FACULTY OF ENGINEERING & TECHNOLOGY

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

OPERATING SYSTEMS

ENCS3390

Project Report

Student Names:

Nasser Awwad 1191484

Mazen Batrawi 1190102

Rami Majadbeh 1190611

Instructor: Dr. Ahmad Afaneh

Date: 9/1/2022

Abstract

In this project, we implemented a virtual memory management system using Round Robin scheduler for processes with two page replacement algorithms: FIFO (First In First Out) and LRU (Least Recently Used). Each process, scheduler and page replacement algorithms were simulated in a different thread.

Contents

Theory

Round Robin Scheduler

Is a CPU scheduling algorithm where each process is assigned a fixed time slot which make each process get equal share of the CPU.

FIFO (First In First Out)

One of the page replacement algorithms, that is used for memory management to decide which page to replace when a new page is referenced, FIFO is considered one of the simplest page replacement algorithms, as it will replace the first page that was used with the newest page that comes in, using a queue that will keep track of the pages in memory, with the oldest page first ready to be swapped out

The LRU (Least recently used)

The LRU algorithm replaces the least recently used pages greedily, relying on that the pages that have been used a lot in the last few instructions will probably be used again a lot in the next few instructions, and the pages that aren't used frequently won't be used a lot for a long time.

Simulation

File Generator

In this project, a file generator was used that contains all the information regarding number of processes, size of memory (taken 40 ,200,500 frames as an experiment), minimum frames per process, and information about process containing pid, start, duration, size, and memory traces that are in the memory as stated in the project, the file was generated using C++, it was used and tested in order to create right values and print them to a file in order to copy and paste them into the input file on Python.

Round Robin

In order to schedule the process, a round robin or RR scheduler was created, a round robin is a scheduler that takes a time quantum to run each process which can be entered upon running the program, first of all we initialized the memory.

```
def RoundRobin(algo):
    global Memory, ReadyQueue, threadList, time, Cycles, BlockedQueue,lock
    Memory = [Frame(-1, -1, False) for i in range(MemorySize)]
```

then, each process that arrived to the CPU was sent to the Ready queue to get ready. Each process was in a thread, also the round robin simulation was also in another thread.

```
while not ProceessesFinished():
    lock.acquire()
    ProcessArrival()
    if len(ReadyQueue) == 0:
        time = time + 1
        lock.release()
        continue
    p = ReadyQueue[0]
    if p.is_alive() == False and p not in threadList:
        threadList.append(p)
        p.start()
```

We used the lock to synchronize the threads, we notice here that each round robin quantum was locked.

Then we started to check each memory trace of the pages if they are located in the main memory.

```
for i in range(p.PageLocation, p.PageLocation + quantum):
    if i < len(p.Pages):
        check = CheckMemory(p.Pages[i],p)
```

If yes then it a memory hit and it will goes to the next page and so on until the quantum time ends.

If not, then we have a page fault and we need to get it from the disk and send it to the memory. So we pop the current process from the ready and put it in the blocking queue.

```
if not check:                    time = time + 1
    ReadyQueue.popleft()         Cycles += 5
    BlockedQueue.append(p)       out = True
    lock.release()               break
```

```
if NumOfPages < len(Memory):
    Memory[MemoryIndex] = page
    Memory[MemoryIndex].RefBit = True
    FIFOList.put(MemoryIndex)
    NumOfPages += 1
    MemoryIndex = (MemoryIndex + 1) % MemorySize
```

But the memory could be full, so we will need a page replacement algorithm. In this project we used two algorithms.

First one is the FIFO, so we put the FIFO function in a thread and acquire the lock. In FIFO algorithm, the victim frame is the first frame that gets in the memory. after that the process will exit the blocked queue and will get in the ready queue again.

```
else:
    VictimIndex = FIFOList.get()
    FIFOList.put(VictimIndex)
    page.RefBit = True
    Memory[VictimIndex] = page
process.PageLocation = process.PageLocation + 1
Cycles += DiskCyclesHit
PageFaults += 1
process.NumFaults += 1
ReadyQueue.append(process)
BlockedQueue.popleft()
```

Second algorithm is the LRU. Which will be implemented also in another thread that is synchronized with the other threads. In this algorithm, the least frame used which is the oldest frame in the memory will be removed and the new frame will be added to the end of the memory.

```
else:
    if NumOfPages >= len(Memory):
        Memory.remove(Memory[0])
        page.RefBit = True
        Memory.append(page)
```

Also if the frame is in the memory and entered again then it will be removed from where it is and will be added to the end of the memory.

```
if index != -1:
    Memory[index].RefBit = True
    Cycles += MemoryCyclesHit
    Memory.remove(Memory[0])
    Memory.append(page)
```

Then it will continue like this until all processes are finished. All the data of the processes are saved and the turn-around time and also the waiting time.

Each memory hit equal 1 cycle, each disk hit equal 300 cycles,each context switch equals 5 cycles and each second equals 1000 cycles.

Also we assumed that each memory trace check will take 1 second.

Conclusion

In this project, multiple important operating system concepts we covered and implemented, as for simulating memory, processes, a round robin scheduler, two of the page replacement algorithms, as well as synchronization, for the page replacement we implemented the FIFO and LRU page replacement algorithms and covered the difference in theory and implementation above, in the same way, page faults were detected in both of them, on the other hand, a round robin scheduler was simulated using a time quantum determined by the user, as pages were passed to the algorithms through the scheduler, in the end, Simple synchronization was used in the python program using the threading library, specifically, the acquire() and release() functions to prevent deadlocks between the different threads that contain the processes.