CSCI 460 Operating Systems
Assignment 1

This assignment is on memory management. You are asked to implement a simple paging system.

The main input to the program includes size of the memory (in bytes), frame size/page size (in bytes), and the initial job requests (inputted as a text file). You could assume that the input page size is a factor of the memory size. This would make easier to divide the main memory into set of frames. **But the job size do not have to be divisible by the page size.**

Your program will simulate the snapshot of the hypothetical memory system. Moreover, new job requests should be able to be sent to the simulator through your command line. Following are possible command line commands that you can issue to your simulator.

Start the program:

```
./page_system_simulator memory_size_in_bytes page_size_in_bytes
      initial_job_requests.txt
```

Dynamic job requests:

$ Job_number bytes – a new job and the size of the job. (Note that if the memory already has a job of the same job number it should be rejected).
$ Job_number 0 – remove the job from the memory.
$ JoB_number -1 – suspend the execution of the job manually and swap the pages back to secondary storage.
$ Job_number -2 – resume a suspended job manually. If the job is already suspended, then it should be moved back to the main memory from the secondary storage. You might have to suspend another job to make sure that you have enough space to resume.
$ print – print the status of the memory. You should come up with a way to nicely print this in the terminal. You should print the memory system preferably as a set of chunks of frames and which program page is loaded to each of the frames. If a frame is free, you should indicate that as well. Then for each job/process you should print the page table as well along with other information. If a process is swapped back to secondary storage, that can be indicated in the page table. Note that it should include the internal fragmentation of each process as well.
$ exit – exit the program.

Following is an example initial_job_requests.txt file.

Job_ID	Size
1	35000
2	4096
3	4096
4	4096
5	4096
6	4096
7	4096
8	4096
2	0
4	0
6	0
8	0
9	12000
10	24000
7	-1
3	-1
print	
7	-2
print	

Remember that this is a simple paging system, therefore you need your whole program to be in the memory for it to be executed. Therefore, if the memory does not have enough space you need to move a job/s already in the memory. You could use First-In-First-Out policy. Basically, if the main memory does not have enough free space, then, you could swap the oldest job/s in the memory. Note that you may have to more than one job from the memory. Furthermore, if the input job is too large for the memory, you should reject the job.

If you look at the example initial_job_requests.txt file, and assume that you invoke your program with following parameters:

```
./page_system_simulator 65536 4096 initial_job_requests.txt
```

Basically, there are 16 frames in the main memory as 65536/4096 = 16.

Then according to the initial requests file, we load 1,2,3,4,5,6,7,8 jobs into the main memory. Then we remove/stop executing the processes 2,4,6,8 from the system. Basically, this means that these processes no longer exist in the system. Then, we try to load process 9 and 10 to the memory. After that we suspend process 7 and 3. Then at last we resume the process 7. Now, for this example, if we have used the FIFO replacement policy, when we print the status of the paging system, we should see the following in the main memory:

Frames 1-6 should be used by the process 10, process 7 takes frame 7. Process 9 takes frames 10,12,14. Process 5 takes frame 13. Processes 1 and 3 are suspended and not in the main memory. Process 2,4,6,8 has already finished processing therefore, there will not be any information about them in the system. However, there must be page tables for processes 1,3,5,7,9 and 10. These tables should indicate where each of its pages resides in the main memory, if there are not in the main memory (which means they are swapped to secondary storage), it should be indicated in the page table. Moreover, you should print out the internal fragmentation of each process that are already in the memory when you execute print command.

Run your program for the given input files. For file 1 use 65536 and 4096 as the memory size and the page size. For the file 2 use 20000 bytes and 1000 bytes as the memory size and the page size. You must upload the outputs for each of these inputs.

Due Date: 11:59 PM on Friday, September 20, 2024. You should submit your source code an output as **two separate files** on D2L assignment 1 folder. Preferably in the form of name_-1.c and output-i.txt. You can use C, C++, Java, python, or any reasonable programming language. You can discuss your progress with your peers; however, you should complete your assignment by yourself. You cannot use **generative AI to create the code for you.**