Memory Management: Main Memory

1 Usage of Memory

1.1 Memory Sharing

As a result of CPU scheduling, we can improve:

- the utilisation of the CPU
- the speed of the computer's response to its users

To realise this increase in performance, however, we must be able to keep several processes in memory at once

1.2 Types of Memory

Memory comes in many types:

- Cache
- Main Memory
- Storage Memory
- Virtual Memory

The cache and main memory are referred to as primary storage. The cache primarily supports, and it managed by, the CPU. It is invisible to the operating system. It uses the same memory management approaches as main memory, therefore we are going to focus on main memory

1.3 Logical/Physical Addressing

- **Logical address**: The address generated by the CPU
- **Physical address**: The address seen by the memory management unit
- **Memory Management Unit (MMU) scheme**: 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 (application) program deals with logical addresses; it never sees the real physical address

1.4 Memory Partitioning

Memory is a finite resource and as with the CPU, needs managed to be used as efficiently as possible

The main memory is usually split into two partitions:

- Kernel processes are usually held in one partition
- User processes are held in another partition

Each partition is contained in a single contiguous section of memory

1.5 Contiguous allocation

The main memory is usually split into two partitions:

- The resident operating system is usually held in low memory with the interrupt vector
- User processes are then held in high memory
- Each process is contained in a single contiguous section of memory

Basic hardware

- Relocation register scheme is used to protect user processes from each other, and from changing operating-system code and data
- The relocation register contains the value of the smallest physical address
- The limit register contains the range of logical addresses and each logical address must be less than the limit register

Multiple-partition allocation:

- Hole: block of available memory
- Holes of various sizes are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- The OS maintains information about allocated partitions and free partitions (holes)

1.6 Fragmentation

External fragmentation
External fragmentation is memory space that exists to satisfy a request but it is not contiguous

We can reduce external fragmentation by **compaction**:

- Shuffle memory contents to place all free memory in one block
- Only possible if relocation is dynamic, and is done at execution time

Internal fragmentation
Internal fragmentation is allocated memory that may be slightly larger than requested memory.
This size difference is memory internal to a partition, but not being used

2 Paging

The logical address space of a process can be non contiguous: the process is allocated physical memory whenever it is available

The physical memory is divided into fixed size blocks called **frames**
Typically, the size is a power of 2, between 512 and 8192 bytes

The logical memory is divided into blocks of the same size called **pages**

The OS sets up a page table to translate logical to physical addresses

Paging then leads to internal fragmentation

2.1 Address translation scheme

The address generated by the CPU is divided into:

- Page number (p): used as an index into a page table which contains the base address of each page in physical memory
- Page offset (d): combined with base address to define the physical memory address that is sent to the memory unit

2.2 Protection

Memory implementation implemented by associating protection bit with each frame.

A **Valid-invalid bit** is attached to each entry in the page table:

- ”vaid” indicates that the associated page is in the process: logical address space, and thus a legal page
- ”invalid” indicates that the page is not in the process: logical address space

3 Virtual memory

Definition:
Virtual memory is the capability of the operating system that enables programs to address more memory locations than are actually provided in main memory

Virtual memory systems help remove much of the burden of memory management from the programmers, freeing them to concentrate on application development.

- The logical address space is much larger than the physical address space
- Pages are swapped in and out of main memory
- The physical address space is shared by several processes
- Only part of the process needs to be in the physical address space for execution
- A free frame list of the physical address space is maintained by the operating system

3.1 Virtual address space

Each process views the address space as a contiguous block of memory holding the objects it needs to execute

- Code
- Data
- Heap
- Stack

3.2 Instruction Execute Cycle

A typical instruction execute cycle

- Fetch an instruction from memory
- Decode the instruction
- Execute the instruction
  This may cause operands to be fetched into memory
- After the instruction acts over the operands the result may need to be stored in memory

3.3 Binding of Instructions

Program must be brought into memory and placed within a process for it to be executed from the ready queue

Address binding of the instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known, absolute code can be generated
- **Load time** Must generate relocatable code if memory location is not known at compile time
- **Execution time**: Binding delayed until run-time if the process can be moved during its execution from one memory segment to another
  Need hardware support for address maps e.g. base and limit registers

3.4 Dynamic Loading

- Better memory space utilization (unused routine is never loaded)

- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required (implemented through program design)

3.5 Dynamic Linking

- Dynamic linking for system libraries, postpones until execution time
- A small piece of code, called the stub, is used to locate the appropriate memory-resident library routine
- The Stub replaces itself with the address of the routine, and executes the routine
- The Operating system is needed to check if a called routine is in some process: memory allocation
- All processes that use a library execute the same copy of the library

3.6 Swapping

A process can be **swapped** temporarily out of memory to a backing store, and brought back for continued execution

A backing store is a fast disk large enough to accommodate copies of all memory images for all uses and must provide direct access to these memory images

Roll out, roll in: swapping variant used for priority bases scheduling algorithms: lower priority process is swapped ot so higher priority processes can be loaded and executed

Major part of swap time is transfer time: total transfer time is directly proportional to the amount of memory swapped

4 Memory Allocation

4.1 Dynamic storage problem

How to satisfy a request of size n from a list of free holes?

- **First-fit** Allocate the first hole that is big enough
- **Best-fit** Allocate the smallest hole that is big enough
  - Must search the entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst fit**: Allocate the largest hole:
  - Must search the entire list
  - Produces the largest leftover hole

First fit and best fir are usually better than worst fir in terms of speed and storage utilisation

4.2 Implementation of Page Table

- Page table it kept in main memory
- Page table base register (PTBR) points to the page table
- Page table length register (PTLR) indicates the size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data/instruction
  - The two memory access problem can be solved with the use of a special fast-lookup hardware cache called the associative registers or translation look-aside buffers (TLBs)

4.3 Shared spaces

Private code and data:

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Shared code:

- One copy of read only (re-entrant) code shared among processes (text editors, compilers, window systems)
- Shared code must appear in the same location in the logical address space of all processes

5 Segmentation

Segmentation: Memory-management scheme that supports the user view of memory.

A program is a collection of segments. A segment is a logical unit such as:

- Main program
- Procedure
- Function
- Local variables, global variables
- Common block
- Stack
- Symbol table, arrays...

5.1 Benefits and drawbacks of Segmentation

Benefits:

- Protection is improved because segments represent semantically defined portions of the program, therefore instructions can be read only whereas data can be written to
- Sharing of code segment level is easier due to the read only properties

Issues:

- Like with contiguous allocation: *fragmentation*
- Like with paging: *segment table*