Faculty of Engineering & Technology

Electrical & Computer Engineering Department

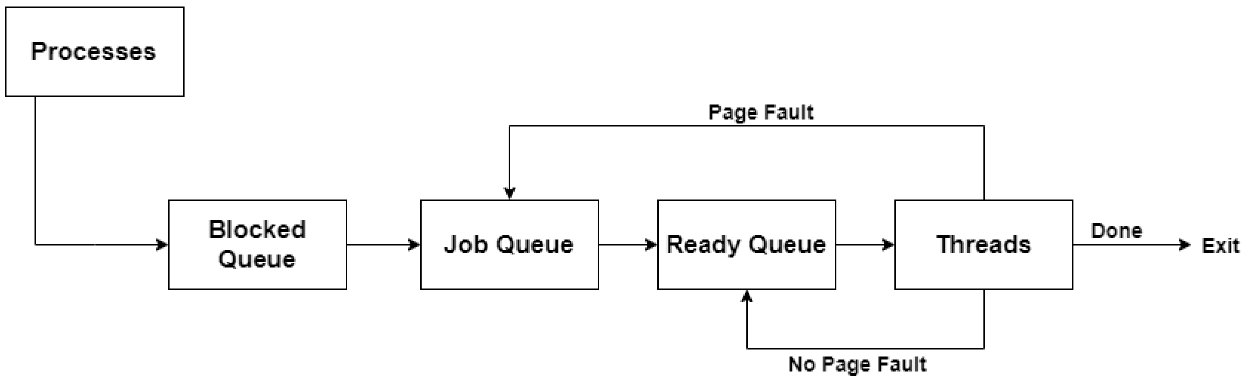
ENCS3390

Memory Manager Simulator

Student Names	Student ID	Section
NoorAldeen Tirhi	1190081	4
Abdallah Daoud Mohammad	1190515	4
Mohammad AbuJaber	1190298	2

Theory

This simulation was written using python3, python 3.10 in particular, it will not run properly on python 2.7.



All processes are read from the configuration file, any process that has its arrival time exceeded is loaded into the blocked queue.

The scheduler then checks if the memory can provide the processes in the blocked queue with minimum required frames, the process is moved into the job queue.

The memory manager starts getting required pages for the processes in the job queue.

The memory manager thread assigns free frames to the top process in the job queue, if it needs more frames.

The memory manager checks if the top process in job queue is assigned enough frames, if this condition is met, the process is moved into the ready queue.

The scheduler will take the process from the ready queue and put it on a thread, if there is one available.

While running on threads, processes might try to access a page that isn't available on memory (page fault), if this happens the process is sent to the job queue, where the memory manager will grab the page it needs and load it into the memory, however if the thread goes through the quantum time without the process having a page fault, then the process returns to the back of the ready queue, also if the process finishes all its page accesses it exits the program, and the memory manager frees its frames

Code

```
class Shared: # to access from memory thread and modify it's value
            memorySizeByFrame = 0 # changed when read file
            availableFrames = 0 # changed when read file
            maxAvailableFrames = 0 # changed when read file
            minFramesPerProcess = 0 # changed when read file
            turnOff = False # to kill threads
            preStatus = '' # the system previous status to ptint the changes only
            memoryChange = False # lock from memory manager thread to have proper
results
            pageFaultCounter = 0 #used to count number of page faults
            Hits = 0 # used to count number of memory hits
```

This a class for all global data, most the data here is read from the file, preStatus is used to report the previous status of the simulator, turnOff is used to kill threads when they finish work(when they get through all memory traces for all processes), the memoryChange variable is used as a lock when threads enter critical section.

```
class Process:
            def __init__(self,pid,start,duration,size,memoryTraces):
                #read from file and fill these variables
                #,pid, start, duration, size, memoryTraces
                self.pid = pid # process id
                self.start = start # arrival time
                self.duration = duration
                self.size = size # process size by frames
                self.memoryTraces = memoryTraces
                self.pageAccesses = []
                self.hasFrames = 0
                self.minNeededFrames = Shared.minFramesPerProcess +
math.ceil(self.size/1024) # ceil, where frames num is integer
                for trace in memoryTraces:
                    self.pageAccesses.append(hex(int(trace,16)>>12))
                self.pageTable = {}
                for page in self.pageAccesses:
                    self.pageTable[page] = [False, 0, 0]
```

Process class, holds all information concerning a certain process, such as PID, start time, duration, size etc.….

The minimum needed frames for a process are calculated by adding the frames needed for a page table with the minimum frames per process.

The frames needed for a page table is calculated by assuming each entry in the page table needs 4 bytes, for pointing at memory frames and have the valid bit, time brought and last use time, the number of bytes needed is 4*(process size), number of frames = 4*(process size)/4096.

The memory traces as read from file point to particular bytes, which are not significant to the simulation, so they are replaced with the pages containing the particular bytes pointed to, this is done by shifting each memory trace 12 bits to the right (as specified by the project).

Page table is represented by a dictionary, each cell is pointed to with the page id, and contains the valid bit, time it was brought to memory and the last time it was accessed.

Threads

```python
def memoryManager(pageReplacementFunction): # in memory manager thread
    while(True):
        if Shared.turnOff: break
        #####################################################
        # free finished Processes frames
        for process in finishedProcessestofreeItsFrames:
            finishedProcessestofreeItsFrames.pop()
            Shared.availableFrames += process.hasFrames
            Shared.maxAvailableFrames += process.minNeededFrames
        #####################################################

        if len(jobQueue) > 0:
            timer = cycles + 300
            while(timer > cycles): Shared.memoryChange = False
            if Shared.availableFrames < 1:
                pageReplacementFunction()
                Shared.availableFrames += 1
            process = jobQueue[-1]
            process.pageTable[process.nextPage()] = [True, cycles,cycles]
            process.hasFrames += 1
            if process.hasFrames >= process.minNeededFrames:
                readyQueue.insert(0, jobQueue.pop())
            Shared.availableFrames -= 1
        Shared.memoryChange = False # release
```

This is the memory manager thread, each time it runs, it checks the turn off variable, to know if it should run or not.

It then checks the finishedProcesstofreeItsFrames list, and frees the frames assigned to processes that are done working.

The thread then checks the job Queue, and brings the needed page of the top processes in the job queue to the memory if there is an available frame, if not, a page replacement algorithm is engaged, page replacement time (I/O wait) is accounted for with the timer variable, finally the top process gets sent to the ready queue after it acquired the minimum needed frames.

The memory lock is then released, it is firstly acquired in the scheduler

```
def runThread(threadNumber): # runProcessOnThread
     while(True):
         if Shared.turnOff: break
         process = processOnThread[threadNumber]
         if process != True:
             for i in range(quantum):
                 nextPage = process.nextPage()
                 if nextPage != '':
                     if process.isInMainMemory(nextPage):
                         process.execute(cycles+i)
                         if i+1 == quantum: readyQueue.insert(0, process)
                     else:
                         jobQueue.insert(0, process
                         Shared.pageFaultCounter += 1
                         break
                 else:
                     finishedProcesses.append(process)
                     finishedProcessestofreeItsFrames.insert(0, process)
                     break
             processOnThread[threadNumber] = True
```

this is the function that is used when running a process, multiple processes will run on multiple threads at a time, a particular thread will run only one process and start going through the page accesses of the process, if a page fault occurs the process is sent to the back of the job queue.

```
while(True):  # this loop presents scheduler thread 'Round Robin' scheduler
            arrivalProcesses()
            l = len(blockedQueue)
            for i in range(l):
                minNeededFrames = blockedQueue[-1].minNeededFrames
                if Shared.maxAvailableFrames >= minNeededFrames:
                    jobQueue.insert(0, blockedQueue.pop())
                    Shared.maxAvailableFrames -= minNeededFrames   # *1

            Shared.memoryChange = True   # lock
            while Shared.memoryChange:
                pass
            trackAll()
            if (len(newProcesses) + len(blockedQueue) + len(jobQueue) +
len(readyQueue)) == 0:
                Shared.turnOff = True
                break
            l = len(readyQueue
            if l > threadsNumber-2:
                l = threadsNumber-2
            for i in range(l):
                processOnThread.insert(i, readyQueue.pop())
            while(True):
                finished = True
                for finished in processOnThread:
                    if(finished != True):
                        finished = False
                        break
                if finished:
                    break
            cycles += quantum + 5
```

This is the scheduler running on thread 0, it first checks the blocked queue and attempts to move its processes out of it into the job queue, it then allows the memory manager thread to work and waits for it to finish using the memoryChange Boolean as a lock, after that it checks to all the process queue, if they are all empty, that means the simulation is done, if not it start assigning processes in the read queue to the available thread, and waits for all of them to stop working for whatever reason.

Page Replacement Algorithms

```
def FIFO(): #Completely replaced
            minTime = math.inf
            queue = [jobQueue[-1]]
            if Shared.maxAvailableFrames > 0:
                queue = readyQueue + jobQueue
            for process in queue:
                if (process.hasFrames >process.minNeededFrames)or(len(queue)<2):
                    table = process.pageTable
                    for page in table:
                        if table[page][0] and table[page][1] < minTime:
                            firstPage = page
                            minTime = table[page][1]
                            firstPageProcess = process
            firstPageProcess.pageTable[firstPage] = [False, 0,0]
```

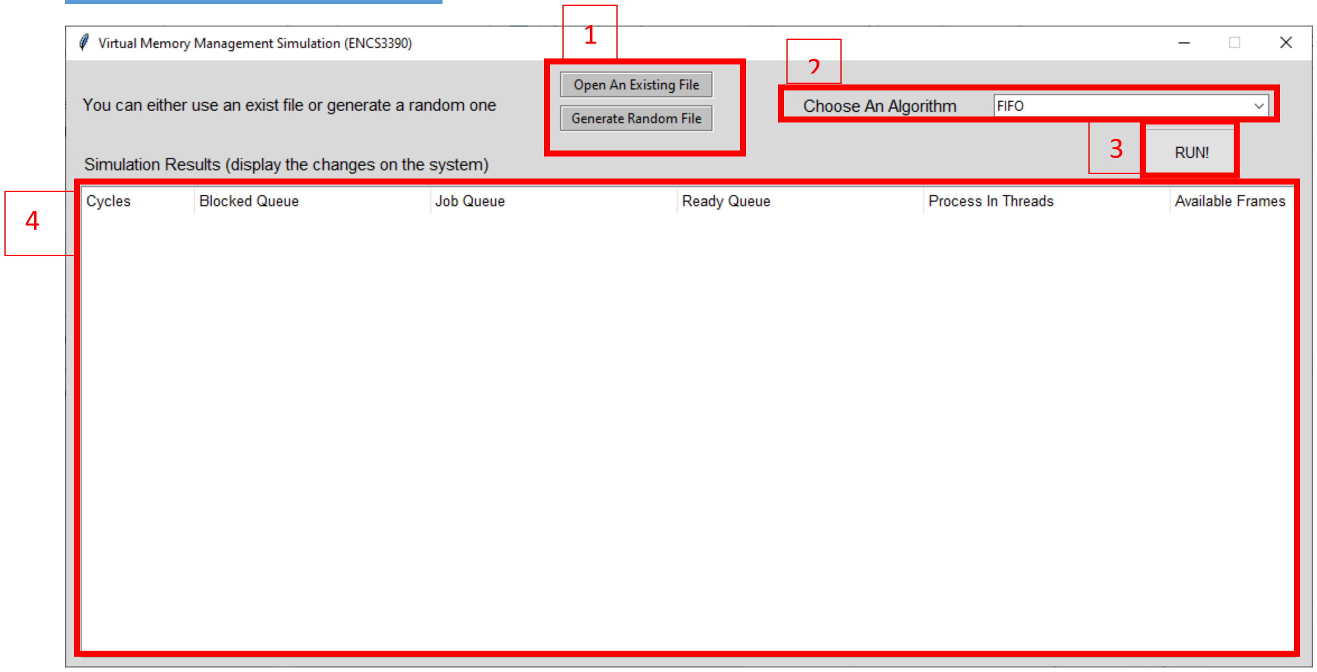
look into all processes with more frames than the minimum, compare all their pages, then remove the page from with the least brought time.

LRU is the exact same process, except remove the page that was least recently used.

```
def secondchance():
            evictPage = ''
            ePageProcess = ''
            queue = [jobQueue[-1]]
            if Shared.maxAvailableFrames > 0
                queue = readyQueue + jobQueue
            while True:
                for process in queue:
                    if (process.hasFrames > process.minNeededFrames) or
(len(queue) < 2):
                        table = process.pageTable
                        for page in table:
                            if table[page][0] and table[page][2] == 0:
                                evictPage=page
                                ePageProcess=process
                                break
                            elif table[page][0]:
                                table[page][2] = 0
                        if evictPage != '':
                            break
                if evictPage != '':
                    break
            ePageProcess.pageTable[evictPage] = [False, 0, 0]
```

look into all frames with more frames than the minimum, go through their pages, every page it goes through that has last use time! = 0 will have its last use time set to 0, and if it finds one with last time use =0, it will evict it and exit, this will keep looping until it evicts a page.

Interface and results

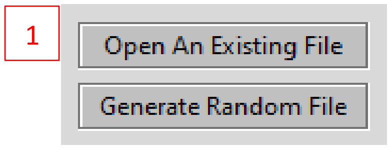


You can either use an exist file or generate a random one

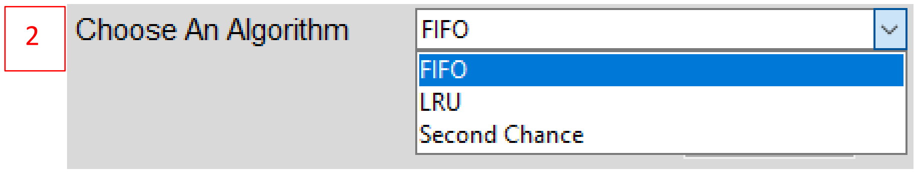
Simulation Results (display the changes on the system)

When running the code, you will get an interface showing your options, this program allows you to choose the configuration file you want to run the code on, or you can generate a random file, it also allows the user to choose the page replacement algorithm, between FIFO, LRU, and Second Chance.

Before running the simulation, the user must choose a configuration file, by default the program will run a file name “generated__config.txt”, if no file is available the user can simply generate a random file.

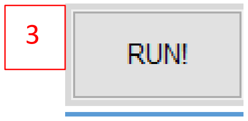


After choosing the configuration file, the user chooses the page replacement algorithm.

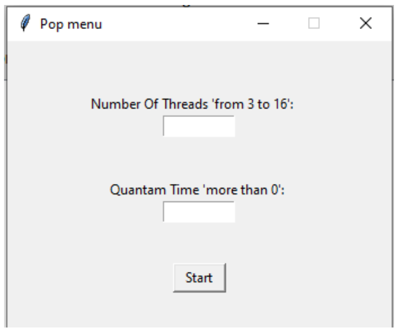


Choose An Algorithm: FIFO

After that the user can simple click the big “RUN!” button to start the simulation.



After clicking run, a prompt will show up asking the user for the number of threads to be run and quantum time of the scheduler, the threads include the threads used for the scheduler and memory manager

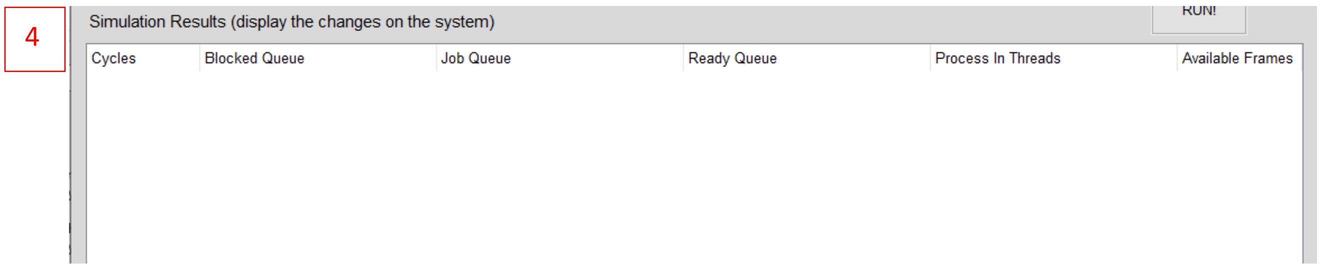


Pop menu

Number Of Threads 'from 3 to 16':

Quantam Time 'more than 0':

The program also provides the user with a number of text columns that show any change happening to the process queues, or the number of occupied memory spaces.

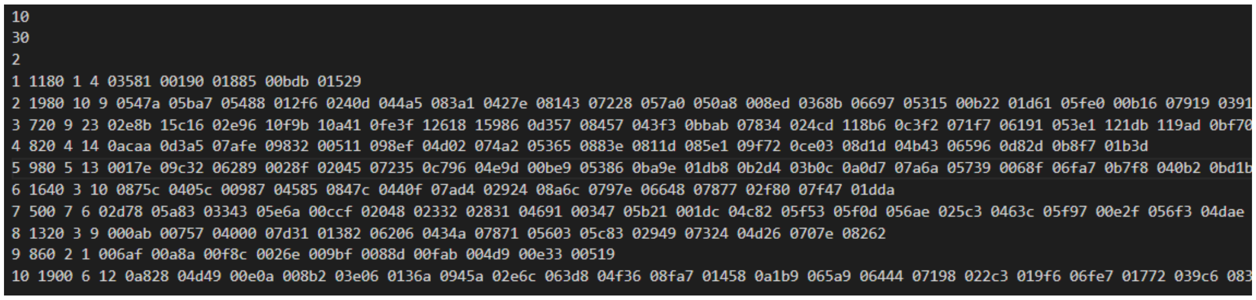


Simulation Results (display the changes on the system)

Cycles	Blocked Queue	Job Queue	Ready Queue	Process In Threads	Available Frames

Test Results:-

This section of the report will show a test result of the file on a randomly generated file, the random file's contents are:



```
10
30
2
1 1180 1 4 03581 00190 01885 00bdb 01529
2 1980 10 9 0547a 05ba7 05488 012f6 0240d 044a5 083a1 0427e 08143 07228 057a0 050a8 008ed 0368b 06697 05315 00b22 01d61 05fe0 00b16 07919 0391
3 720 9 23 02e8b 15c16 02e96 10f9b 10a41 0fe3f 12618 15986 0d357 08457 043f3 0bbab 07834 024cd 118b6 0c3f2 071f7 06191 053e1 121db 119ad 0bf70
4 820 4 14 0acaa 0d3a5 07afe 09832 00511 098ef 04d02 074a2 05365 0883e 0811d 085e1 09f72 0ce03 08d1d 04b43 06596 0d82d 0b8f7 01b3d
5 980 5 13 0017e 09c32 06289 0028f 02045 07235 0c796 04e9d 00be9 05386 0ba9e 01db8 0b2d4 03b0c 0a0d7 07a6a 05739 0068f 06fa7 0b7f8 040b2 0bd1b
6 1640 3 10 0875c 0405c 00987 04585 0847c 0440f 07ad4 02924 08a6c 0797e 06648 07877 02f80 07f47 01dda
7 500 7 6 02d78 05a83 03343 05e6a 00ccf 02048 02332 02831 04691 00347 05b21 001dc 04c82 05f53 05f0d 056ae 025c3 0463c 05f97 00e2f 056f3 04dae
8 1320 3 9 000ab 00757 04000 07d31 01382 06206 0434a 07871 05603 05c83 02949 07324 04d26 0707e 08262
9 860 2 1 006af 00a8a 00f8c 0026e 009bf 0088d 00fab 004d9 00e33 00519
10 1900 6 12 0a828 04d49 00e0a 008b2 03e06 0136a 0945a 02e6c 063d8 04f36 08fa7 01458 0a1b9 065a9 06444 07198 022c3 019f6 06fe7 01772 039c6 083
```

the memory traces extend further, but there's no need to show them.

The parameters this file gives is that it has 10 processes, on RAM comprised of 30 frames, with a minimum space per process of 2 Frames, and the actual processes to be run tested on.

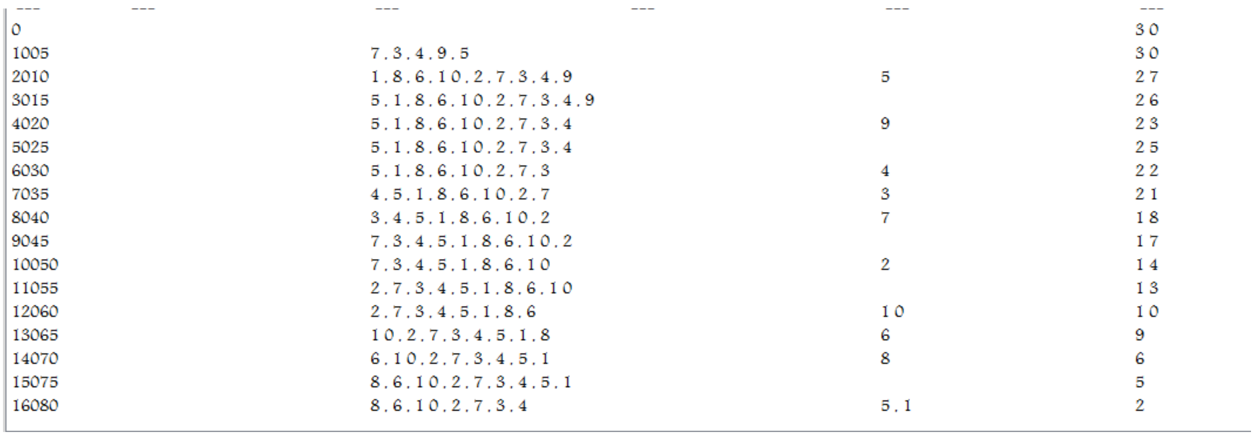
The results here are what happens when you run this file with 8 threads and a quantum time of 1000 cycles, this exact same simulation was run 3 times each time with a different page replacement algorithm.

These columns will print out any change happening to the process queues, this first screenshot is from running the Second Chance page replacement algorithm.

Processes do not linger for long in the blocked queue, in fact they shifted out before the first memory change, meaning they could all be housed in the memory, which initially has capacity of 30 frames.

They do stay for long in the job queue, though where the memory manager will bring their pages to the memory, as you can see the memory runs out of free frames quickly.

Any process that runs into a page fault is visibly sent to the back of the job queue, and you can see that it is the top processes of the job queue that is sent to the ready queue and quickly aftet the threads.



```
 ---        ---                 ---                       ---                  ---                  ---
0                                                                                                   30
1005                    7,3,4,9,5                                                                   30
2010                    1,8,6,10,2,7,3,4,9                                5                         27
3015                    5,1,8,6,10,2,7,3,4,9                                                        26
4020                    5,1,8,6,10,2,7,3,4                                9                         23
5025                    5,1,8,6,10,2,7,3,4                                                          25
6030                    5,1,8,6,10,2,7,3                                  4                         22
7035                    4,5,1,8,6,10,2,7                                  3                         21
8040                    3,4,5,1,8,6,10,2                                  7                         18
9045                    7,3,4,5,1,8,6,10,2                                                          17
10050                   7,3,4,5,1,8,6,10                                  2                         14
11055                   2,7,3,4,5,1,8,6,10                                                          13
12060                   2,7,3,4,5,1,8,6                                   10                        10
13065                   10,2,7,3,4,5,1,8                                  6                         9
14070                   6,10,2,7,3,4,5,1                                  8                         6
15075                   8,6,10,2,7,3,4,5,1                                                          5
16080                   8,6,10,2,7,3,4                                    5,1                       2
```

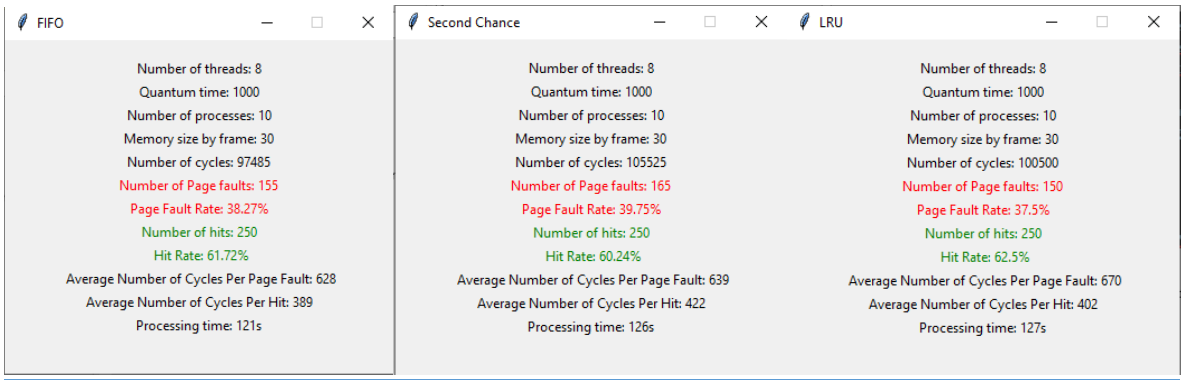
In the middle part all frames are full, so the memory manager keeps evicting pages, although in the middle part, process 1 finished all its work and cleared out 4 frames by leaving, which got quickly occupied again.

26130	5.8.10.6.3.4	2.7.1	0
27135	7.1.2.5.8.10.6.3	4	0
28140	4.7.1.2.5.8	10.6.3	0
29145	10.6.3.4.7.1.2.5	8	0
30150	8.10.6.3.4.7	1.2.5	0
31155	5.2.8.10.6.3.4	7	4
32160	7.5.2.8.10	6.3.4	1
33165	3.4.6.7.5.2.8	10	0
34170	10.3.4.6.7	5.2.8	0
35175	5.8.2.10.3.4.6	7	0

At the end of the simulation all processes pages are cleared from the memory and it returns to the original capacity of 30 frames.

97485	3		17
98490		3	16
99495	3		16
100500		3	15
101505		3	14
102510	3		14
103515		3	13
104520		3	12
105525			30

After every simulation a box shows up with the simulation results, such as page fault count and hit rate.


FIFO

Number of threads: 8
Quantum time: 1000
Number of processes: 10
Memory size by frame: 30
Number of cycles: 97485
Number of Page faults: 155
Page Fault Rate: 38.27%
Number of hits: 250
Hit Rate: 61.72%
Average Number of Cycles Per Page Fault: 628
Average Number of Cycles Per Hit: 389
Processing time: 121s

Second Chance

Number of threads: 8
Quantum time: 1000
Number of processes: 10
Memory size by frame: 30
Number of cycles: 105525
Number of Page faults: 165
Page Fault Rate: 39.75%
Number of hits: 250
Hit Rate: 60.24%
Average Number of Cycles Per Page Fault: 639
Average Number of Cycles Per Hit: 422
Processing time: 126s

LRU

Number of threads: 8
Quantum time: 1000
Number of processes: 10
Memory size by frame: 30
Number of cycles: 100500
Number of Page faults: 150
Page Fault Rate: 37.5%
Number of hits: 250
Hit Rate: 62.5%
Average Number of Cycles Per Page Fault: 670
Average Number of Cycles Per Hit: 402
Processing time: 127s

End