**Operating Systems**

**Assignment 2**

Sarthak Bhatia

07714803118

**Ques.1 a)Differentiate between Multiprogramming and Multitasking.**

|  |  |
| --- | --- |
| **Multi Programming** | **Multi Tasking** |
| 1. In Multiprogramming, at the same time, we can run the multiple processes concurrently on one processor. | 1. In Multitasking, at the same time, we can execute multiple tasks by the use of multiple CPUs. |
| 1. Multiprogramming is implemented by using the concept of context switching | 2. Multitasking is implemented by using the concept of time-sharing. |
| 1. In Multiprogramming, to execute the processes, only one CPU is used. | 3. In Multitasking, to allot a task we need Multiple CPUs. |
| 1. Multiprogramming requires more time to execute processes. | 4. Multitasking requires less time to execute processes. |
| 1. The reason to use multiprogramming is that we can reduce the CPU's ideal time. | 5. The reason to use multitasking is we can easily run multiple processes concurrently through time-sharing. |
| 1. Multiprocessing is more expensive. | 6. Multiprocessing is less expensive. |

**Ques1. b) Justify the statement ‘Performance degrades due to Frequent Context Switching’.**

**Ans.** The disadvantage of context switching is that it requires some time for context switching i.e. the context switching time. Time is required to save the context of one process that is in the running state and then get the context of another process that is about to come in the running state. During that time, there is no useful work done by the CPU from the user perspective. So, context switching is pure overhead in this condition.

**Ques. 1 c) If a solution to a critical section problem does n0t satisfy the ‘Bounded waiting’ condition what will be the consequences?**

**Ans.** If the critical section problem does not satisfy the ‘bounded waiting’ condition, the process would have to wait forever into the critical section. Thus, there should be a boundary on getting chances to enter into the critical section, otherwise if the condition is not satisfied, there is a possibility of starvation.

**Ques. 1 d) Show how contiguous memory allocation leads to external fragmentation.**

**Ans.** In Contiguous memory allocation which is a memory management technique, whenever there is a request by the user process for the memory then a single section of the contiguous memory block is given to that process according to its requirement. Contiguous Memory allocation is achieved just by dividing the memory into the fixed-sized partition.

The memory can be divided either in the fixed-sized partition or in the variable-sized partition in order to allocate contiguous space to user processes.

As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will be no external fragmentation. Let us understand this with the help of an example: In the above diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. There is a Process P4 of size 15 MB. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.

**Ques. 1 e) Compare various types of page tables.**

**Ans.**

* **Inverted page tables-**It is best to think of the inverted page table (IPT) as an off-chip extension of the TLB using normal system [**RAM**](https://www.minitool.com/lib/random-access-memory.html). Unlike the true page table, it may not be able to save all current mappings. The operating system must be prepared to handle misses, just like using MIPS-style software-filled TLB. IPT combines a page table and a frame table into one data structure. The core of a fixed-size table is that the number of rows equals the number of frames in memory. If there are 4,000 frames, the inverted page table has 4,000 rows. Each row has an entry for the virtual page number (VPN), the physical page number (not the physical address), some other data, and the method of creating the collision chain.
* **Multilevel page tables-**Multilevel page tables are also called “hierarchical page tables”. Although the inverted page table keeps a list of the mappings installed by all frames in physical memory, this can be very wasteful. To avoid this, we can create a page table structure containing virtual page mapping. This is done by keeping several page tables covering a certain block of virtual memory. For example, we can create smaller 4K pages with 1024 entries, covering 4M virtual memory. This is useful because the top and bottom layers of virtual memory are usually used when the process is running - the top layer is usually used for text and data segments, and the bottom layer is usually used for the stack, with free memory in between. Multilevel page tables can reserve some smaller page tables to cover only the top and bottom of the memory, and create new page tables only when necessary.Now, each of these smaller page tables is linked together by the master page table, effectively creating a tree data structure. Not only are two levels required, but multiple levels may be required. The virtual address in this mode can be divided into three parts: the index in the root page table, the index in the subpage table, and the offset in the page.
* **Virtualized page tables-** Creating a page table structure that contains the mapping of each virtual page in the virtual address space may end up being wasteful. However, the excessive space problem can be solved by placing the page table in virtual memory and letting the virtual memory system manage the memory of the page table.However, a part of this linear page table structure must always reside in physical memory to prevent circular [**page faults**](https://www.minitool.com/lib/page-fault.html) and find key parts of the page table that do not exist in the page table.
* **Nested page tables-** You can implement nested page tables to improve the performance of hardware virtualization. By providing hardware support for page-table virtualization, simulation requirements are greatly reduced. For x86 virtualization, the current choices are Intel's Extended Page Table feature and AMD's Rapid Virtualization Indexing feature.

# Ques. 2 a) Let us assume a 24 bit physical address space and 20 bit logical address space. The page size is 1 KB. Calculate:

**Ans.**

1. **Number of page**

Logical address/page\_size= 2^20/2^3= 2^17.

# Number of frames

The size of a frame is the same as that of a page, so the size of a frame is 1 KB. Thus, the number of frames is 2^17.

# Number of bits to denote page number

17 bits.

# Number of bits to represent offset

3 bits.

# Maximum amount of internal fragmentation

Process size= 2^20. Number of pages= 2^17. Page size 1KB.

Internal fragmentation= 1000- 256= 744 bytes.

**Ques. 2 b) Write the pseudo code for Bakery Algorithm to solve n process mutual exclusion.Explain it also.**

**Ans.**

repeat

choosing[i] := true

number[i] := max(number[0], number[1], ..., number[n - 1])+1;

choosing[i] := false;

for j := 0 to n - 1

do begin

while choosing[j] do no-op;

while number[j] != 0

and (number[j], j) < (number[i], i) do no-op;

end;

critical section

number[i] := 0

remainder section

until false;

# Explanation –

Firstly the process sets its “choosing” variable to be TRUE indicating its intent to enter critical section. Then it gets assigned the highest ticket number corresponding to other processes. Then the “choosing” variable is set to FALSE indicating that it now has a new ticket number. This is in- fact the most important and confusing part of the algorithm.

It is actually a small critical section in itself ! The very purpose of the first three lines is that if a process is modifying its TICKET value then at that time some other process should not be allowed to check its old ticket value which is now obsolete. This is why inside the for loop

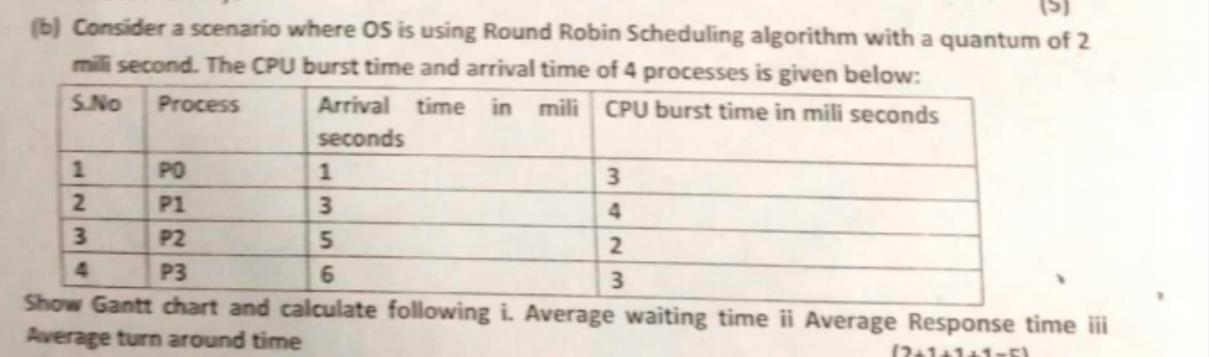
before checking ticket value we first make sure that all other processes have the “choosing” variable as FALSE.

After that we proceed to check the ticket values of processes where the process with least ticket number/process id gets inside the critical section. The exit section just resets the ticket value to zero.

**Ques. 3 a) Which one of the operating system architectures would you select for a real time operating system? Why?**

**Ans.** Micro kernel Architecture- Microkernel is one of the classification of the kernel. Being a kernel it manages all system resources. But in a microkernel, the **user services** and **kernel services** are implemented in different address space. The user services are kept in **user address space**, and kernel services are kept under **kernel address space**, thus also reducing the size of kernel and size of operating system as well.

**Ques. 3 b)**

****

**Ans.**

|  |  |  |
| --- | --- | --- |
| name | response time | wait time |
| p0 | 5 | 2 |
| p1 | 7 | 3 |
| p2 | 3 | 1 |
| p3 | 7 | 4 |

Average waiting time is 2.5

Average response time is 5.5

Sequence is like that ->p0->p1->p0->p2->p1->p3->p3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P0 | P1 | P0 | P2 | P1 | P3 | P3 |
| 1 | 3 | 5 | 6 | 8 | 10 | 12 |

**Ques. 4 a)** ‘**Paging solves the problem of one type of fragmentation but creates the problem of another type of fragmentation.’ Do you agree? Why or Why not?**

**Ans.** Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

* + Paging reduces external fragmentation, but still suffers from internal fragmentation.
  + Paging is simple to implement and assumed as an efficient memory management technique.
  + Due to equal size of the pages and frames, swapping becomes very easy.
* Page table requires extra memory space, so may not be good for a system having small RAM.

**Ques. 4 b) List the contents of the Process Control Block and explain its significance. Will there be a separate PCB for each thread?**

**Ans.** A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc. The process table is an array of PCB's that logically contains a PCB for all of the current processes in the system.

* **Pointer –** It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
* **Process state –** It stores the respective state of the process.
* **Process number –** Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
* **Program counter –** It stores the counter which contains the address of the next instruction that is to be executed for the process.
* **Register –** These are the CPU registers which includes: accumulator, base, registers and general purpose registers.
* **Memory limits –** This field contains the information about memory management systems used by operating systems. This may include the page tables, segment tables etc.
* **Open files list –** This information includes the list of files opened for a process.

In a multithreaded process, all of the process' **threads** share the same memory and open files. Within the shared memory, **each thread** gets **its own** stack. An operating system **had** to keep track of processes, and stored **its per**-process information in a data structure called a process control block (**PCB**).