OPERATING SYSTEM II ASSIGNMENT

(MEMORY MANAGEMENT)

SUBMITTED BY

TIJANI MUSTAPHA ADEKUNLE

H/CS/19/0711

SUBMITTED TO:

DADDY ADEGBOYE O.J

ON

11TH OF APRIL, 2020

WHAT IS MEMORY MANAGEMENT?

Memory management is the process by which the operating system manages and controls the allocation, accessibility and usage of the memory (primary memory). Memory management activities performed by the operating system are:

- Keeping track of the memory (i.e., knowing the status of the memory)
- Allocation of process to the memory by specifying its usage and time usage in multiprogramming
- Memory de-allocation when not in use or needed by a process.

MEMORY MANAGEMENT TECHNIQUES

For the effective and efficient usage of the memory, these techniques are employ by the operating system:

1. Swapping
2. Fixed Partition
3. Paging
4. Segmentation

Swapping:

This is the process whereby a process is temporarily released from the memory into the backing store for another process to be allocated to the memory. The temporarily released process is called **swapped-out or roll-out process.** The newly allocated process is called **swapped-in or roll-in process.**

The swapped-out process later be allocated to the same memory which it was swapped-out from (depending on the execution binding used) for execution (this happens when a required action, like IO requirements, is been completed).

Actions that lead to swapping:

1. Expiration of Quantum
2. Scheduling and rescheduling by the CPU scheduler
3. Arrival of higher priority processes will swap out lower priority processes
4. Awaiting a IO completion

Major things to note about swapping:

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images and must be of a large size to accommodate the copies of memory images for all users.
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process 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
- Process is reallocated to the memory at which it was swapped out only if binding is done at load time or assembly time
- Process is reallocated to a new memory if biding is done at execution time
- The time taken for processes to be swapped is relatively small (i.e., in **milliseconds**)

- Before a process could be swapped, it must be idle
- Swapping is normally disabled by default, but starts when there are lots of running process and a limited amount of memory
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

Paging

Paging is a memory management technique that permits the physical memory of a process to be noncontagious (it avoids the physical or logical memory of a process from getting contagious). It solves the problem of fitting different sizes memory chunks into the backing store.

Paging helps to increase the efficiency of sapping by reducing the swapping time. Paging is accepted, supported and handled by hardware. However, hardware and the operating system are being closely integrated in recent development (like the 64 bits architecture microprocessors) to implement paging.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages of Paging

1. Ability to share code (reentrant codes) among users
2. Ability to share memory
3. Paging reduces external fragmentation.
4. Paging is simple to implement and assumed as an efficient memory management technique.
5. Due to equal size of the pages and frames, swapping becomes very easy.

Disadvantages of Paging

1. Page table requires extra memory space, so may not be good for a system having small RAM.

Method of Implementing Paging

For Paging to occur, physical memory must be broken down into fixed-size blocks known as frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source. The backing store is divided into fixed-sized blocks that are of the san1.e size as the memory frames. Every address generated the CPU is divided into two parts: a {p) and a. The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit. The page size (like the frame size) is defined by the hardware. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.

Keys to Note about Paging

- Page table is kept in main memory
- Page-table base register (PTBR) points to the page table
- Page-table length register (PRLR) indicates 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 by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs)

Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length

of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

Methods of Implementing Segmentation

Segmentation is a memory-management scheme that supports this user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset. For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a two tuple: