Thank you!

# SYSC4001B
# Operating Systems

Grade / ???

## Assignment 3
**Date Due: 5-03-2021**
**Time Due: Before 21:00 EDT**
**Student: Gabriel Benni K. Evensen [101119814]**
**Student: Omar Jemal [101080896]**

---

- Teams of two.
- One submission per team.
- The programming bit of the project is to be done using the Pair Programming technique.
- Submit the assignment electronically via the cuLearn portal.

---

1. **Part I:- Concepts [40 marks]**

   (a) Consider the following page reference string in an OS with a Demand Paging memory management strategy:- **[12 marks]**

   201, 302, 203, 404, 302, 201, 205, 206, 302, 201, 302, 203, 207, 206, 203, 302, 201, 302, 203, 206

   i. How many page faults will occur for the following page replacement algorithms when 3 frames are allocated to the program? Use:-
      A. LRU
      B. FIFO
      C. Optimal page replacement strategy

   ii. Repeat (i) for 5 frames allocated to the program.

   **Answer:-**

   i. Answers are below in the tables:-
      A. LRU:- 15 total page faults

| LRU | Reference string | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pages | 201 | 302 | 203 | 404 | 302 | 201 | 205 | 206 | 302 | 201 | 302 | 203 | 207 | 206 | 203 | 302 | 201 | 302 | 203 | 206 |
| | 201 | 201 | 201 | 404 | | 404 | 205 | 205 | 205 | 201 | | 201 | 207 | 207 | | 302 | 302 | | | 302 |
| | 0 | 302 | 302 | 302 | | 302 | 302 | 206 | 206 | 206 | | 203 | 203 | 203 | | 203 | 203 | | | 203 |
| | 0 | 0 | 203 | 203 | | 201 | 201 | 201 | 302 | 302 | | 302 | 302 | 206 | | 206 | 201 | | | 206 |
| Total page faults = 15 | | | | | | | | | | | | | | | | | | | | |

Figure 1: LRU page fault table

B. FIFO:- 16 total page faults

| FIFO | Reference string | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 201 | 302 | 203 | 404 | 302 | 201 | 205 | 206 | 302 | 201 | 302 | 203 | 207 | 206 | 203 | 302 | 201 | 302 | 203 | 206 |
| Pages | 201 | 201 | 201 | 404 | | 404 | 404 | 206 | 206 | 206 | | 203 | 203 | 203 | | 302 | 302 | | 302 | 206 |
| | 0 | 302 | 302 | 302 | | 201 | 201 | 201 | 302 | 302 | | 302 | 207 | 207 | | 207 | 201 | | 201 | 201 |
| | 0 | 0 | 203 | 203 | | 203 | 205 | 205 | 205 | 201 | | 201 | 201 | 206 | | 206 | 206 | | 203 | 203 |
| Total page faults = 16 | | | | | | | | | | | | | | | | | | | | |

Figure 2: FIFO page fault table

C. Optimal page replacement strategy:- 11 page faults

| OPT | Reference string | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 201 | 302 | 203 | 404 | 302 | 201 | 205 | 206 | 302 | 201 | 302 | 203 | 207 | 206 | 203 | 302 | 201 | 302 | 203 | 206 |
| Pages | 201 | 201 | 201 | 201 | | | 201 | 201 | | | | 203 | 203 | | | 203 | 203 | | | 206 |
| | 0 | 302 | 302 | 302 | | | 302 | 302 | | | | 302 | 207 | | | 302 | 302 | | | 302 |
| | 0 | 0 | 203 | 404 | | | 205 | 206 | | | | 206 | 206 | | | 206 | 201 | | | 201 |
| Total page faults = 11 | | | | | | | | | | | | | | | | | | | | |

Figure 3: OPT page fault table

ii. Part II answers are below in the tables:-

A. LRU:- 8 total page faults

| LRU | Reference string | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 201 | 302 | 203 | 404 | 302 | 201 | 205 | 206 | 302 | 201 | 302 | 203 | 207 | 206 | 203 | 302 | 201 | 302 | 203 | 206 |
| Pages | 201 | 201 | 201 | 201 | | | 201 | 201 | | | | 201 | 201 | | | | | | | |
| | 0 | 302 | 302 | 302 | | | 302 | 302 | | | | 302 | 302 | | | | | | | |
| | 0 | 0 | 203 | 203 | | | 203 | 206 | | | | 206 | 206 | | | | | | | |
| | 0 | 0 | 0 | 404 | | | 404 | 404 | | | | 203 | 203 | | | | | | | |
| | 0 | 0 | 0 | 0 | | | 205 | 205 | | | | 205 | 207 | | | | | | | |
| Total page faults = 8 | | | | | | | | | | | | | | | | | | | | |

Figure 4: LRU page fault table

B. FIFO:- 10 total page faults

| FIFO | Reference string | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 201 | 302 | 203 | 404 | 302 | 201 | 205 | 206 | 302 | 201 | 302 | 203 | 207 | 206 | 203 | 302 | 201 | 302 | 203 | 206 |
| Pages | 201 | 201 | 201 | 201 | | | 201 | 206 | | 206 | 206 | 206 | 206 | | | | | | | |
| | 0 | 302 | 302 | 302 | | | 302 | 302 | | 201 | 201 | 201 | 201 | | | | | | | |
| | 0 | 0 | 203 | 203 | | | 203 | 203 | | 203 | 302 | 302 | 302 | | | | | | | |
| | 0 | 0 | 0 | 404 | | | 404 | 404 | | 404 | 404 | 203 | 203 | | | | | | | |
| | 0 | 0 | 0 | 0 | | | 205 | 205 | | 205 | 205 | 205 | 207 | | | | | | | |
| Total page faults = 10 | | | | | | | | | | | | | | | | | | | | |

Figure 5: FIFO page fault table

C. Optimal page replacement strategy:- 7 page faults

OPT

| Pages | Reference string | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 201 | 302 | 203 | 404 | 302 | 201 | 205 | 206 | 302 | 201 | 302 | 203 | 207 | 206 | 203 | 302 | 201 | 302 | 203 | 206 |
| | 201 | 201 | 201 | 201 | | | 201 | 201 | | | | | 201 | | | | | | | |
| | 0 | 302 | 302 | 302 | | | 302 | 302 | | | | | 302 | | | | | | | |
| | 0 | 0 | 203 | 203 | | | 203 | 203 | | | | | 203 | | | | | | | |
| | 0 | 0 | 0 | 404 | | | 404 | 206 | | | | | 206 | | | | | | | |
| | 0 | 0 | 0 | 0 | | | 205 | 205 | | | | | 207 | | | | | | | |
| Total page faults = 7 | | | | | | | | | | | | | | | | | | | | |

Figure 6: OPT page fault table

---

(b) Consider a system with memory mapping done on a page basis and using a single-level page table. Assume that the necessary page table is always in memory and a memory reference takes 250ns. **[6 marks]**

i. How long does a paged memory reference take? Explain your answer.

ii. If we add TLB that imposes an overhead of 30ns on a hit or a miss. If we assume that 80% of all memory references hit in the TLB, what is the EAT? Explain your answer.

iii. Why adding another layer, TLB, can improve performance? Are there situations where the performance may be worse with TLB than without TLB? Explain.

**Answer:-**

i. A paged memory references takes two memory accesses; one for the page lookup, and another for the actual memory access (i.e., we need two different memory read operations). Thereby we get:-

$$2 \times (250ns) = 500ns \tag{1}$$

ii. Since we are expecting a 30ns overhead on both misses and hits, we should add this constant to our calculated 500ns value, and our given 250ns value. We expect our result to be greater than it otherwise would have been, had there been no imposed overhead, i.e., our memory accesses would have been performed faster without the overhead. Our equation is as follows:-

$$(1-p) \times ma + p \times \text{page fault time} = (1-0.80) \times (500ns + 30ns) + 0.80 \times (250ns + 30ns) = 330ns \tag{2}$$

Note that we could also have factored 30ns out of the equation for simplicity but we have included it as a way to show our understanding of its application.

iii. One benefit of adding another TLB is the improved performance; access to the page table is faster than without. The "engineering assumption" here is that the TLB is accurate and precise in finding the desired page table.

On the other hand, if our TLB is not accurate and precise it will lead to increased overhead due to frequent misses (i.e., page access time increases).

---

(c) Consider a paged logical address space (composed of 32 pages of 2 Kbytes each) mapped into a 1-Mbyte physical memory space **[6 marks]**

i. What is the format of the processors logical address?

ii. What is the length and width of the page table (disregarding all the control bits)?

iii. What is the effect on the page table if the physical memory space is reduced by half? Assume that the number of page entries and page size stay the same.

**Answer:-**

i. Since there are 32 pages we will need to calculate the log base two of this, follows:-

$$log_2(32) = 5 \tag{3}$$

Thus, it can be seen that we will need 5 bits for the addressing of each page.
The offset for each page is equal to:-

$$log_2(2048) = 11 \tag{4}$$

Thus, we will need 11 bits for the offset for each page.
Note that we used 2048, as opposed to 2000 as we are talking in terms of bits and bytes, meaning whole numbers only. If we were to take the log base two of 2000 we would get a value very close to 11, but slightly lesser than, and not whole. This can be done but it makes more sense to deal with the powers of two.

ii. The length is 32 as to map the logical addresses to the physical (1 MByte) addresses. In other words, to map the logical addresses to the physical corresponding ones, the length must be 32 pages.
The width of the page table, disregarding the control bits, is equal to the offset of the logical addresses (as calculated above) subtracted from the physical memory size. We can find the physical memory size is as follows:-

$$1MByte = 2^{20} \tag{5}$$

Recalling the previously calculated offset, $2^{11}$, we can find the width as follows:-

$$2^{20} - 2^{11} = 1046528 = 2^9 \tag{6}$$

Thus, it can be seen that the width of the page table is 9 bits.

iii. The number of pages decreases directly proportionally with the size of the memory space (not sure if this should be mentioned as we are told to assume the number of page entries remains). That is to say, if the physical memory space is reduced by half, then must also the number of pages be reduced by half. Moreover, the physical address space may be reduced by half (i.e., if we previously needed X bits to represent the addresses, we now need [X/2], from my understanding). Since our address bits is halved, our page width decreases by 1. Defragmentation has to happen when there are frames at the to-be-removed portion of the memory.

---

(d) Explain what a race condition is. Discuss a concrete examples of a race condition.
Why is disabling interrupts not a good mechanism to prevent race conditions? Explain and justify.
**[4 marks]**

**Answer:-**
Race conditions is a problem wherein several concurrent processes attempt to access data simultaneously. So, we do not know which process will access the data and at what time that access is. Simply put, from our understanding, the problem is in that the output given by the process accessing the data first is very much dependent on which process accessed the data last, it may have been modified.

At a lower level, a race condition is caused when a process/instruction/set of instructions is trying to both read and write at the same time, this of course is not possible and creates a problem.

If we disable interrupts we will essentially be creating a monopoly for the current running process as it cannot be switched by the CPU. If interrupts are disabled indefinitely it may lead to the system crashing. In other words, we have no way of dealing with events even of greater importance.

---

(e) Briefly explain how semaphores work **[2 marks]**

**Answer:-** A semaphore is a tool (hardware-involved) for ensuring that shared data used in critical sections of code is protected. Semaphores can demand that processes wait (wait()) until the current critical section is finished, this is called being "atomic". Semaphores end this waiting using a signal (signal()) function. Processes that are told to "wait()" must wait in some block until they receive the "all-good" signal() (assuming no error).

There are two types of semaphores; **counting semaphores**, and **binary semaphores**. A counting semaphore is not restricted to the positive integer domain, rather it is unrestricted. The binary semaphore may only have a value of 0, or 1. Binary semaphores are often used as a means of providing mutual exclusion. Counting semaphores are used to control how frequently access to a certain resource is provided. The semaphore is **initialized to the number of available resources**, and each **process must wait()** to use said resource. After the appropriate tests have been conducted (i.e., ensuring the semaphore is not 0) the **semaphore is decremented** and the **process is allowed to execute** (assuming tests passed) using the resource. (Silberchats, pp. 214)

---

(f) Explain, in detail, what an open operation must do. Consider the case of a file opened for APPEND, in a file system using a hierarchical directory structure **[4 marks]**

**Answer:-**

i. A system call is made (via open())
ii. File is located
iii. Check if we have the permissions for the file
iv. System call returns a pointer to the head of the location of the file such that we can use the file.
v. When we are finished with the file there is a system call to clear the aforementioned pointer to the head of the file location (via close()).

---

(g) (from Silberschatz) Consider a file system that uses inodes to represent files. Disk blocks are 8Kb in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system? **[4 marks]**

**Answer:-**
We are told that disk blocks are 8 KBytes in size, thus to find the number of pointers per block we must divide this by the required 4 bytes. Follows:-

$$\frac{8192bytes}{4bytes} = 2048 = 2^{11} \tag{7}$$

Thus, it can be seen that there are 2048 pointers per disk block, and there are 11 disk blocks. Since we have a single, double, triple indirect disk blocks. Thereby we arrive at the following equation:-

$$(11\times 8KBytes)+(2048^1\times 8KBytes)+(2048^2\times 8KBytes)+(2048^3\times 8KBytes) = 68753047648KBytes \tag{8}$$

Thus, we can see that the maximum file size that can be stored is 68753047648 KB. This is equal to 64.03 TB

(h) Explain what you can do in case (a) if you need to store a file that is larger than the maximum size computed. Give an example showing how you can define a larger file, and what the size of that file would be. **[2 marks]**

**Answer:-**
To increase a file size we must make the size of its respective block greater. If we wanted to change a 16 KB block's size to 32 KB we would do the following (12 direct blocks, 1 single indirect block, 1 double indirect block, 1 triple indirect block):-

$$(12 \times 2048) + 1(512^1 \times 2048) + 1(512^2 \times 2048) + 1(512^3 \times 2048) = 275415851008KB = 275.415TB \tag{9}$$

---

**References:-**


1. Silberschatz, Abraham. “Operating System Concepts.” pp. 3-500.
2. IBM Knowledge Center, ”What is JCL”
3. Advance in Electronic and Electric Engineering. ISSN 2231-1297, Volume 3, Number 2 (2013), pp. 155-162 © Research India Publications http://www.ripublication.com/aeee.htm
4. https://web.stanford.edu/ ouster/cgi-bin/cs140-spring14/lectures.php
5. Fair And Efficient C P U Scheduling Algorithm’s Jeyaprakash Chelladurai BTech, University of Madras, Chennai, Tamil Nadu (India), 2003
https://core.ac.uk/download/pdf/84872662.pdf
6. https://www.cs.hmc.edu/ sjung/cs110/hw2.htm
7. https://www.geeksforgeeks.org/difference-between-user-level-thread-and-kernel-level-thread/