0xCD18C000 \)

During compilation, all variables and procedures in the source code of a program are assigned some specific addresses, known as **Logical addresses**, where the set of these address is called **logical address space**.

  i.e., in a machine code, all references to data and code are made by specifying the logical addresses and not by the variable or procedure names
Address binding

Definition

**Binding.** Is the process of setting up some relationships between two entities or items.

Initially, there is no correspondence between a program identifier space and the memory address space.

To execute a program, we need to map its program identifier space into memory address space.

Definition

**Address Binding.** is a mapping from one address space to another.

It can take place at one of the following times: Compile time, loader time, and execution time.
Address binding

1 Compile time. If it is known, at the compile time, which addresses the program will occupy in the main memory. In this case, logical addresses are the same as that of physical addresses [static allocation].

\[(\text{symbolic } @ \xrightarrow{\text{compile}} \text{ absolute } @)\]

In this scheme a binary program is called absolute binary image.

The program should always be loaded in the same location.

Used in single-task operating systems e.g., *.com files in MS-DOS.
Address binding

Load time. If it is not known, at compile time, which addresses the program will occupy in the main memory, the compiler generates relocatable addresses that will be converted into absolute addresses at the load time.

(symbolic \( \@ \) \( \xrightarrow{\text{compile}} \) relocatable \( \@ \) \( \xrightarrow{\text{load}} \) absolute \( \@ \))

Used in multiprogramming systems where it may not be possible to know in advance where in the main memory a binary image will be loaded.

It depends on which partition is available \[\text{dynamic allocation}\].

In this scheme a binary program is called relocatable binary image.

Relocatable address \( r \) also called relative addresses (relative to 0x000).

The relocatable loader updates all relocatable addresses depending on where in the main memory the image is loaded.
Address binding

Execution time. If a process can be moved during its execution from one memory partition to another, then binding should be delayed till runtime [dynamic relocation].

(symbolic $\rightarrow$ compile + loader $\rightarrow$ relocatable $\rightarrow$ execution $\rightarrow$ absolute)

When the CPU executes a program, it generates relative addresses and not physical memory addresses.

Those relative addresses are popularly known as logical addresses.

Logical addresses are translated at runtime by a dedicated hardware known as MMU (Memory Management Unit).

The Memory Management Unit can be implemented in different ways depending on how main memory is allocated.
Logical and Physical address space

Definition

**Logical address:** A.k.a, virtual address, is a memory address that is generated by the CPU (what the program sees) during program execution.

*Example:* in `Mov Ax, [0xF23E]` the address `[0xF23E]` is a logical address.

The user uses the logical address to access the physical address.

The set of all logical addresses generated by a program are called the program logical address space.

Definition

**Physical address:** is a memory address that is seen by the memory unit (the real address in RAM) that points to a physical memory location.

*Example:* logical addresses are translated/mapped by the a hardware component called MMU (Memory Management Unit) into physical addresses.
Swapping

Definition

**Swapping**: Is the process of moving a program (i.e., its address space) from the RAM to the hard disk drive (swap out) or the reverse (swap in).

It allows to;

- Virtually increase the size of the main memory.
- Increase the degree of multiprogramming.

It is performed by an operating system module called dispatched.

- It has a cost: For a 100MB process to be moved to HDD at 50MBps, it takes around $2 + \lambda$ seconds.
- On modern OSs, swapping is trigged when the total of unused (available) memory falls below a threshold amount.
- Another approach consists of swapping a portion of a program and not the whole program.
- Not used in mobile OSs, where OSs terminates processes if memory is needed.
Memory Allocation

Memory allocation strategies
Memory Allocation

There are various strategies that are used to manage the memory. All these strategies allocate memory to processes using either of the following two approaches:

- Contiguous memory allocation.
  - Single partition Vs multiple partitions.
  - Fixed size Vs Variable size.
- Noncontiguous memory allocation.
  - Segmentation Vs paging.
Single partition (One Contiguous Zone)

In this strategy, the main memory is divided into two zones, one to store the OS code and the other for user programs.

- In single-task operating systems, only one program can be loaded at a time.
- Programs are limited by the size of the user space.
- The memory is not well exploited (used and shared).
- A $\lambda$-Byte program can occupy the whole user space.
Multiple partitions

In this strategy, the main memory is divided into two zones, one to store the OS code and the other for multiple user programs.

**Multiple static partitions.** Partition have the same size (a system parameter). It was originally used by IBM OS/360.

- Used by multi-task OSs where multiple programs can be loaded at a time.
- The user zone is divided into multiple partitions of equal size.
- Degree of multiprogramming is bound by the number of partitions.
- Internal fragmentation.
- No longer used.
Multiple partitions.

In this strategy, the main memory is divided into two zones, one to store the OS code and the other for multiple user programs.

**Multiple variable partitions.** Partition have variable sizes (depending on program needs).

- Used by multi-task operating systems where multiple programs can be loaded at a time.
- The user zone is divided into multiple partitions of variable size.
- A base address and a size are used in PDT (Partition Description Table).
- External fragmentation.
Fragmentation

There are two types of fragmentation:

1. **External fragmentation.** Happens when there is a certain amount of non-contiguous memory space available but cannot be used by some programs.

**Compaction.** Relocate programs to bundle the small memory holes into a large whole.

**Segmentation and/or Paging.** Program address space is not contiguous.
Fragmentation

There are two types of fragmentation:

1. **Internal fragmentation.**
   happens when a program is using a memory partition which size is larger than the program.

2. **Segmentation and/or Paging.**
   program address space is not contiguous.
Segmentation and Paging after reading week!
End.