Department of Electrical and Computer Engineering
Faculty of Engineering
King Abdulaziz University

EE463 | Operating System
Spring 2020

Project

Page Replacement using ESC and FIFO Method

Date of Submission: 20th of April 2020

Student	ID
Ruba Bin Jabal	1606486
Haneen Alsaedi	1606278

Instructor: Dr. Abdulghani Al-Qasimi

Contents

Table of Figures

Table of Tables

INTRODUCTION

The objective of this project is to learn what is involved in virtual memory management and implement the missing memory modules of OSP2 including page replacement strategies; which are enhanced second chance (ESC) and FIFO. The following classes was implemented; the main class, MMU (memory management unit), which is a hardware that is responsible for memory access in a computer. The other classes are FrameTableEntry, PageFaultHandler, PageTableEntry, and PageTable. Those classes are implemented, explained, and examined.

OVERVIEW & SYSTEM DESIGN

The memory management unit is responsible for providing access to the main memory and it is the gateway to memory for executing threads. Most modern operating system supports multiprogramming, thus, alow multiple processes/threads to be allocated in the memory simultaneously. Therefore, a basic require from memory management technique is to partition the main memory, in either fixed-size or variable-size, to be allocated by variant processes. The focus will be on utilizing fixed-size partitions by "paging" technique, which allow a noncontiguous allocation of the process's partitions in memory. In this technique, the partitions refer as a page, where each process may have a table of pages. Those pages collectively create its logical address space. the logical memory consists of page table and PTBR (page table base register), where each page could be referred by its logical address [page number | offset]. The partitions in logical/virtual memory is called page, whereas in physical memory called frames. The MMU responsible of converting a logical address into a physical address of the main memory frames, and it use PTBR to define the location of a page table that belongs to specific process. The key feature is that, normally the size of virtual memory is larger than physical memory, however, only one frame could be assigned to a exactly one page at a time.

The overall schema mapping virtual memory to physical memory is shown in Figure 1.

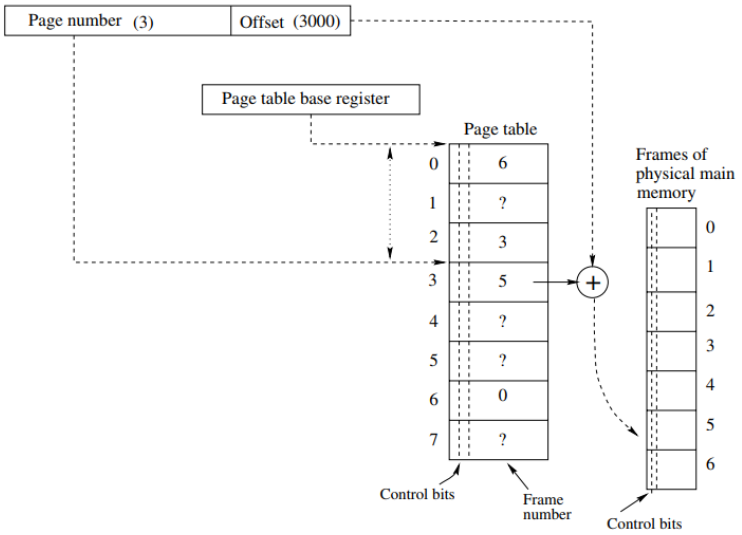

Figure 1 - Logical/Virtual address translation using page tables

- Important Terminology

Table 1 – Terminology and description

CHARACTERISTIC	DESCRIPTION
Locked Frames	A Locked frame has a positive lock count for the associated page. Lock operation is performed by the thread requesting I/O operation. When doing the request an I/O request block (IORB) is enqueued on the device's device queue. Locking prevent other thread to grab the frame.
Dirty Frames	A frame that its contents has been modified after the page was swapped into the frame.
Clean Frames	A frame that its contents has NOT been modified after the page was swapped into the frame.
Reserved bit	First reason: prevent other pagefault process to grab the frame. Thus, in pagefault handler the reserved bit set to true after finding suitable frame and set it to be false after finishing. Second reason: Reserving protects frames from the page replacement.
Reserved frames	Frame that its reserved bit is *true*.
Free Frame	frame that is reserved and doesn't has page assigned to it.

Validity bit	Indicates whether the page has a main-memory frame assigned to it. ***if*** *Validity bit=false* ***then*** *page in invalid* ***if*** *Validity bit=true* ***then*** *page in valid*
Reference bit	Indicates whether the frame has been referenced by a page.

- Page Fault and Page Replacement

A **pagefault** (kind of interrupt) occurs when a running thread makes a reference to invalid page, which is check by the MMU. Then the **pagefault handler** module is responsible to find and assign a suitable frame to the page. There is an important part to be highlighted which is the **swap device**. It is viewed as a real device with a special device number in OSP2. Thus, every task has a corresponding **swap file** on the swap device, which contains an exact image of the process memory. The swap device used as secondary storage to keep a copy of the entire process space, where running thread is mentioning to this copy in case it is referencing to invalid page. Thus, the swap file is used in the procedure of **pagefault handler.**

procedure of **pagefault handler** goes as following:

1. Suspend the thread that caused the interrupt and resume it before returning from the pagefault handler.
2. Find a suitable frame to assign to page, either by finding free frame if any or perform page replacement algorithm.
3. Assign the frame to the page and reserved it.
4. Perform a swap-in operation to ensure the frame has the exact image of the page which is available in the swap file as mentioned before.
5. Update the page table and set the page's validity bit to true.

In above step 2, in case of there are no frames, the page replacement algorithm is responsible of choosing a frame occupied by some other page and use it to satisfy the pagefault. The algorithms implemented in this project are ESC and FIFO. The first one gives the frame another chance if its reference bit is 1 by setting the reference bit to be 0. Otherwise select the frame that its reference bit

is 1 and not dirty (best choice) and freeing the frame. If all pages dirty, select one and do some operation; swapping out, freeing the frame, and swapping in. This algorithm avoids replacing heavily used page; however, it might need to search the frame table multiple time.

The FIFO algorithm is simpler, it chooses the older page for replacement. It requires creating FIFO queue where the new pages are added in the tail and select the pages from the head. The draw back is that it is likely to replace heavily used page.

METHODS DESCRIPTION

Class MMU

Method Name: **Init()**		
Description	The purpose of this method is to initialize the frame table and the static variables.	
Input	*Output*	*Precondition*
null	null	none

Method Name: **do_refer()**		
Description	The purpose of this method is to set the dirty and the reference bits if the page is valid. Otherwise, check the return of getValidatingThread() if it is null, which indicate the pagefault occurs due to the original thread, then initialize a pagefault interrupt. Then set also the dirty and the reference bits. If getValidatingThread() doesn't return null, means other thread of the same task has already caused a pagefault, thus, suspent the thread until the page becomes valid. It needs to define the page of the thread's logical memory by using getVirtualAddressBits() and getPageAddressBits() methods. getPageAddressBits() returns the number of bits for the page number getVirtualAddressBits() returns the number of bits in the Virtual Address(VA) where; No. of Bits for VA = No. of Bits for PN + No. of Bits for offset thus, No. of Bits for offset used to calculate the page size ($2^{\text{no. of bit for offset}}$), then get the page address (memoryAddress/page size), and obtain the page.	
Input	*Output*	*Precondition*
int memoryAddress, **int** referenceType, **ThreadCB** thread	null	none

Class FrameTableEntry

There were no methods defined this class, it was just required to define the constructor and call *super(frameID)* in first line. The constructor takes frame id as an input parameter.

Class PageTableEntry

Method Name: **do_lock()**		
Description	The purpose of this method is to increase the lock count on the page by one. The method increment lockCount if the page is valid, otherwise, if the page is invalid and validation event is present for the page, start page fault by calling PageFaultHandler.handlePageFault(). The method return ***FAILURE*** if the pagefault due to locking fails or the status of the created IORB becomes ThreadKill. Otherwise the method returns ***SUCCESS***	
Input	*Output*	*Precondition*
IORB iorb	Int (***SUCCESS or FAILURE)***	none

Method Name: **do_unlock()**		
Description	This method decreases the lock count on the page by one, but make sure the lockCount not become negative.	
Input	*Output*	*Precondition*
null	null	none

Class PageTable

The constructor must initialize the page table after calling *super(ownerTask)* at the first line. Before initializing the page table and fill it with items of type PageTableEntry, the page table size calculated using MMU.*getPageAddressBits*().

Method Name: **do_deallocateMemory()**		
Description	The purpose of this method is to frees up main memory occupied by the task. Then un-reserves the frames if it was reserved by the page's task. The freeing up conducted by un-set the dirty and the reference flags and setPage to null.	
Input	*Output*	*Precondition*
null	null	none

Class PageFaultHandler

Method Name: **numFreeFrames ()**		
Description	The purpose of this method is to count and returns the current number of free frames in the frame table. Free frame described in Table 1.	
Input	*Output*	*Precondition*
null	null	none

Method Name: **getFreeFrame()**		
Description	The purpose of this method is to returns the first free frame found in the frame table.	
Input	*Output*	*Precondition*
null	FrameTableEntry object	none

Method Name: **SecondChance()**		
Description	The following method implements page replacement algorithm which is the enhanced second chance. The idea of this algorithm is to provide another chance for the fame if its reference flag was set. So, during the search if reference flag=true, make it false and move to the next frame. Otherwise, frees up the clean frame and update the page table if the following condition satisfied; the frame's page is not null, the reference, dirty, and reserved flag is false, also the lock count is zero. As mentioned before, the reserve and flag prevent the page replacement method to take this frame. This method also saves the id of first dirty page to return it in case there is not enough clean page. The search must stop if the number of free frames become equal to the number of want free value which was defined in the MMU.	
Input	*Output*	*Precondition*
null	FrameTableEntry object	none

Method Name: **do_handlePageFault()**	
Description	The purpose of this method is to handle the page faults by getting free frame from the memory or conducing page replacement algorithm. this method should return ***FAILURE*** if the page is valid (it doesn't need to call page fault) or if the thread associated with page is killed. In addition, the method return ***NotEnoughMemory*** if all frames locked or reserved. The ValidatingThread of the page must set to the *thread* that input to the function, and set it to null before leaving the method. Also, before handling the page fault, the thread must be suspended with the help of SystemEvent and resumed before leaving the method. No page replacement algorithm should be invoked unless there is no free frame. If there is free frame, one frame will be obtained and reserved, then swap-in operation is performed. Otherwise, the page replacement algorithm invoked to get and reserve frame, if the frame is clean then perform the same procedure as before. If it

	is clean, then swap-out operation is performed, freeing the frame, then swap-in. after each swap-in and swap-out the method should check if the thread was killed, if yes, then notify the thread and dispatch it then return ***FAILURE.*** Finally, before leaving update the page and frame table, Un-reserved the page, notify and dispatch the thread, and return ***SUCCESS.***

Input	*Output*	*Precondition*
ThreadCB thread, **int** referenceType, **PageTableEntry** page	Int (***SUCCESS, FAILURE or NotEnoughMemory)***	none

RUNNING *OSP* SIMULATOR (ESC)

The simulator was running with three input parameter files. The first one with 15, the second with 30, and the third with 64. Where the maximum number allowed is 64.

Parameter 1: *Number of frames=15*

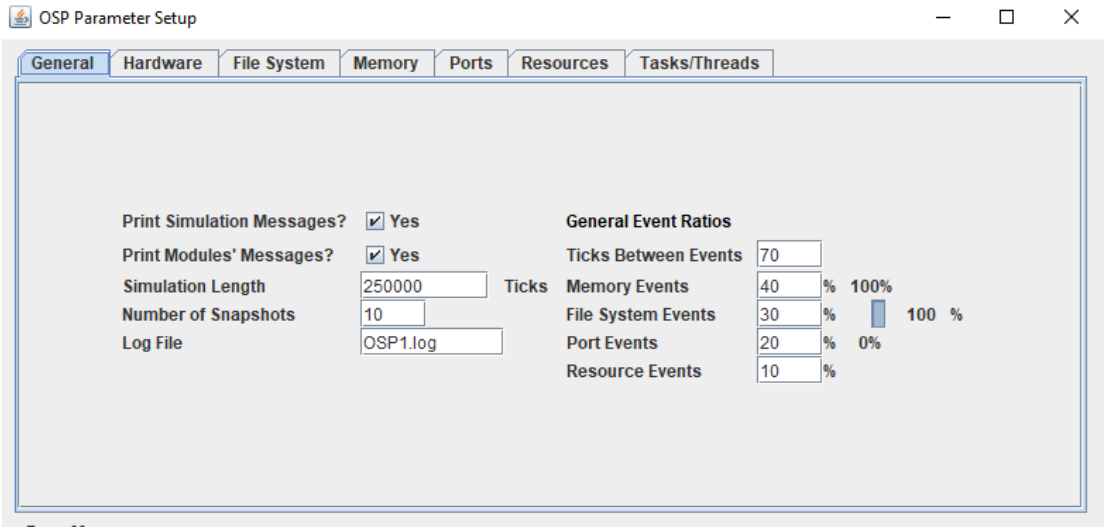

Figure 2 - parameter setup for parameter file 1

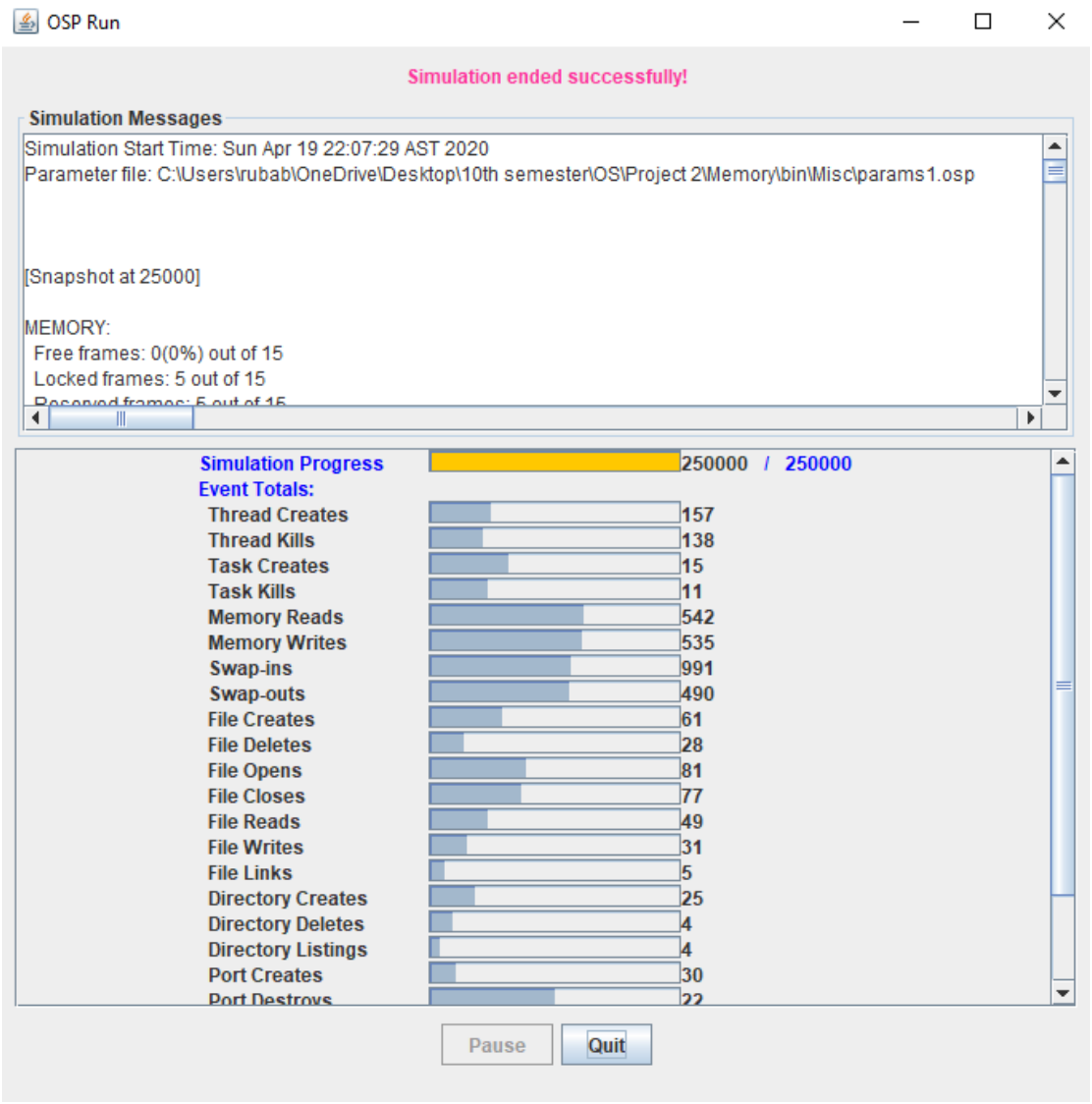

Figure 3 - simulation message for parameter file 1

CPU Utilization: **92.256%**

MEMORY:

Free frames: 13(86%) out of 15
Locked frames: 1 out of 15
Reserved frames: 14 out of 15
Referenced frames: 0 out of 15

Parametr 2: *Number of frames= 30*

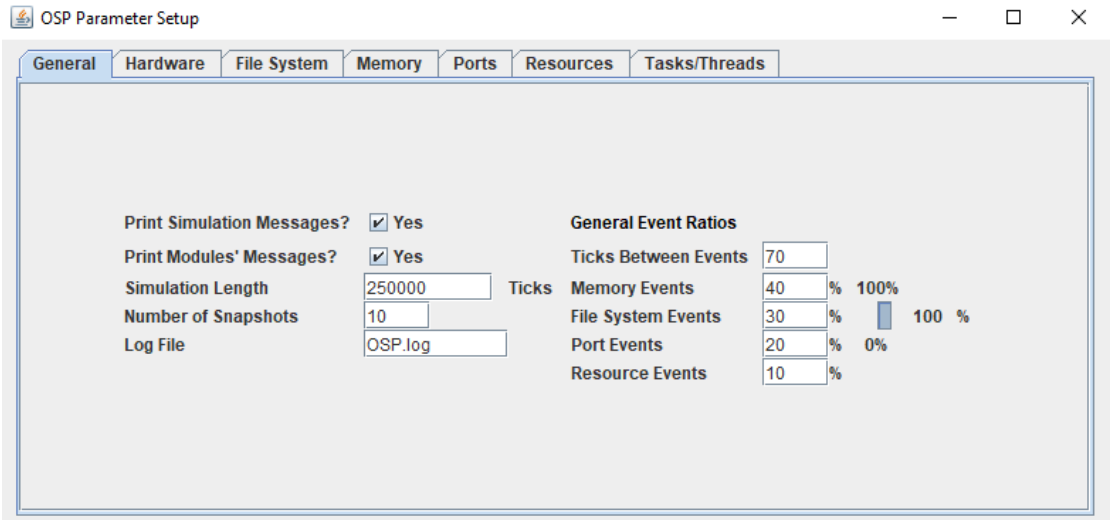

Figure 4 - parameter setup for parameter file 2

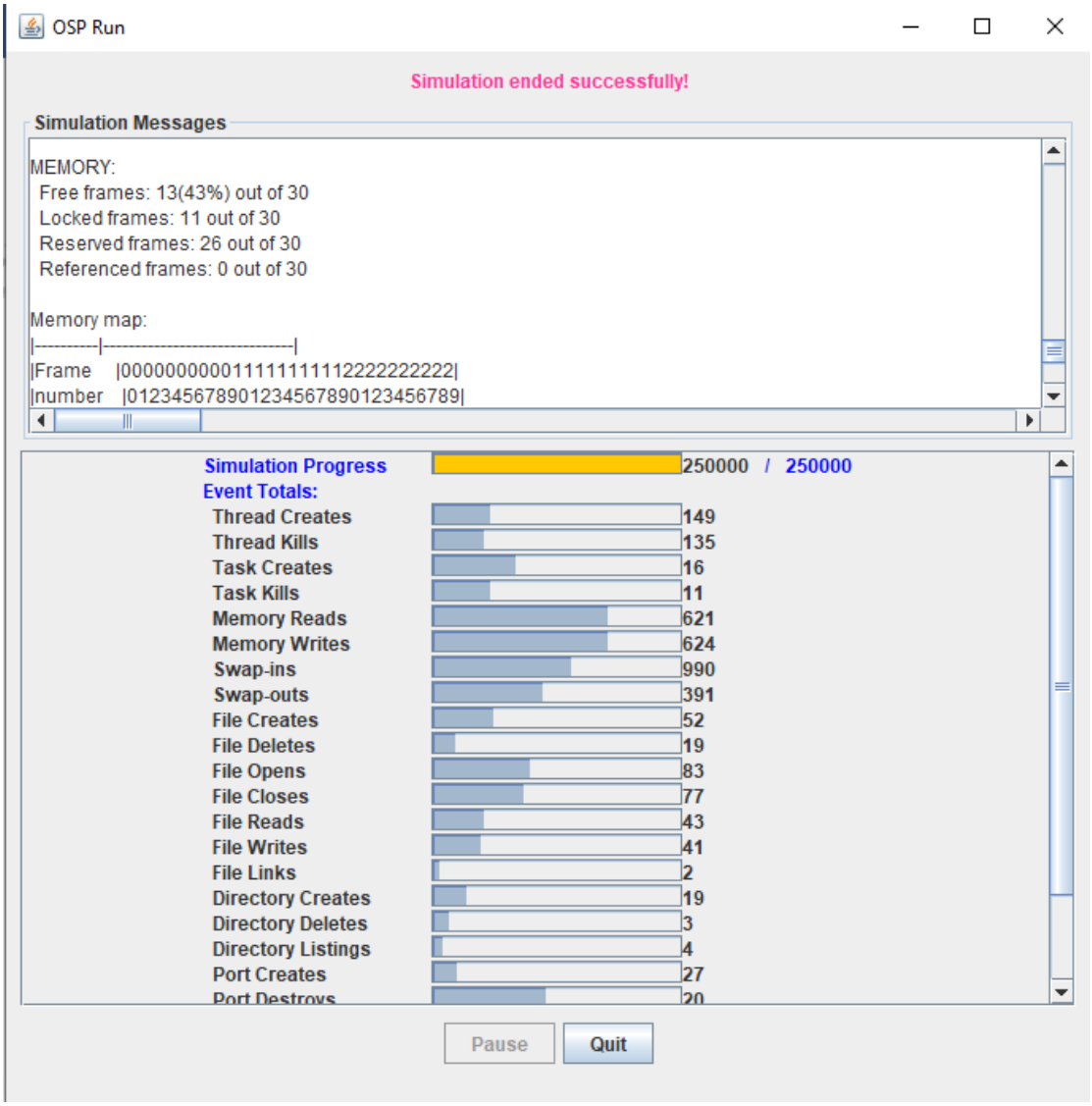

Figure 5 - simulation message for parameter file 2

CPU Utilization: **85.6484%**

MEMORY:

Free frames: 13(43%) out of 30
Locked frames: 11 out of 30
Reserved frames: 26 out of 30
Referenced frames: 0 out of 30

Parameter 3: *Number of frames=64*

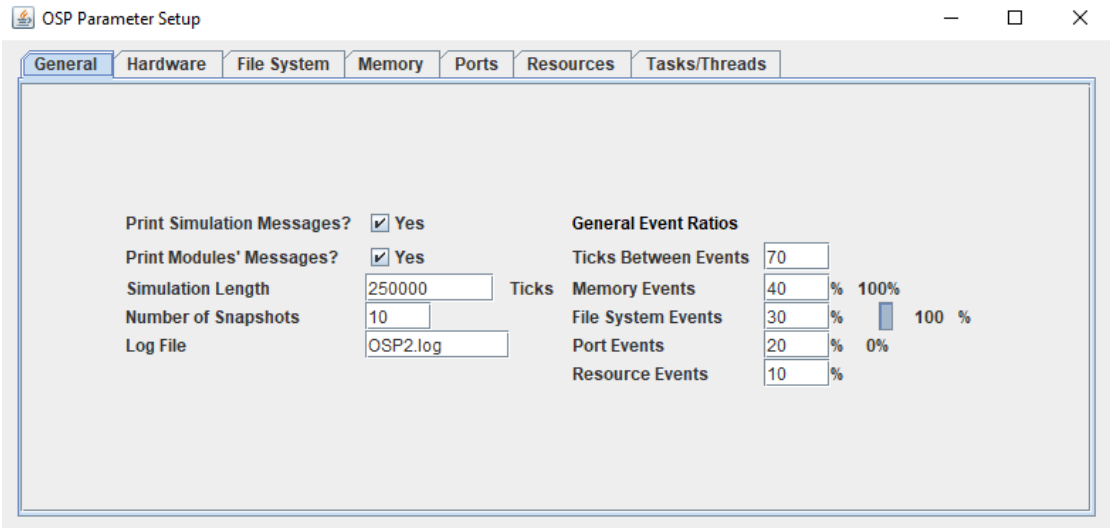

Figure 6 - parameter setup for parameter file 2

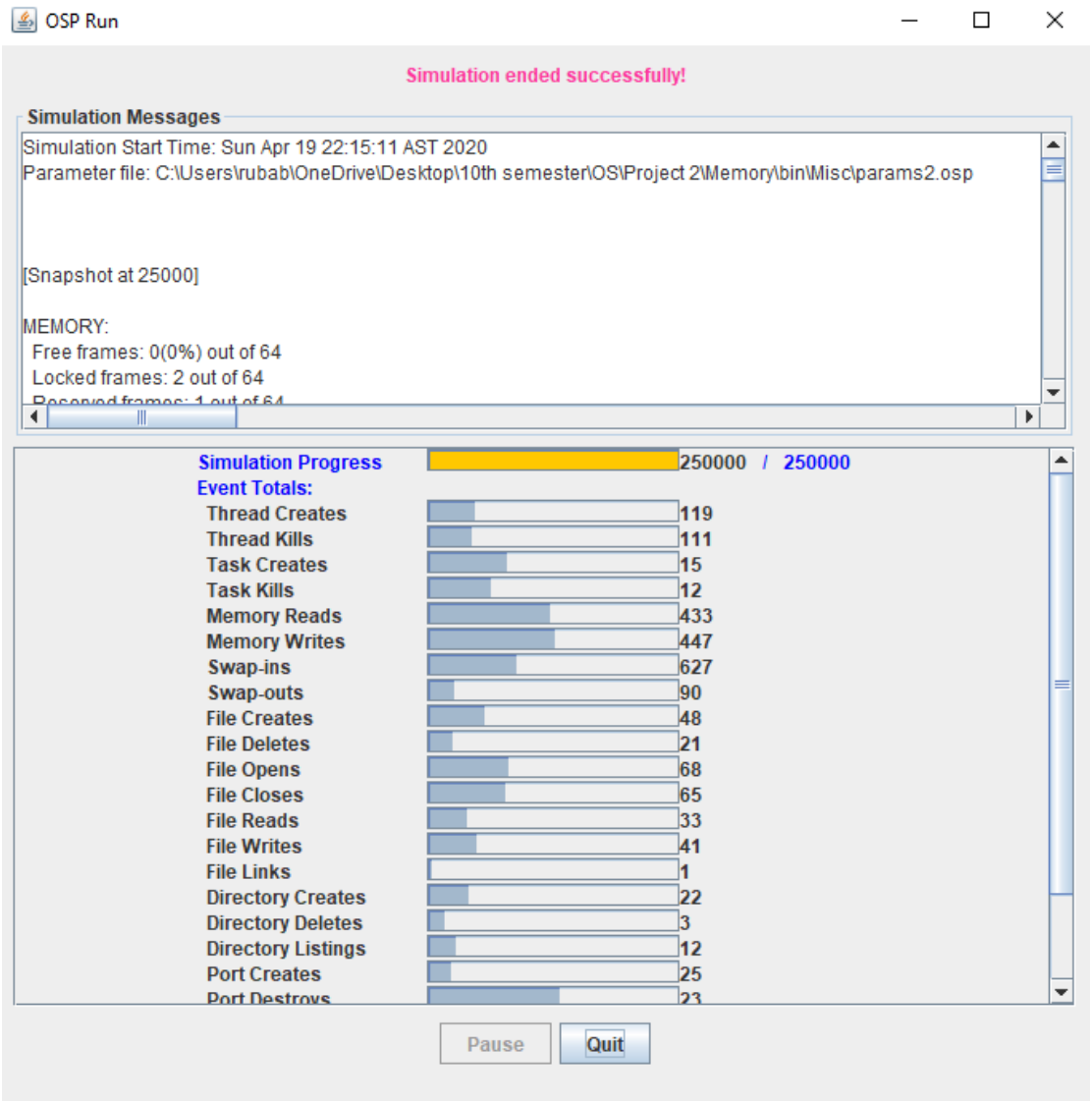

Figure 7 - simulation message for parameter file 3

CPU Utilization: **63.4320%**

MEMORY:

Free frames: 37(57%) out of 64
Locked frames: 1 out of 64
Reserved frames: 3 out of 64
Referenced frames: 6 out of 64

RESULTS

The results were tabulated and discussed for the following measurements: CPU utilization, total throughput, and the page fault rate. The first measure could be found directly in the log file, while the others calculated using the following formulas. The required parameters for the calculation were obtained from the simulation output as shown in Figure 3, 5, and 7 or from the log file.

Total Number of Tasks/Threads

$$total\ number\ threads\ = \text{created threads} - \text{alive threads}$$

Total Throughput of Tasks/Threads

$$Total\ Throughput\ = \frac{total\ number\ threads}{total\ running\ time}$$

Page-Fault Rate

$$Page\ Fault\ Rate = \frac{Swap\ in}{memory\ reads + memory\ writes}$$

Where;

Total memory access = $\text{memory reads} + \text{memory writes}$

Number of Page Fault= Swap in

The following table tabulated the results of the specified measurements, comparing the enhanced second chance page replacement algorithm with the algorithm in the DEMO.
\

Table 2 – Results of ESC Page Replacement and the DEMO

	File 1		File 2		File 3	
Number of main memory frames	15		30		64	
Compared method	DEMO	ESC	DEMO	ESC	DEMO	ESC
CPU utilization	79.9648%	92.256%	64.0532%	85.6484%	73.2832%	63.4320%
total throughput	$4.36*10^{-4}$	$5.52*10^{-4}$	$4.48*10^{-4}$	$5.4*10^{-4}$	$4.72*10^{-4}$	$4.44*10^{-4}$
page fault rate	1.48	0.9201	1.39	0.7951	1.18	0.7125

FINDINGS

As shown from Table 2, for both the Demo and ESC, the page fault rate increased with the decreasing of number of physical memory frames. Therefore, as we increased the number of frames that are available in the memory, the frequency of page faults was low. That is due to the less needing of performing the page replacement method, which is as stated before, it requires only when there wasn't enough free frame to be assigned to the requested page. Therefore, the number of performing swap-out and swap-in operation is less.

Furthermore, there is inversely proportional relationship between the CPU utilization and number of frames. As we increased the number of frames, the CPU utilization decreases. This happens because when the page fault rate is high, the thrashing is high as well. This means, most of the time, the system is running for resolving page faults and busy of multiple operations, thus we can see a decreasing of the CPU utilization. That's clarified the significant differences in utilization. Also, higher throughput means better performance indicating that greater number of processes were completed in a unit of time. In ESC, when the number of frames increases, the throughput decreases. On other hand, in the Demo, the throughput increases.

Now, comparing between the two-page replacement algorithms which are the enhanced second chance and the one used in the Demo; the pre-paging. In can be seen from the Table 2 that ESC is scientifically better than pre-paging, referring to the compared measurements. Although the page fault rate should be between zero and one, where zero means no page replacement and one means all referenced were page fault. The demo was displayed page fault rate greater than one for all parameter files. Thus, ESC has better replacing mechanism where it prefers to replace the clean frames first.

CONCLUSION

In conclusion, the objectives of this project were met. By the end of this project, the implementation of OSP2, including page replacement strategies, was done. Also, the implementation was done successfully for the main class, the MMU class, and other classes like FrameTableEntry, PageFaultHandler, PageTableEntry, and PageTable. All these classes helped in building the final design output of this project.