Name: Imogen Hay URN: 6579619 Username: ih00264

COM1032 Operating Systems

Coursework Assignment

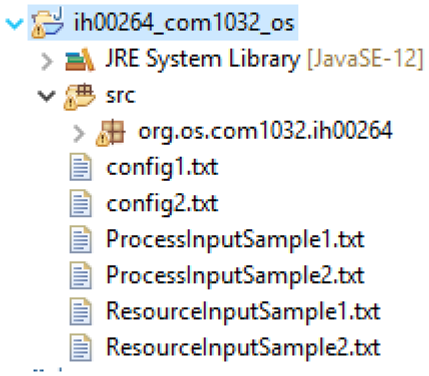
20/05/2020

With my signature I confirm that I am the sole author of the written work here enclosed, that I have compiled it in my own words and mentioned all persons who were significant facilitators of the work.

I.hay

1 Code Setup

To set up my code I recommend using an eclipse IDE and opening the zip folder I have submitted as an archive. This should create a Java project with the 'JRE System Library', 6 text files and a source folder containing a package holding 18 java files. Make sure the input files are in the project but not in the source folder, so the directory used automatically by the program is correct. In eclipse it should look like this with the java files in the 'org.os.com1032.ih00264' package:



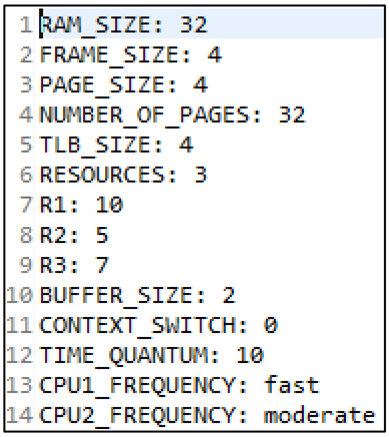
```
ih00264_com1032_os
  JRE System Library [JavaSE-12]
  src
    org.os.com1032.ih00264
  config1.txt
  config2.txt
  ProcessInputSample1.txt
  ProcessInputSample2.txt
  ResourceInputSample1.txt
  ResourceInputSample2.txt
```

In eclipse make sure you are using a Java Development Kit (e.g. jdk-14.0.1) this will allow the project to be automatically compiled and run by pressing Ctrl+F11. Running 'MainBooting.java' should boot the OS simulation and output 'OS OPTIONS' in the console and prompt further user input which I will explain in the user manual.

2 User Manual

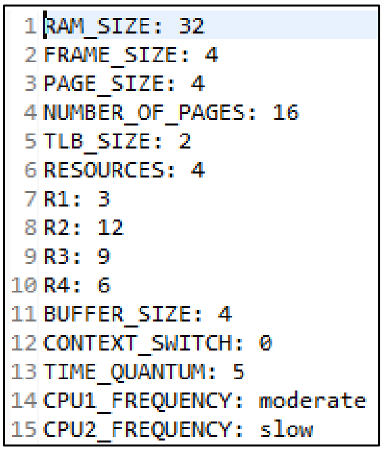
Booting OS:

Once you have started running the program you will be prompted to enter a configuration file. There are currently two options 'config1.txt' or 'config2.txt'. Each is a test case containing information on the hardware of the OS. For the process scheduler each file simulates an OS with different processor speeds, context switch and time quantum. For the memory management unit each file has different amounts of physical and virtual memory set to show my design can work with different values. Each configuration also has different amounts or resources so can be used as test cases for the deadlock prevention system.



```
RAM_SIZE: 32
FRAME_SIZE: 4
PAGE_SIZE: 4
NUMBER_OF_PAGES: 32
TLB_SIZE: 4
RESOURCES: 3
R1: 10
R2: 5
R3: 7
BUFFER_SIZE: 2
CONTEXT_SWITCH: 0
TIME_QUANTUM: 10
CPU1_FREQUENCY: fast
CPU2_FREQUENCY: moderate
```

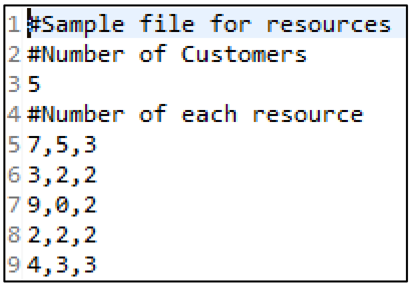
config1.txt



```
RAM_SIZE: 32
FRAME_SIZE: 4
PAGE_SIZE: 4
NUMBER_OF_PAGES: 16
TLB_SIZE: 2
RESOURCES: 4
R1: 3
R2: 12
R3: 9
R4: 6
BUFFER_SIZE: 4
CONTEXT_SWITCH: 0
TIME_QUANTUM: 5
CPU1_FREQUENCY: moderate
CPU2_FREQUENCY: slow
```

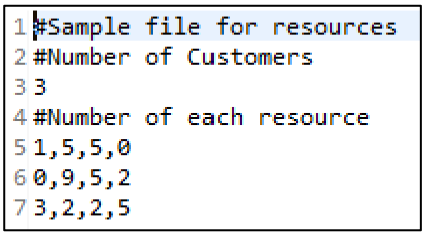
config2.txt

Once you choose one of these files the ProcessInputSample and ResourceInputSample that works with this configuration will automatically be selected. There are currently two resource input samples which serve as test cases for my deadlock prevention system. Each have a different number of customers and resources to highlight that my design will work with any number of either.



```
#Sample file for resources
#Number of Customers
5
#Number of each resource
7,5,3
3,2,2
9,0,2
2,2,2
4,3,3
```

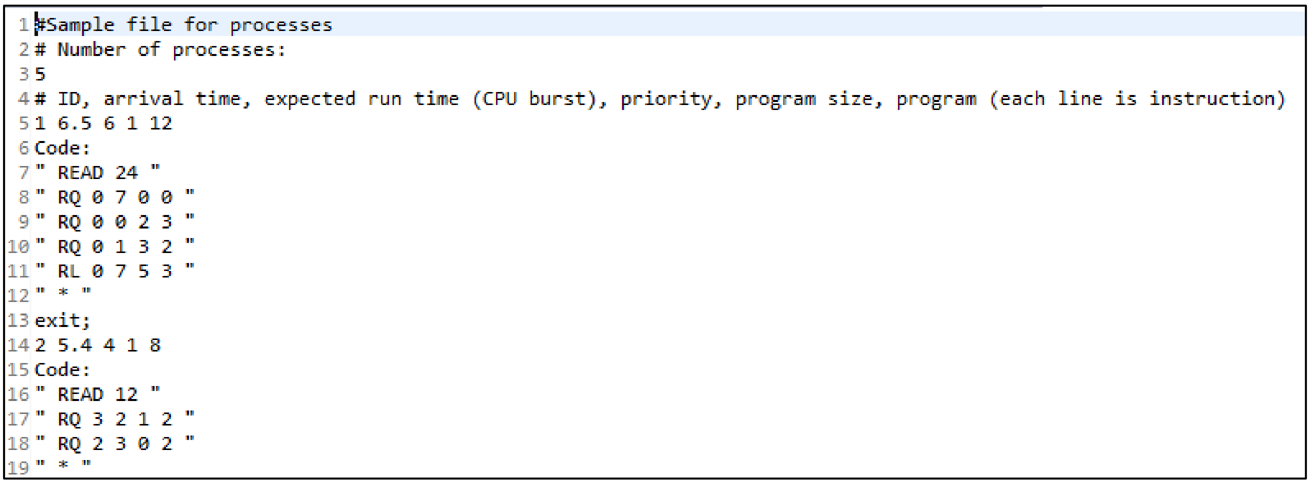
ResourceInputSample1.txt



```
#Sample file for resources
#Number of Customers
3
#Number of each resource
1,5,5,0
0,9,5,2
3,2,2,5
```

ResourceInputSample2.txt

There are also two process input samples which are used mainly as two testcases for the process scheduler, but each process has a program that tests some parts of memory management and my deadlock prevention system. Here is an example section from ProcessInputSample1.txt:



```
#Sample file for processes
# Number of processes:
5
# ID, arrival time, expected run time (CPU burst), priority, program size, program (each line is instruction)
1 6.5 6 1 12
Code:
" READ 24 "
" RQ 0 7 0 0 "
" RQ 0 0 2 3 "
" RQ 0 1 3 2 "
" RL 0 7 5 3 "
" * "
exit;
2 5.4 4 1 8
Code:
" READ 12 "
" RQ 3 2 1 2 "
" RQ 2 3 0 2 "
" * "
```

You can see the rest of these text files in my project.

Once you have selected a configuration file you will then be prompted to choose a page replacement algorithm for my memory management unit which serves as a virtual memory manager. You can enter either ‘lru’ (lest recently used) or ‘fifo’ (first in first out). You can then choose a process scheduling algorithm by entering a number from 0-6. The algorithms these numbers represent are listed here:

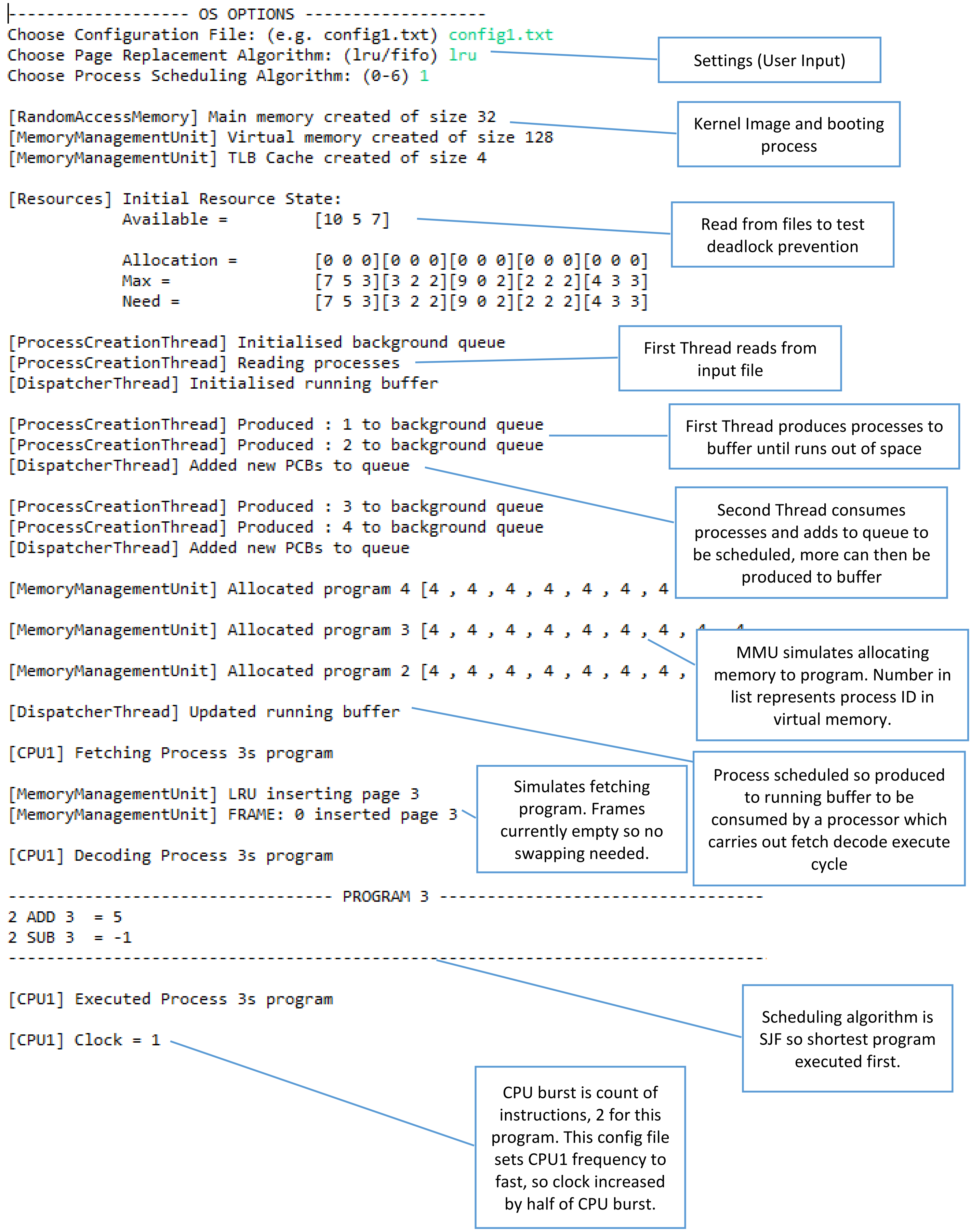
0 = FCFS (first come first serve)
1 = SJF (shortest job first)
2 = RR (round robin)
3 = HPFSN (highest priority first static priority, non-preemptive)
4 = HPFSP (highest priority first static priority preemptive)
5 = HPFD (highest priority first dynamic priority)
6 = WRR (weighted round robin)

After this the kernel image loading will be simulated and the selected config file will be read so data structures can be initialised. The simulation then continues by starting threads which schedule the processes and executes their programs. Once all processes have been processed an event list is printed and the user can enter furthur instructions. I will discuss which instructions are supported by this OS in the architecture section but to end the simulation enter ‘EXIT’. If any errors occur that stop the simulation from running or you would like to select a different configuration terminate the current simulation and run it again.

Functionalities:

To show the functionalities of my OS in action I will start the simulator using each of the configuration files and explain the output. Each file is a test case. Since my process scheduler automatically asks the MMU to allocate and deallocate memory this counts as test case for both subsystems.

For the first test I selected config1.txt which automatically uses the test cases ProcessInputSample1.txt and ResourceInputSample1.txt. I also chose the page replacement algorithm ‘lru’ and process scheduling algorithm 1 (shortest job first).



```
------------------- OS OPTIONS -------------------
Choose Configuration File: (e.g. config1.txt) config1.txt
Choose Page Replacement Algorithm: (lru/fifo) lru
Choose Process Scheduling Algorithm: (0-6) 1

[RandomAccessMemory] Main memory created of size 32
[MemoryManagementUnit] Virtual memory created of size 128
[MemoryManagementUnit] TLB Cache created of size 4

[Resources] Initial Resource State:
            Available =         [10 5 7]

            Allocation =        [0 0 0][0 0 0][0 0 0][0 0 0][0 0 0]
            Max =               [7 5 3][3 2 2][9 0 2][2 2 2][4 3 3]
            Need =              [7 5 3][3 2 2][9 0 2][2 2 2][4 3 3]

[ProcessCreationThread] Initialised background queue
[ProcessCreationThread] Reading processes
[DispatcherThread] Initialised running buffer

[ProcessCreationThread] Produced : 1 to background queue
[ProcessCreationThread] Produced : 2 to background queue
[DispatcherThread] Added new PCBs to queue

[ProcessCreationThread] Produced : 3 to background queue
[ProcessCreationThread] Produced : 4 to background queue
[DispatcherThread] Added new PCBs to queue

[MemoryManagementUnit] Allocated program 4 [4 , 4 , 4 , 4 , 4 , 4 , 4

[MemoryManagementUnit] Allocated program 3 [4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4

[MemoryManagementUnit] Allocated program 2 [4 , 4 , 4 , 4 , 4 , 4 , 4 ,

[DispatcherThread] Updated running buffer

[CPU1] Fetching Process 3s program

[MemoryManagementUnit] LRU inserting page 3
[MemoryManagementUnit] FRAME: 0 inserted page 3

[CPU1] Decoding Process 3s program

---------------------------------- PROGRAM 3 ----------------------------------
2 ADD 3  = 5
2 SUB 3  = -1
--------------------------------------------------------------------------------

[CPU1] Executed Process 3s program

[CPU1] Clock = 1
```

Settings (User Input)

Kernel Image and booting process

Read from files to test deadlock prevention

First Thread reads from input file

First Thread produces processes to buffer until runs out of space

Second Thread consumes processes and adds to queue to be scheduled, more can then be produced to buffer

MMU simulates allocating memory to program. Number in list represents process ID in virtual memory.

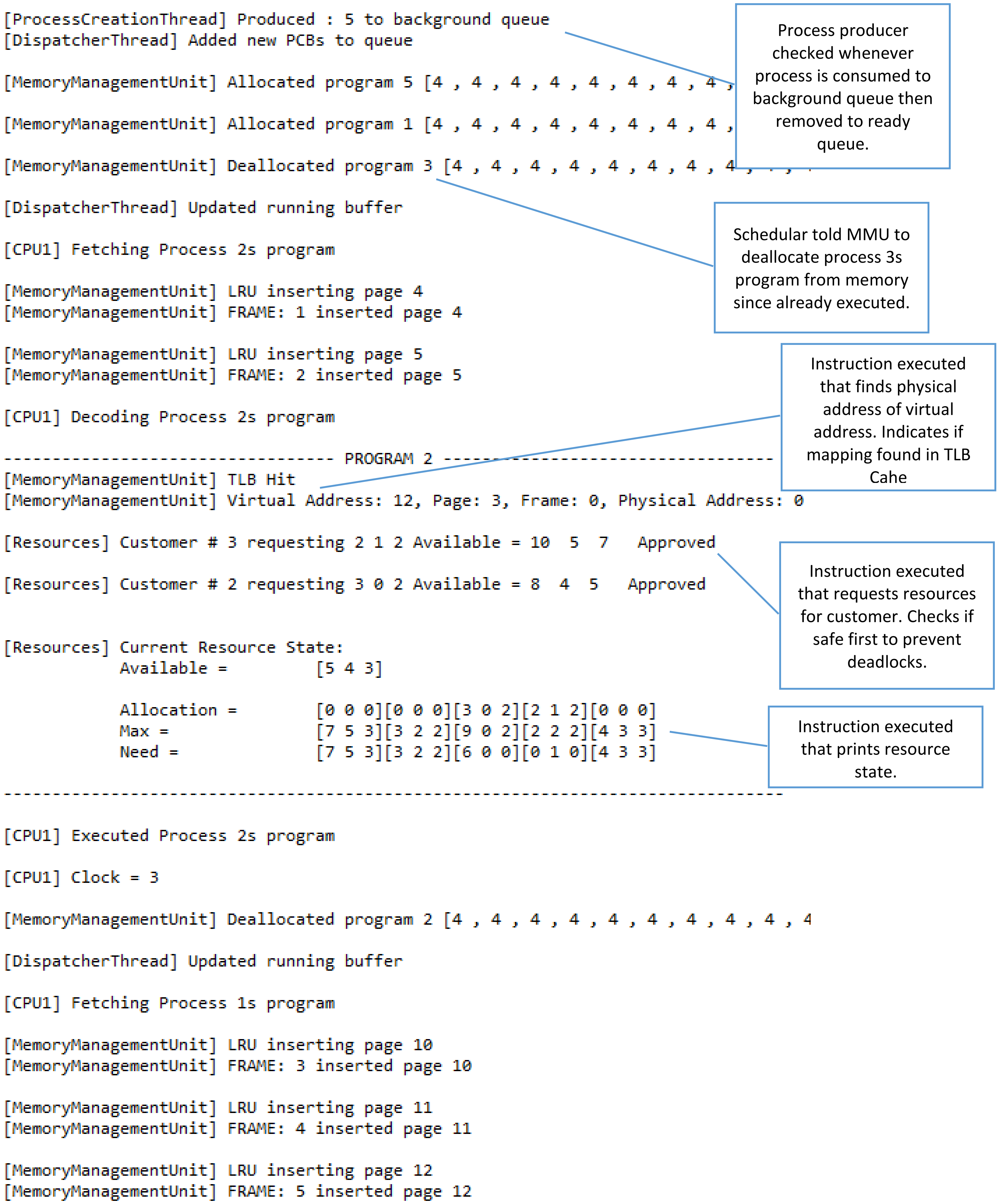
Process scheduled so produced to running buffer to be consumed by a processor which carries out fetch decode execute cycle

Simulates fetching program. Frames currently empty so no swapping needed.

Scheduling algorithm is SJF so shortest program executed first.

CPU burst is count of instructions, 2 for this program. This config file sets CPU1 frequency to fast, so clock increased by half of CPU burst.

Name: Imogen Hay URN: 6579619 Username: ih00264



```
[ProcessCreationThread] Produced : 5 to background queue
[DispatcherThread] Added new PCBs to queue

[MemoryManagementUnit] Allocated program 5 [4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 ,

[MemoryManagementUnit] Allocated program 1 [4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 ,

[MemoryManagementUnit] Deallocated program 3 [4 , 4 , 4 , 4 , 4 , 4 , 4 , 4

[DispatcherThread] Updated running buffer

[CPU1] Fetching Process 2s program

[MemoryManagementUnit] LRU inserting page 4
[MemoryManagementUnit] FRAME: 1 inserted page 4

[MemoryManagementUnit] LRU inserting page 5
[MemoryManagementUnit] FRAME: 2 inserted page 5

[CPU1] Decoding Process 2s program

---------------------------------- PROGRAM 2 ----------------------------------
[MemoryManagementUnit] TLB Hit
[MemoryManagementUnit] Virtual Address: 12, Page: 3, Frame: 0, Physical Address: 0

[Resources] Customer # 3 requesting 2 1 2 Available = 10  5  7   Approved

[Resources] Customer # 2 requesting 3 0 2 Available = 8  4  5   Approved


[Resources] Current Resource State:
            Available =          [5 4 3]

            Allocation =         [0 0 0][0 0 0][3 0 2][2 1 2][0 0 0]
            Max =                [7 5 3][3 2 2][9 0 2][2 2 2][4 3 3]
            Need =               [7 5 3][3 2 2][6 0 0][0 1 0][4 3 3]

--------------------------------------------------------------------------------

[CPU1] Executed Process 2s program

[CPU1] Clock = 3

[MemoryManagementUnit] Deallocated program 2 [4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4

[DispatcherThread] Updated running buffer

[CPU1] Fetching Process 1s program

[MemoryManagementUnit] LRU inserting page 10
[MemoryManagementUnit] FRAME: 3 inserted page 10

[MemoryManagementUnit] LRU inserting page 11
[MemoryManagementUnit] FRAME: 4 inserted page 11

[MemoryManagementUnit] LRU inserting page 12
[MemoryManagementUnit] FRAME: 5 inserted page 12
```

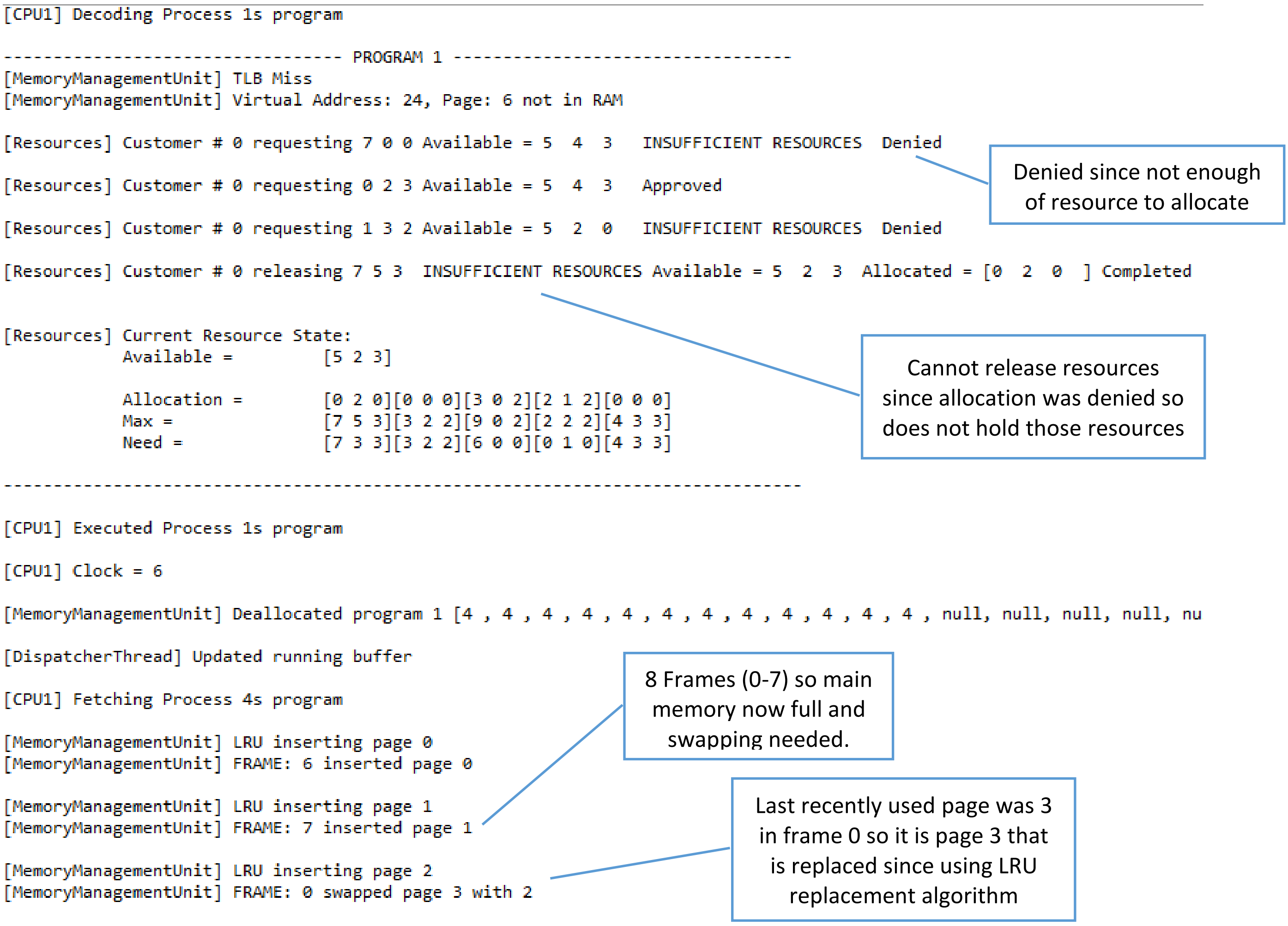
Process producer checked whenever process is consumed to background queue then removed to ready queue.

Schedular told MMU to deallocate process 3s program from memory since already executed.

Instruction executed that finds physical address of virtual address. Indicates if mapping found in TLB Cahe

Instruction executed that requests resources for customer. Checks if safe first to prevent deadlocks.

Instruction executed that prints resource state.



```
[CPU1] Decoding Process 1s program

---------------------------------- PROGRAM 1 ----------------------------------
[MemoryManagementUnit] TLB Miss
[MemoryManagementUnit] Virtual Address: 24, Page: 6 not in RAM

[Resources] Customer # 0 requesting 7 0 0 Available = 5  4  3   INSUFFICIENT RESOURCES  Denied

[Resources] Customer # 0 requesting 0 2 3 Available = 5  4  3   Approved

[Resources] Customer # 0 requesting 1 3 2 Available = 5  2  0   INSUFFICIENT RESOURCES  Denied

[Resources] Customer # 0 releasing 7 5 3  INSUFFICIENT RESOURCES Available = 5  2  3  Allocated = [0  2  0  ] Completed


[Resources] Current Resource State:
            Available =         [5 2 3]

            Allocation =        [0 2 0][0 0 0][3 0 2][2 1 2][0 0 0]
            Max =               [7 5 3][3 2 2][9 0 2][2 2 2][4 3 3]
            Need =              [7 3 3][3 2 2][6 0 0][0 1 0][4 3 3]

--------------------------------------------------------------------------------

[CPU1] Executed Process 1s program

[CPU1] Clock = 6

[MemoryManagementUnit] Deallocated program 1 [4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , 4 , null, null, null, null, nu

[DispatcherThread] Updated running buffer

[CPU1] Fetching Process 4s program

[MemoryManagementUnit] LRU inserting page 0
[MemoryManagementUnit] FRAME: 6 inserted page 0

[MemoryManagementUnit] LRU inserting page 1
[MemoryManagementUnit] FRAME: 7 inserted page 1

[MemoryManagementUnit] LRU inserting page 2
[MemoryManagementUnit] FRAME: 0 swapped page 3 with 2
```

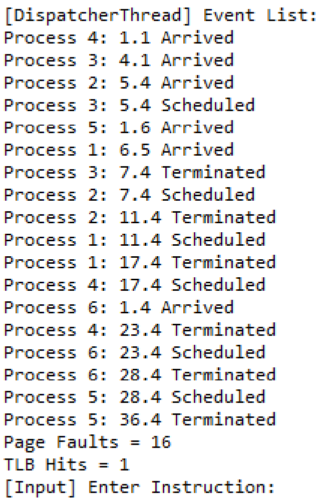
Denied since not enough of resource to allocate

Cannot release resources since allocation was denied so does not hold those resources

8 Frames (0-7) so main memory now full and swapping needed.

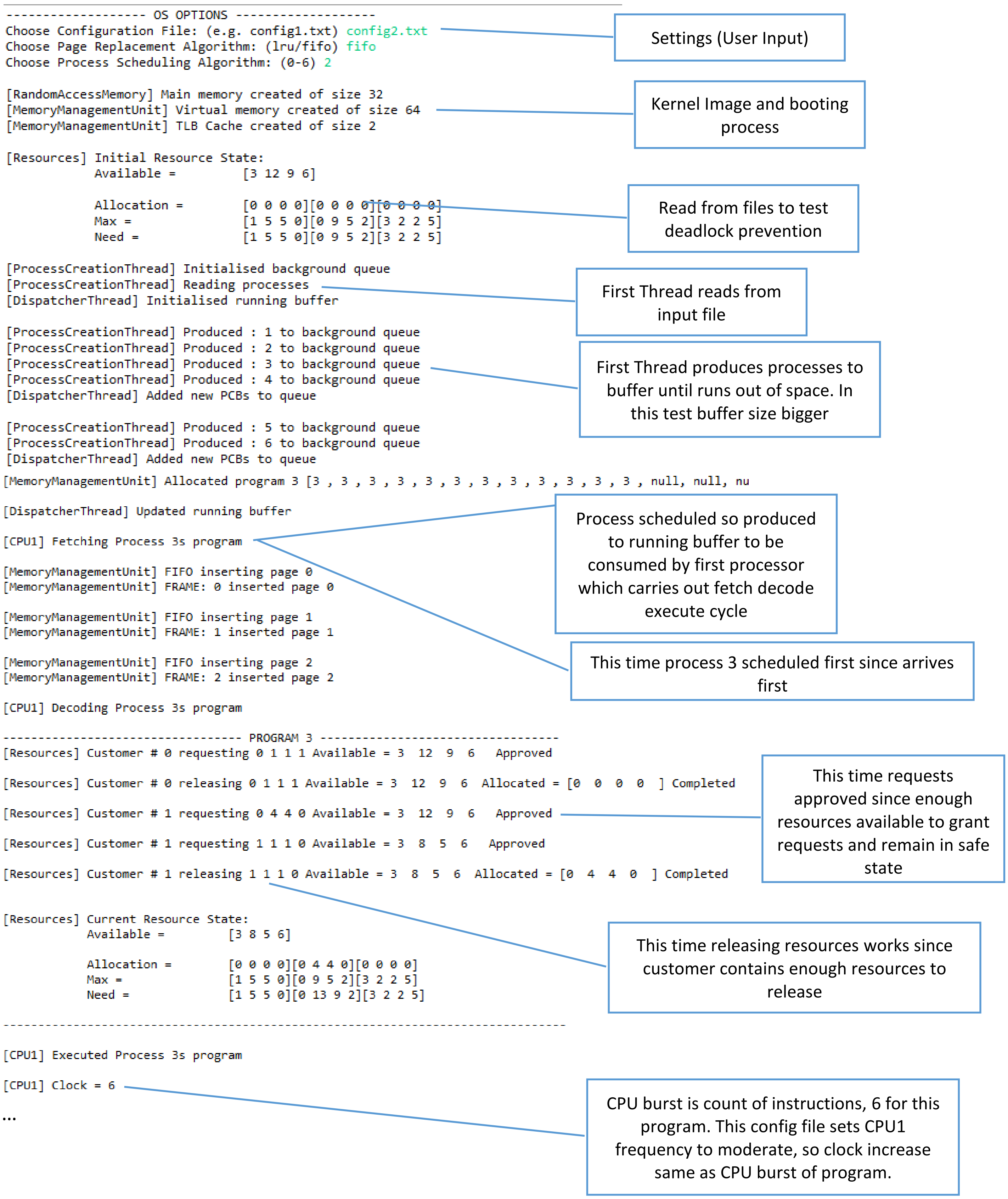
Last recently used page was 3 in frame 0 so it is page 3 that is replaced since using LRU replacement algorithm

I have explained the functionalities that the beginning of the output of this test simulation shows. It shows examples of booting, cache use, buffer use, multithreading, synchronisation, deadlock prevention, memory management, page replacement, multitasking and process scheduling. If you run the same test yourself, you will see the same logic continued and the rest of virtual memory when programs are allocated and deallocated (CPUs used may vary). Since I selected shortest job first you will see the programs get longer from start to finish. This simulation should end with this output of events and then allow you to enter more instructions.



```
[DispatcherThread] Event List:
Process 4: 1.1 Arrived
Process 3: 4.1 Arrived
Process 2: 5.4 Arrived
Process 3: 5.4 Scheduled
Process 5: 1.6 Arrived
Process 1: 6.5 Arrived
Process 3: 7.4 Terminated
Process 2: 7.4 Scheduled
Process 2: 11.4 Terminated
Process 1: 11.4 Scheduled
Process 1: 17.4 Terminated
Process 4: 17.4 Scheduled
Process 6: 1.4 Arrived
Process 4: 23.4 Terminated
Process 6: 23.4 Scheduled
Process 6: 28.4 Terminated
Process 5: 28.4 Scheduled
Process 5: 36.4 Terminated
Page Faults = 16
TLB Hits = 1
[Input] Enter Instruction:
```

For the next test I selected config2.txt which automatically uses the test cases ProcessInputSample2.txt and ResourceInputSample2.txt. I also chose the page replacement algorithm ‘fifo’ and process scheduling algorithm 2 (round robin). I used these algorithms since last time I used ‘lru’ page replacement and SJF a non-pre-emptive algorithm so this time I will demonstrate a pre-emptive one. For this test I will only annotate functionalities that were not featured in the last test.



```
------------------ OS OPTIONS ------------------
Choose Configuration File: (e.g. config1.txt) config2.txt
Choose Page Replacement Algorithm: (lru/fifo) fifo
Choose Process Scheduling Algorithm: (0-6) 2

[RandomAccessMemory] Main memory created of size 32
[MemoryManagementUnit] Virtual memory created of size 64
[MemoryManagementUnit] TLB Cache created of size 2

[Resources] Initial Resource State:
            Available =         [3 12 9 6]

            Allocation =        [0 0 0 0][0 0 0 0][0 0 0 0]
            Max =               [1 5 5 0][0 9 5 2][3 2 2 5]
            Need =              [1 5 5 0][0 9 5 2][3 2 2 5]

[ProcessCreationThread] Initialised background queue
[ProcessCreationThread] Reading processes
[DispatcherThread] Initialised running buffer

[ProcessCreationThread] Produced : 1 to background queue
[ProcessCreationThread] Produced : 2 to background queue
[ProcessCreationThread] Produced : 3 to background queue
[ProcessCreationThread] Produced : 4 to background queue
[DispatcherThread] Added new PCBs to queue

[ProcessCreationThread] Produced : 5 to background queue
[ProcessCreationThread] Produced : 6 to background queue
[DispatcherThread] Added new PCBs to queue
[MemoryManagementUnit] Allocated program 3 [3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , null, null, nu

[DispatcherThread] Updated running buffer

[CPU1] Fetching Process 3s program

[MemoryManagementUnit] FIFO inserting page 0
[MemoryManagementUnit] FRAME: 0 inserted page 0

[MemoryManagementUnit] FIFO inserting page 1
[MemoryManagementUnit] FRAME: 1 inserted page 1

[MemoryManagementUnit] FIFO inserting page 2
[MemoryManagementUnit] FRAME: 2 inserted page 2

[CPU1] Decoding Process 3s program

--------------------------------- PROGRAM 3 ---------------------------------
[Resources] Customer # 0 requesting 0 1 1 1 Available = 3  12  9  6   Approved

[Resources] Customer # 0 releasing 0 1 1 1 Available = 3  12  9  6  Allocated = [0  0  0  0  ] Completed

[Resources] Customer # 1 requesting 0 4 4 0 Available = 3  12  9  6   Approved

[Resources] Customer # 1 requesting 1 1 1 0 Available = 3  8  5  6   Approved

[Resources] Customer # 1 releasing 1 1 1 0 Available = 3  8  5  6  Allocated = [0  4  4  0  ] Completed

[Resources] Current Resource State:
            Available =         [3 8 5 6]

            Allocation =        [0 0 0 0][0 4 4 0][0 0 0 0]
            Max =               [1 5 5 0][0 9 5 2][3 2 2 5]
            Need =              [1 5 5 0][0 13 9 2][3 2 2 5]

-----------------------------------------------------------------------------

[CPU1] Executed Process 3s program

[CPU1] Clock = 6
...
```

Settings (User Input)

Kernel Image and booting process

Read from files to test deadlock prevention

First Thread reads from input file

First Thread produces processes to buffer until runs out of space. In this test buffer size bigger

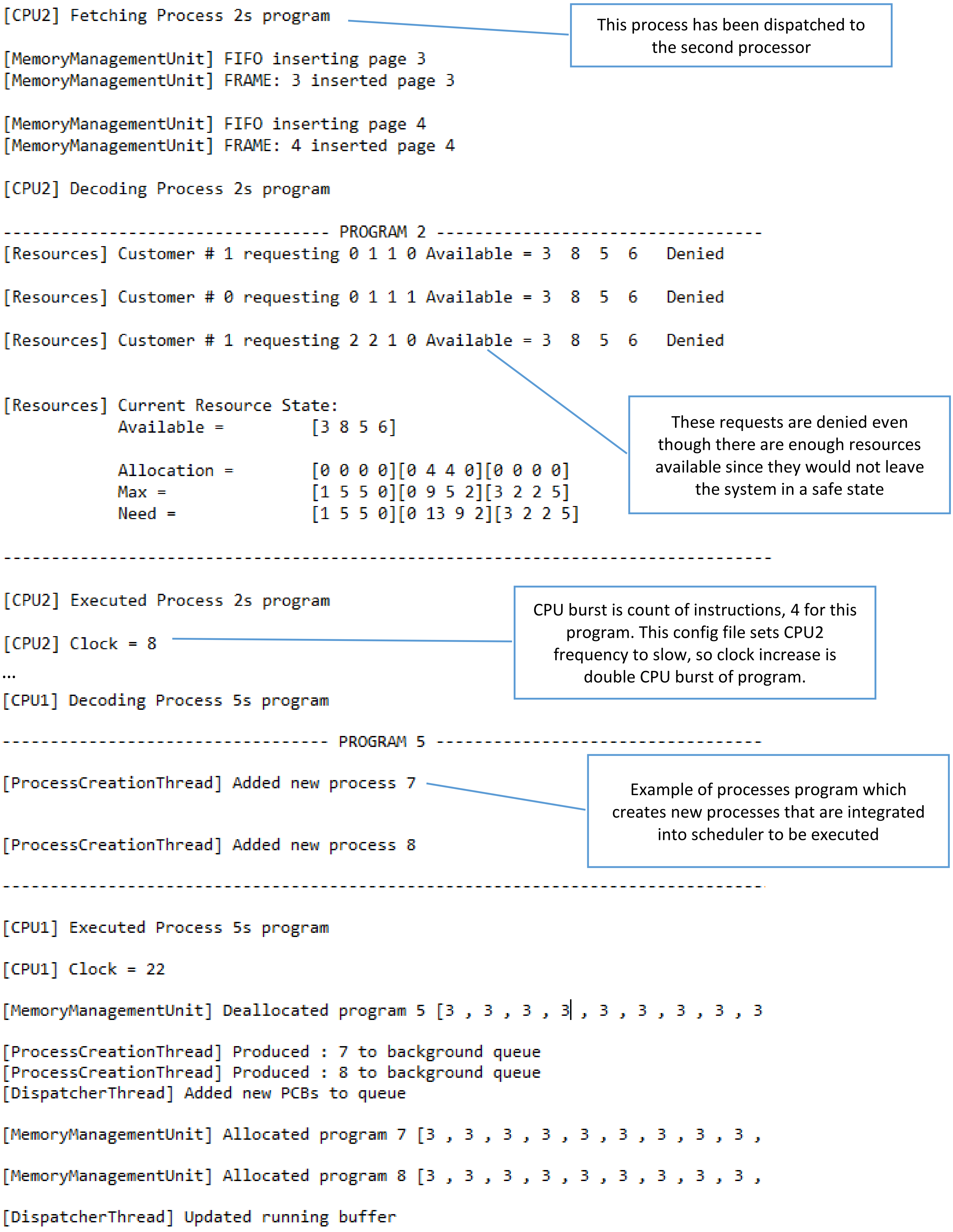
Process scheduled so produced to running buffer to be consumed by first processor which carries out fetch decode execute cycle

This time process 3 scheduled first since arrives first

This time requests approved since enough resources available to grant requests and remain in safe state

This time releasing resources works since customer contains enough resources to release

CPU burst is count of instructions, 6 for this program. This config file sets CPU1 frequency to moderate, so clock increase same as CPU burst of program.



```
[CPU2] Fetching Process 2s program
```

This process has been dispatched to the second processor

```
[MemoryManagementUnit] FIFO inserting page 3
[MemoryManagementUnit] FRAME: 3 inserted page 3

[MemoryManagementUnit] FIFO inserting page 4
[MemoryManagementUnit] FRAME: 4 inserted page 4

[CPU2] Decoding Process 2s program

---------------------------------- PROGRAM 2 ----------------------------------
[Resources] Customer # 1 requesting 0 1 1 0 Available = 3  8  5  6   Denied

[Resources] Customer # 0 requesting 0 1 1 1 Available = 3  8  5  6   Denied

[Resources] Customer # 1 requesting 2 2 1 0 Available = 3  8  5  6   Denied


[Resources] Current Resource State:
            Available =         [3 8 5 6]

            Allocation =        [0 0 0 0][0 4 4 0][0 0 0 0]
            Max =               [1 5 5 0][0 9 5 2][3 2 2 5]
            Need =              [1 5 5 0][0 13 9 2][3 2 2 5]

--------------------------------------------------------------------------------
```

These requests are denied even though there are enough resources available since they would not leave the system in a safe state

```
[CPU2] Executed Process 2s program

[CPU2] Clock = 8
...
[CPU1] Decoding Process 5s program
```

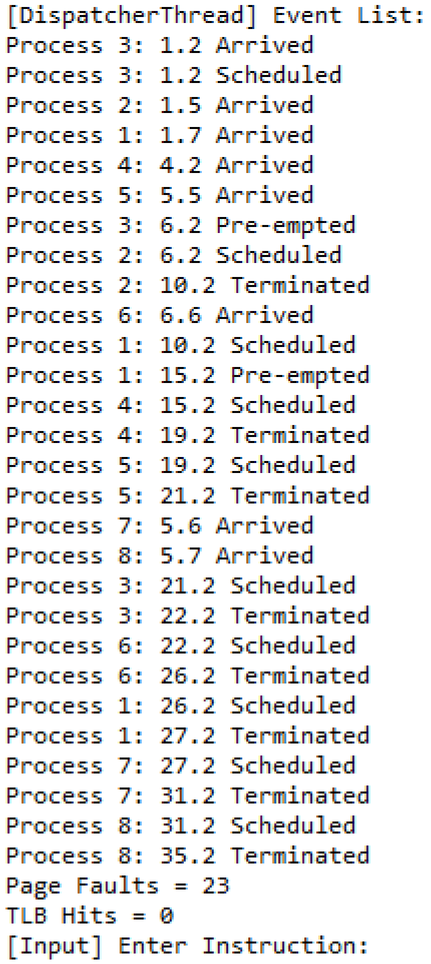
CPU burst is count of instructions, 4 for this program. This config file sets CPU2 frequency to slow, so clock increase is double CPU burst of program.

```
---------------------------------- PROGRAM 5 ----------------------------------

[ProcessCreationThread] Added new process 7


[ProcessCreationThread] Added new process 8

--------------------------------------------------------------------------------
```

Example of processes program which creates new processes that are integrated into scheduler to be executed

```
[CPU1] Executed Process 5s program

[CPU1] Clock = 22

[MemoryManagementUnit] Deallocated program 5 [3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3

[ProcessCreationThread] Produced : 7 to background queue
[ProcessCreationThread] Produced : 8 to background queue
[DispatcherThread] Added new PCBs to queue

[MemoryManagementUnit] Allocated program 7 [3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 ,

[MemoryManagementUnit] Allocated program 8 [3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3 ,

[DispatcherThread] Updated running buffer
```

I have explained the functionalities that parts of the output of this test simulation shows. It shows examples of booting, buffer use, multithreading, synchronisation, deadlock prevention, memory management, page replacement, multitasking, multiprocessing, and process scheduling. If you run the same test yourself, you will see the same logic continued and the rest of virtual memory when programs are allocated and deallocated (CPUs used may vary). I will explain how the algorithms that produce this output work in the technical manual. This simulation should end with this output of events and then allow you to enter more instructions.

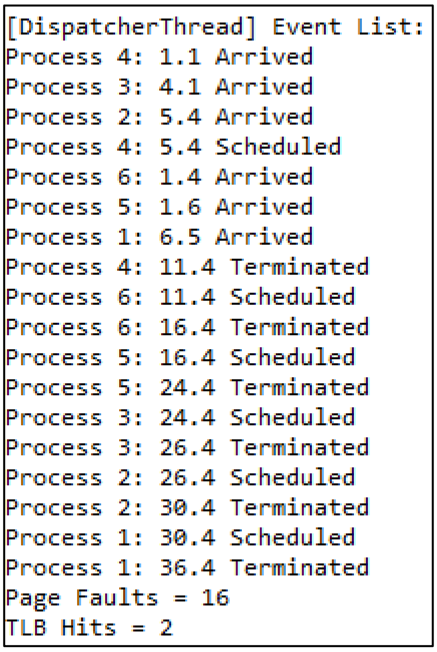


```
[DispatcherThread] Event List:
Process 3: 1.2 Arrived
Process 3: 1.2 Scheduled
Process 2: 1.5 Arrived
Process 1: 1.7 Arrived
Process 4: 4.2 Arrived
Process 5: 5.5 Arrived
Process 3: 6.2 Pre-empted
Process 2: 6.2 Scheduled
Process 2: 10.2 Terminated
Process 6: 6.6 Arrived
Process 1: 10.2 Scheduled
Process 1: 15.2 Pre-empted
Process 4: 15.2 Scheduled
Process 4: 19.2 Terminated
Process 5: 19.2 Scheduled
Process 5: 21.2 Terminated
Process 7: 5.6 Arrived
Process 8: 5.7 Arrived
Process 3: 21.2 Scheduled
Process 3: 22.2 Terminated
Process 6: 22.2 Scheduled
Process 6: 26.2 Terminated
Process 1: 26.2 Scheduled
Process 1: 27.2 Terminated
Process 7: 27.2 Scheduled
Process 7: 31.2 Terminated
Process 8: 31.2 Scheduled
Process 8: 35.2 Terminated
Page Faults = 23
TLB Hits = 0
[Input] Enter Instruction:
```

To demonstrate the other scheduling algorithms, I will just show the event list for each which can be compared to the process input folder to prove they are working accurately. I will use config1.txt for these tests which will use ProcessInputSample1.txt but either test case would work, and the page replacement algorithm will only effect page faults and TLB hits not the events.

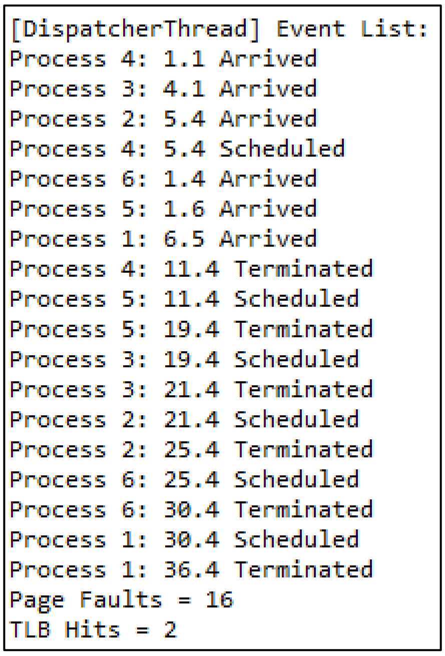
Name: Imogen Hay **URN:** 6579619 **Username:** ih00264

FCFS (0)



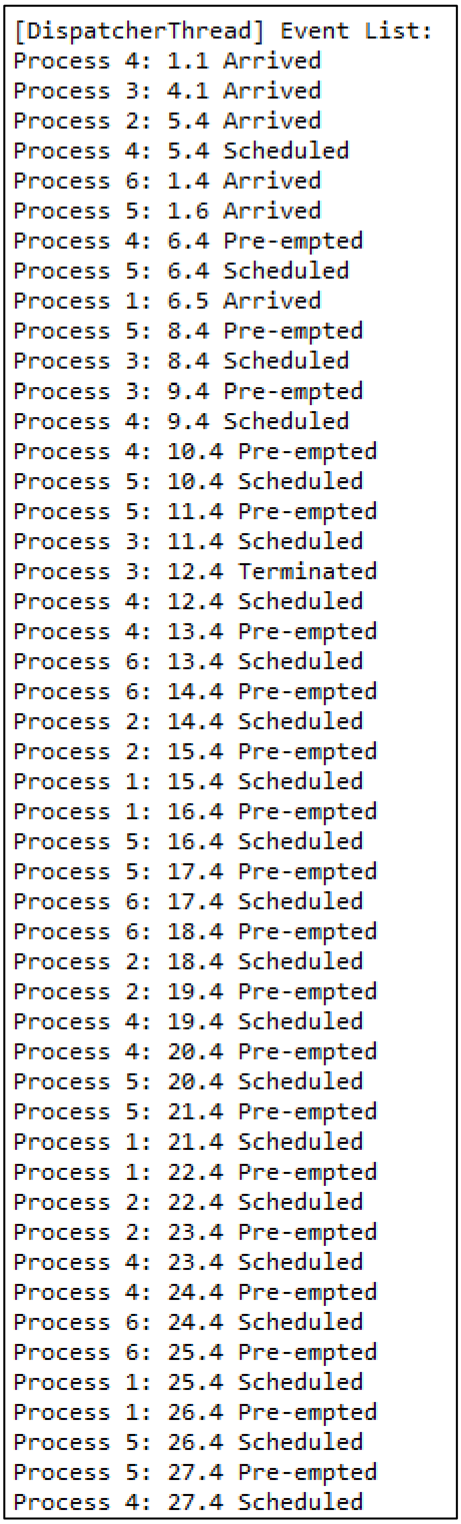
```
[DispatcherThread] Event List:
Process 4: 1.1 Arrived
Process 3: 4.1 Arrived
Process 2: 5.4 Arrived
Process 4: 5.4 Scheduled
Process 6: 1.4 Arrived
Process 5: 1.6 Arrived
Process 1: 6.5 Arrived
Process 4: 11.4 Terminated
Process 6: 11.4 Scheduled
Process 6: 16.4 Terminated
Process 5: 16.4 Scheduled
Process 5: 24.4 Terminated
Process 3: 24.4 Scheduled
Process 3: 26.4 Terminated
Process 2: 26.4 Scheduled
Process 2: 30.4 Terminated
Process 1: 30.4 Scheduled
Process 1: 36.4 Terminated
Page Faults = 16
TLB Hits = 2
```

HPFSN (3)



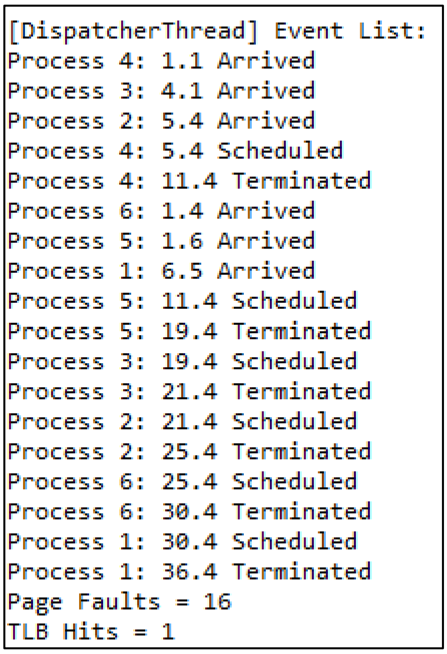
```
[DispatcherThread] Event List:
Process 4: 1.1 Arrived
Process 3: 4.1 Arrived
Process 2: 5.4 Arrived
Process 4: 5.4 Scheduled
Process 6: 1.4 Arrived
Process 5: 1.6 Arrived
Process 1: 6.5 Arrived
Process 4: 11.4 Terminated
Process 5: 11.4 Scheduled
Process 5: 19.4 Terminated
Process 3: 19.4 Scheduled
Process 3: 21.4 Terminated
Process 2: 21.4 Scheduled
Process 2: 25.4 Terminated
Process 6: 25.4 Scheduled
Process 6: 30.4 Terminated
Process 1: 30.4 Scheduled
Process 1: 36.4 Terminated
Page Faults = 16
TLB Hits = 2
```

HPFD (5)



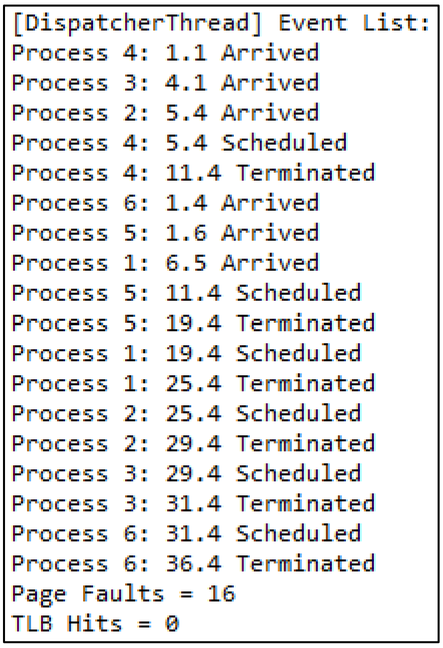
```
[DispatcherThread] Event List:
Process 4: 1.1 Arrived
Process 3: 4.1 Arrived
Process 2: 5.4 Arrived
Process 4: 5.4 Scheduled
Process 6: 1.4 Arrived
Process 5: 1.6 Arrived
Process 4: 6.4 Pre-empted
Process 5: 6.4 Scheduled
Process 1: 6.5 Arrived
Process 5: 8.4 Pre-empted
Process 3: 8.4 Scheduled
Process 3: 9.4 Pre-empted
Process 4: 9.4 Scheduled
Process 4: 10.4 Pre-empted
Process 5: 10.4 Scheduled
Process 5: 11.4 Pre-empted
Process 3: 11.4 Scheduled
Process 3: 12.4 Terminated
Process 4: 12.4 Scheduled
Process 4: 13.4 Pre-empted
Process 6: 13.4 Scheduled
Process 6: 14.4 Pre-empted
Process 2: 14.4 Scheduled
Process 2: 15.4 Pre-empted
Process 1: 15.4 Scheduled
Process 1: 16.4 Pre-empted
Process 5: 16.4 Scheduled
Process 5: 17.4 Pre-empted
Process 6: 17.4 Scheduled
Process 6: 18.4 Pre-empted
Process 2: 18.4 Scheduled
Process 2: 19.4 Pre-empted
Process 4: 19.4 Scheduled
Process 4: 20.4 Pre-empted
Process 5: 20.4 Scheduled
Process 5: 21.4 Pre-empted
Process 1: 21.4 Scheduled
Process 1: 22.4 Pre-empted
Process 2: 22.4 Scheduled
Process 2: 23.4 Pre-empted
Process 4: 23.4 Scheduled
Process 4: 24.4 Pre-empted
Process 6: 24.4 Scheduled
Process 6: 25.4 Pre-empted
Process 1: 25.4 Scheduled
Process 1: 26.4 Pre-empted
Process 5: 26.4 Scheduled
Process 5: 27.4 Pre-empted
Process 4: 27.4 Scheduled
```

HPFSP (4)



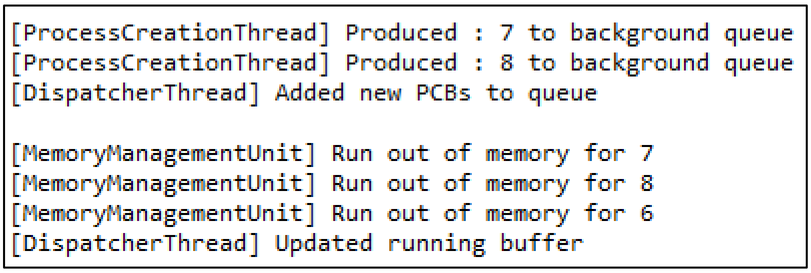
```
[DispatcherThread] Event List:
Process 4: 1.1 Arrived
Process 3: 4.1 Arrived
Process 2: 5.4 Arrived
Process 4: 5.4 Scheduled
Process 4: 11.4 Terminated
Process 6: 1.4 Arrived
Process 5: 1.6 Arrived
Process 1: 6.5 Arrived
Process 5: 11.4 Scheduled
Process 5: 19.4 Terminated
Process 3: 19.4 Scheduled
Process 3: 21.4 Terminated
Process 2: 21.4 Scheduled
Process 2: 25.4 Terminated
Process 6: 25.4 Scheduled
Process 6: 30.4 Terminated
Process 1: 30.4 Scheduled
Process 1: 36.4 Terminated
Page Faults = 16
TLB Hits = 1
```

WRR (6)



```
[DispatcherThread] Event List:
Process 4: 1.1 Arrived
Process 3: 4.1 Arrived
Process 2: 5.4 Arrived
Process 4: 5.4 Scheduled
Process 4: 11.4 Terminated
Process 6: 1.4 Arrived
Process 5: 1.6 Arrived
Process 1: 6.5 Arrived
Process 5: 11.4 Scheduled
Process 5: 19.4 Terminated
Process 1: 19.4 Scheduled
Process 1: 25.4 Terminated
Process 2: 25.4 Scheduled
Process 2: 29.4 Terminated
Process 3: 29.4 Scheduled
Process 3: 31.4 Terminated
Process 6: 31.4 Scheduled
Process 6: 36.4 Terminated
Page Faults = 16
TLB Hits = 0
```

If the memory management unit runs out of RAM for a process it will swap pages however if it runs out of virtual memory (e.g. if you chose config2.txt and scheduling algorithm 5) this happens:



```
[ProcessCreationThread] Produced : 7 to background queue
[ProcessCreationThread] Produced : 8 to background queue
[DispatcherThread] Added new PCBs to queue

[MemoryManagementUnit] Run out of memory for 7
[MemoryManagementUnit] Run out of memory for 8
[MemoryManagementUnit] Run out of memory for 6
[DispatcherThread] Updated running buffer
```

Which results in these processes never being added to the background queue so that they are not scheduled or processed since there is not enough memory for their program to be stored. If you run this example you will see that these processes are not included in the event list.

New Tests:

My project contains 2 sets of example input files but to create a new test case, you must first create a new config file. You can copy the text from one of the current files then change the values on each line after the ':' leaving a space after it. Make sure the amount of virtual memory (PAGE_SIZE * NUMBER_OF_PAGES) is equal to or bigger than the RAM_SIZE and all values are positive.
To add a new resource, increment the number of resources by 1 then add a new line before BUFFER_SIZE. Format this line as 'R[number of resources]: [amount of this resource]'. The CPU frequencies can be set to either fast, moderate, or slow which will affect clock speed.

Next create a new resource input folder. The first two lines should be comments. The third is the number of customers. Fourth line is a comment. Each line after this will represent a customer. The amount of numbers on each line separated by a comma is the amount of resources which you defined in the config file. Format each line as '[amount of first resource customer needs],[amount of second resource customer needs], amount of third resource customer needs],…

Finally, you will need to create a process input file. The first two lines and fourth line are for comments, they must begin with a # so the program ignores these lines. The third line should be a single integer representing the number of processes. Each process is on its own line and is made up of 5 numbers separated by spaces. The format should be:
[ID] [arrival time] [expected run time/ cpu burst] [priority] [program size] e.g. 1 6.5 6 1 12
Optionally you can add a program for each process. To do this on the line below where the process is defined write 'Code:' then on the next line an instruction formatted " [Instruction] ". You can have as many instructions as you like, each on a separate line. After writing all instructions on a new line write 'exit;' then you can add another process. I will discuss which instructions are supported by this OS in the architecture section, but you can see some examples in the sample files. If you are including programs for a process, I recommend making the CPU burst the number of instructions in the program and the program size double this to better demonstrate my memory management unit. If you just want to see the process scheduling algorithms working you can leave out the programs and make these values anything you would like.

It is important that you name each text file using the same number since in my program the process and resource input files are automatically selected depending on which configuration file you chose. E.g. choosing config3.txt will use ProcessInputSample3.txt and ResourceInputSample3.txt. Make sure you use a different number for each set of input files to prevent inconsistencies and that the input files are in the project but not in the source folder (as shown in code setup), so the directory used automatically by the program is correct.

3 Technical Manual

Booting:

The booting steps for my OS are implemented in a main method in the class MainBooting.java so that it boots when the program is run. It works by finding the current user directory where the project is being stored so that the input files can be easily accessed. Some previously explained input is required by the user and then the OS begins loading the kernel image and initialising the data structures used in my simulation. I decided to use a configuration text file for the OS to identify hardware instead of hardcoding them so that a user can easy change these configurations to test different capabilities of my system such as the different algorithms supported.

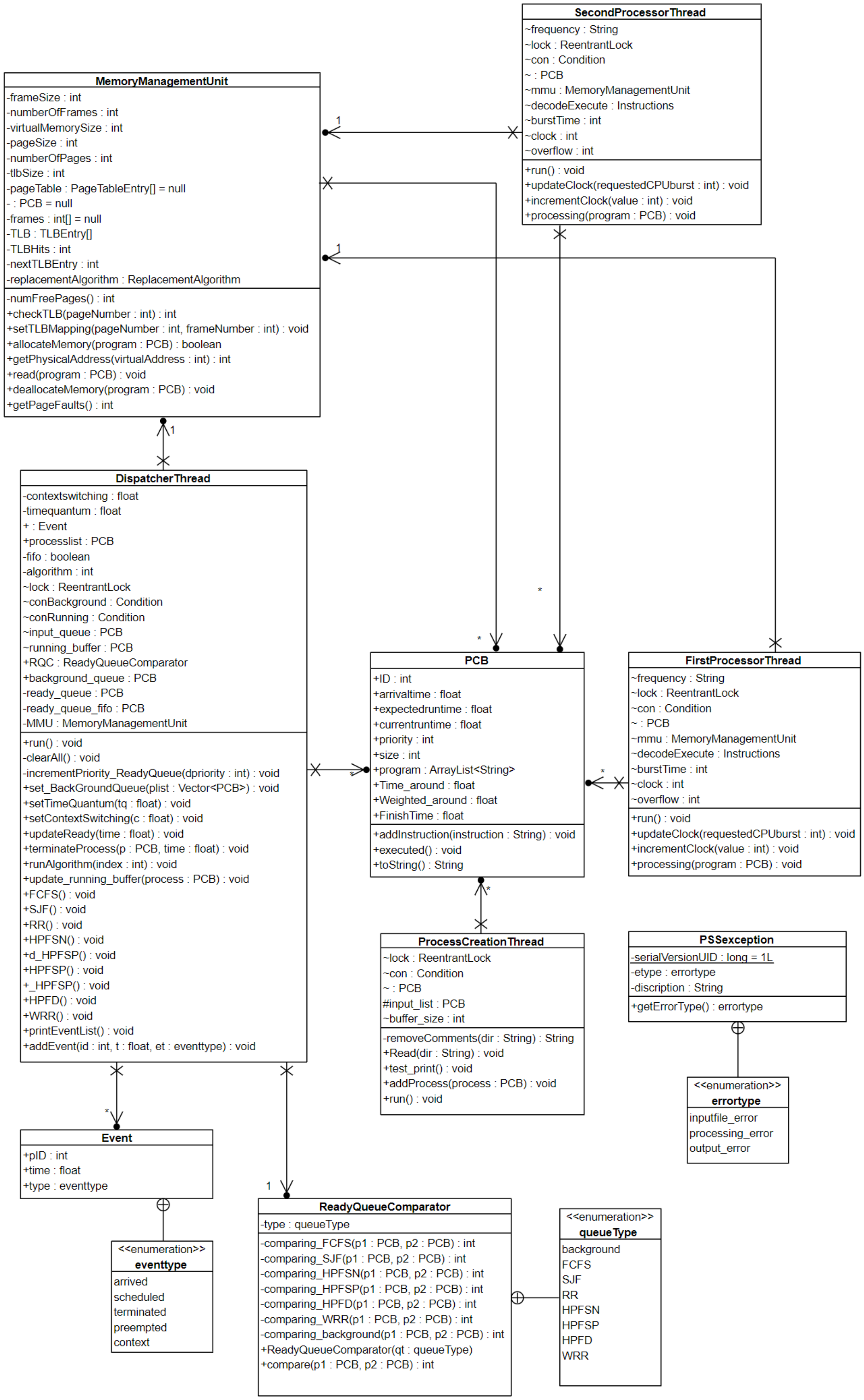
My memory management unit data structure is initialised using the RAM, frame size, page size, number of pages and TLB size obtained from the config file and the page replacement algorithm selected by user input.
Banker's algorithm is then initialised using the resource values acquired from the config file.
Locks and conditions are then initialised for thread synchronisation. The 4 threads are then created, their priority set and then started.

Process Scheduler:

My OS implements a multitasking process scheduler subsystem in which two processors interleave their time on ready processes according to the chosen scheduling policy. It was suggested that I create a separate dispatcher for each processor however I decided to use one dispatcher which receives processes then schedules them according to the chosen algorithm because this means only one scheduler is needed which is more efficient. When a process is scheduled it is stored on a running buffer. One of the free processors will then consume it and execute its program.

I chose to implement a multitasking OS instead of single tasked to allow processor time to be shared more evenly among processes resulting in better use of computer resources.
I implemented 7 scheduling algorithms for the user to choose from to allow more flexibility and since certain algorithms are more suited to specific tasks e.g. if all jobs are the same/similar length SJF will not be effective.
The classes used to implement this subsystem is illustrated below:

Name: Imogen Hay URN: 6579619 Username: ih00264

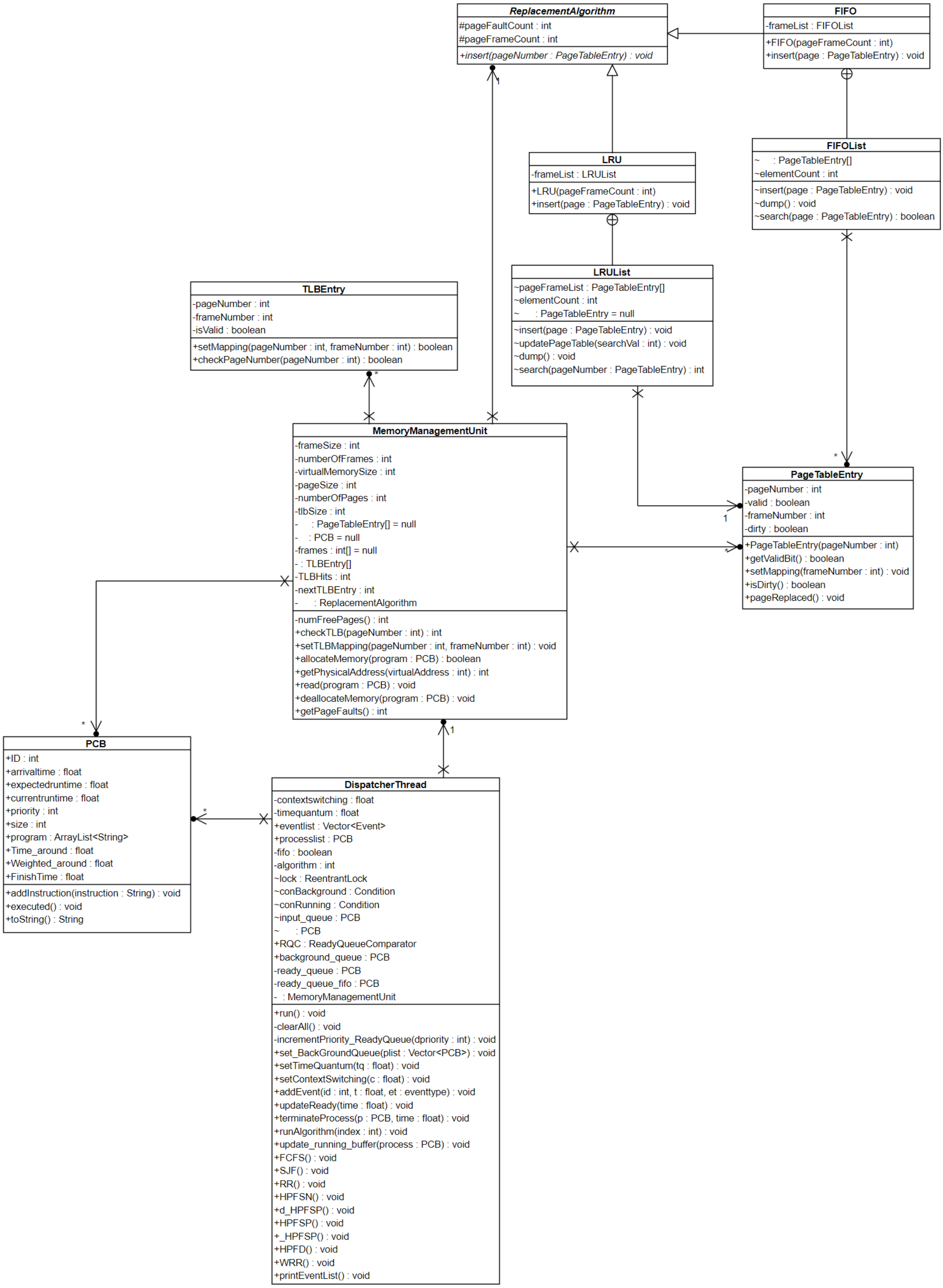

SecondProcessorThread
~frequency : String
~lock : ReentrantLock
~con : Condition
~ : PCB
~mmu : MemoryManagementUnit
~decodeExecute : Instructions
~burstTime : int
~clock : int
~overflow : int
+run() : void
+updateClock(requestedCPUburst : int) : void
+incrementClock(value : int) : void
+processing(program : PCB) : void
MemoryManagementUnit
-frameSize : int
-numberOfFrames : int
-virtualMemorySize : int
-pageSize : int
-numberOfPages : int
-tlbSize : int
-pageTable : PageTableEntry[] = null
- : PCB = null
-frames : int[] = null
-TLB : TLBEntry[]
-TLBHits : int
-nextTLBEntry : int
-replacementAlgorithm : ReplacementAlgorithm
-numFreePages() : int
+checkTLB(pageNumber : int) : int
+setTLBMapping(pageNumber : int, frameNumber : int) : void
+allocateMemory(program : PCB) : boolean
+getPhysicalAddress(virtualAddress : int) : int
+read(program : PCB) : void
+deallocateMemory(program : PCB) : void
+getPageFaults() : int
1
1
1
DispatcherThread
-contextswitching : float
-timequantum : float
+ : Event
+processlist : PCB
-fifo : boolean
-algorithm : int
~lock : ReentrantLock
~conBackground : Condition
~conRunning : Condition
~input_queue : PCB
~running_buffer : PCB
+RQC : ReadyQueueComparator
+background_queue : PCB
-ready_queue : PCB
-ready_queue_fifo : PCB
-MMU : MemoryManagementUnit
+run() : void
-clearAll() : void
-incrementPriority_ReadyQueue(dpriority : int) : void
+set_BackGroundQueue(plist : Vector<PCB>) : void
+setTimeQuantum(tq : float) : void
+setContextSwitching(c : float) : void
+updateReady(time : float) : void
+terminateProcess(p : PCB, time : float) : void
+runAlgorithm(index : int) : void
+update_running_buffer(process : PCB) : void
+FCFS() : void
+SJF() : void
+RR() : void
+HPFSN() : void
+d_HPFSP() : void
+HPFSP() : void
+_HPFSP() : void
+HPFD() : void
+WRR() : void
+printEventList() : void
+addEvent(id : int, t : float, et : eventtype) : void
*
*
*
PCB
+ID : int
+arrivaltime : float
+expectedruntime : float
+currentruntime : float
+priority : int
+size : int
+program : ArrayList<String>
+Time_around : float
+Weighted_around : float
+FinishTime : float
+addInstruction(instruction : String) : void
+executed() : void
+toString() : String
*
*
FirstProcessorThread
~frequency : String
~lock : ReentrantLock
~con : Condition
~ : PCB
~mmu : MemoryManagementUnit
~decodeExecute : Instructions
~burstTime : int
~clock : int
~overflow : int
+run() : void
+updateClock(requestedCPUburst : int) : void
+incrementClock(value : int) : void
+processing(program : PCB) : void
*
ProcessCreationThread
~lock : ReentrantLock
~con : Condition
~ : PCB
#input_list : PCB
~buffer_size : int
-removeComments(dir : String) : String
+Read(dir : String) : void
+test_print() : void
+addProcess(process : PCB) : void
+run() : void
PSSexception
-serialVersionUID : long = 1L
-etype : errortype
-discription : String
+getErrorType() : errortype
<<enumeration>>
errortype
inputfile_error
processing_error
output_error
*
Event
+pID : int
+time : float
+type : eventtype
1
<<enumeration>>
eventtype
arrived
scheduled
terminated
preempted
context
ReadyQueueComparator
-type : queueType
-comparing_FCFS(p1 : PCB, p2 : PCB) : int
-comparing_SJF(p1 : PCB, p2 : PCB) : int
-comparing_HPFSN(p1 : PCB, p2 : PCB) : int
-comparing_HPFSP(p1 : PCB, p2 : PCB) : int
-comparing_HPFD(p1 : PCB, p2 : PCB) : int
-comparing_WRR(p1 : PCB, p2 : PCB) : int
-comparing_background(p1 : PCB, p2 : PCB) : int
+ReadyQueueComparator(qt : queueType)
+compare(p1 : PCB, p2 : PCB) : int
<<enumeration>>
queueType
background
FCFS
SJF
RR
HPFSN
HPFSP
HPFD
WRR

PCB – Used to create process control block for each process that can be accessed by other claseses. Stores information about process and its program. Used to determine whether program has been executed.
PSSException – Not my own work. Used to detect errors whenever a PCB is created. [2]
ProcessCreationThread – Reads processes from input file and uses them to create PCBs which are produced to buffer to be consumed by DispatcherThread.
DispatcherThread – Acts as process scheduler and dispatcher to processors. Contains and uses each scheduling algorithm and controls memory management unit. Consumes background queue from ProcessCreationThread adds to ready queue when scheduled. Produces running buffer which is consumed by a waiting processor. Used scheduling algorithm methods from another scheduler. [2]
ReadyComparator – Not my own work. Used by scheduling algorithms in DispatcherThread to compare PCBs to schedule their processes. [2]
Event – Not my own work. Used to indicate scheduling events and keep event list. [2]
FirstProcessorThread – Simulates fetching, decoding, and executing process.
SecondProcessorThread – Simulates fetching, decoding, and executing process.
Instructions – Stores instruction set used by processors for decoding programs instructions.
MMU – Used by dispatcher and CPUs for allocating and reading from memory. Shared by CPUs.

Each class is commented to further explain the purpose of each method.

Memory Management Unit

This OS implements a memory management unit, which also acts as a virtual memory manager. It works with my process scheduler by allocating virtual memory to a process when it arrives, reading it into main memory when it is scheduled (which may involve swapping if main memory is full) and finally by deallocating it from memory when it is terminated so more space is free. Since this OS is just a simulation to keep things simple each processes program is represented in memory by storing the process, which can be used as a pointer to the program. So, if the program is size 10, ten positions in memory will be filled with that process which is why in the test cases you can just see a list of numbers (process IDs).
I used page and frame structures to partition memory. Main memory is split into frames of fixed size (defined in config) and virtual memory split into pages of the same size. I decided to use paging instead of segmentation because it is easier to map pages to frames than it is for segments since an offset is not required and paging allows a process to be stored in a non-contiguous manor which prevents external fragmentation. The downside of paging is that it can leave gaps in memory if the program size is not divisible by page size but my OS copes with this by storing the empty gaps in the page as null PCBs. I use this algorithm to find the physical address from a virtual address 'x' as seen in getPhysicalAddress: [frame page of x in * frame size] + [position of x in page]
The position of x in the page is found using x-[page*page]. The frame a page is in is found using the page table but first the list of TLB entries is used. I decided to use TLB since it is a type of cache that stores page frame mapping so can be used to quickly find out the frame a page is currently in, increasing overall efficiency of this subsystem. My cache is replaced on a first in first out bases.
I implemented LRU and FIFO replacement algorithms to allow more flexibility of the system and they seem more efficient than MFU (most frequently used) since if a page is used a lot it is likely to be needed again. In my design RAM begins empty since at the beginning no processes have arrived with programs and only swaps programs in when needed by CPU (lazy swapper).


ReplacementAlgorithm
#pageFaultCount : int
#pageFrameCount : int
+insert(pageNumber : PageTableEntry) : void
FIFO
-frameList : FIFOList
+FIFO(pageFrameCount : int)
+insert(page : PageTableEntry) : void
LRU
-frameList : LRUList
+LRU(pageFrameCount : int)
+insert(page : PageTableEntry) : void
FIFOList
~ : PageTableEntry[]
~elementCount : int
~insert(page : PageTableEntry) : void
~dump() : void
~search(page : PageTableEntry) : boolean
LRUList
~pageFrameList : PageTableEntry[]
~elementCount : int
~ : PageTableEntry = null
~insert(page : PageTableEntry) : void
~updatePageTable(searchVal : int) : void
~dump() : void
~search(pageNumber : PageTableEntry) : int
TLBEntry
-pageNumber : int
-frameNumber : int
-isValid : boolean
+setMapping(pageNumber : int, frameNumber : int) : boolean
+checkPageNumber(pageNumber : int) : boolean
1
*
MemoryManagementUnit
-frameSize : int
-numberOfFrames : int
-virtualMemorySize : int
-pageSize : int
-numberOfPages : int
-tlbSize : int
- : PageTableEntry[] = null
- : PCB = null
-frames : int[] = null
- : TLBEntry[]
-TLBHits : int
-nextTLBEntry : int
- : ReplacementAlgorithm
-numFreePages() : int
+checkTLB(pageNumber : int) : int
+setTLBMapping(pageNumber : int, frameNumber : int) : void
+allocateMemory(program : PCB) : boolean
+getPhysicalAddress(virtualAddress : int) : int
+read(program : PCB) : void
+deallocateMemory(program : PCB) : void
+getPageFaults() : int
*
1
PageTableEntry
-pageNumber : int
-valid : boolean
-frameNumber : int
-dirty : boolean
+PageTableEntry(pageNumber : int)
+getValidBit() : boolean
+setMapping(frameNumber : int) : void
+isDirty() : boolean
+pageReplaced() : void
*
1
PCB
+ID : int
+arrivaltime : float
+expectedruntime : float
+currentruntime : float
+priority : int
+size : int
+program : ArrayList<String>
+Time_around : float
+Weighted_around : float
+FinishTime : float
+addInstruction(instruction : String) : void
+executed() : void
+toString() : String
*
DispatcherThread
-contextswitching : float
-timequantum : float
+eventlist : Vector<Event>
+processlist : PCB
-fifo : boolean
-algorithm : int
~lock : ReentrantLock
~conBackground : Condition
~conRunning : Condition
~input_queue : PCB
~ : PCB
+RQC : ReadyQueueComparator
+background_queue : PCB
-ready_queue : PCB
-ready_queue_fifo : PCB
- : MemoryManagementUnit
+run() : void
-clearAll() : void
-incrementPriority_ReadyQueue(dpriority : int) : void
+set_BackGroundQueue(plist : Vector<PCB>) : void
+setTimeQuantum(tq : float) : void
+setContextSwitching(c : float) : void
+addEvent(id : int, t : float, et : eventtype) : void
+updateReady(time : float) : void
+terminateProcess(p : PCB, time : float) : void
+runAlgorithm(index : int) : void
+update_running_buffer(process : PCB) : void
+FCFS() : void
+SJF() : void
+RR() : void
+HPFSN() : void
+d_HPFSP() : void
+HPFSP() : void
+_HPFSP() : void
+HPFD() : void
+WRR() : void
+printEventList() : void

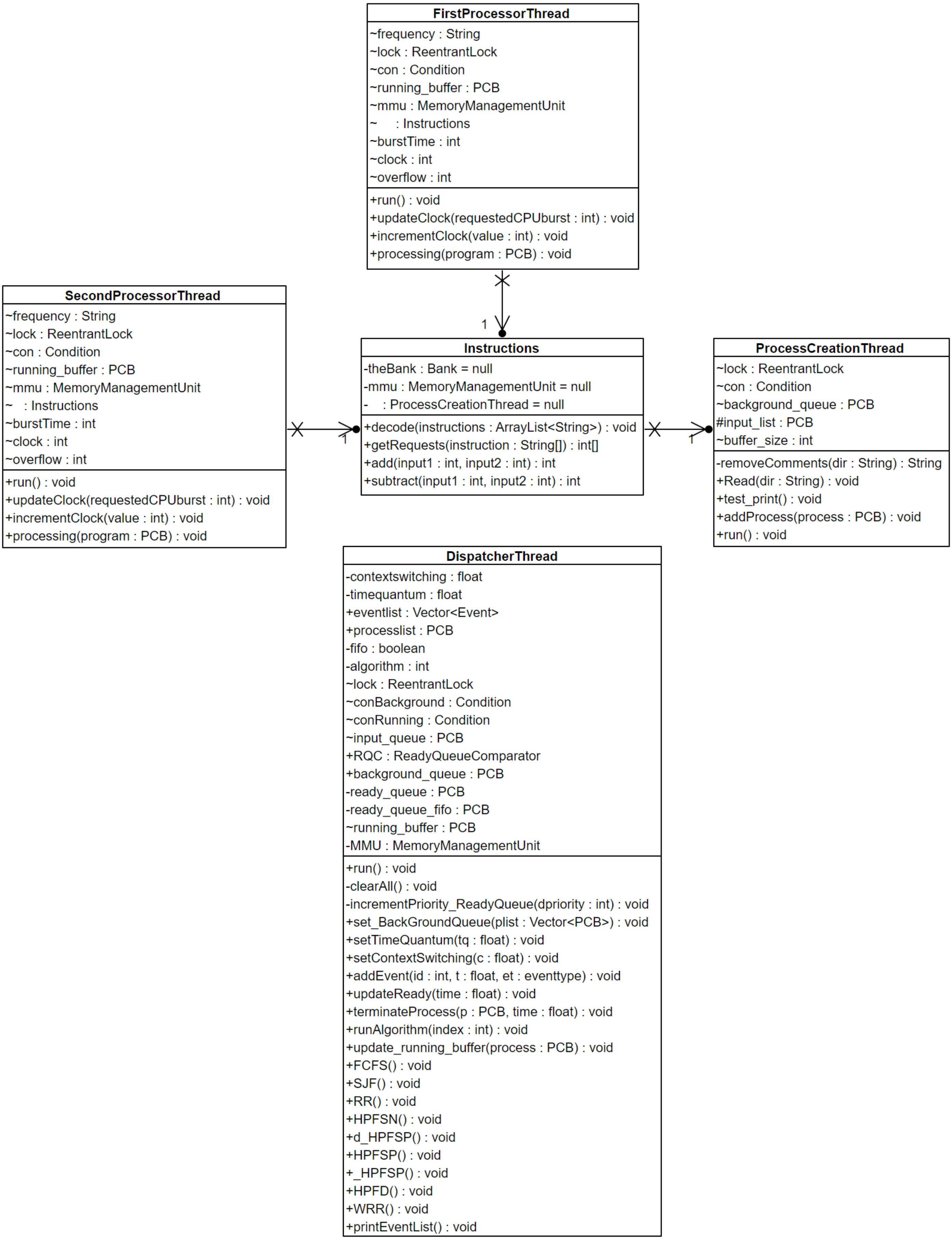
DispatcherThread – Indicates to MMU when memory needs to be allocated, deallocated, or read from (moved to main memory). If virtual memory is full and allocation fails, the process will not be added to the background queue so will not be scheduled or processed.
PCB – Used to represent a program stored in memory since can be used to access program. Null PCBs are created to represent empty parts of virtual memory which is usually a page but can be part of a dirty page if internal fragmentation has occurred.
MemoryManagementUnit – Other parts of OS use this class to send read and write instructions and indicate when memory can be deallocated. It initialises data structures like the list of PageTable and TLB entries and page replacement algorithms which are used to control and monitor memory. Uses first fit allocation so does not need to check all VM for a space (more time efficient).
PageTableEntry – Represents page. Stores frame number of page, negative if not in RAM. Dirty if page is written to. Valid if page is currently in RAM.
ReplacementAlgorithm – Adapted from sample code [1]. Abstract class used by FIFO and LRU. Each of these child classes will use a different strategy to swap PageTableEntries into frames. When a page is inserted a mapping is set, valid set to true. When a page is replaced valid is set to false and mapping is removed.
FIFO – Adapted from sample code [1]. First page to be put in frame is first to be replaced.
LRU – Adapted from sample code [1]. Least recently accessed page is replaced.
TLBEntry – Taken from sample code [1]. Stores a pageFrame mapping that can be quickly accessed since acts a cache memory.

Each class is commented to further explain the purpose of each method.

Multiprocessing and Multithreading:

To implement multiprocessing and multithreading I created an OS with two CPU cores. Each processors fetch, decode, execute cycle is represented as a class which are both threads. The DispatcherThread and ProcessCreationThread are also both implemented as threads. To make these classes threads I extended the Thread class instead of implementing the Runnable interface. I chose this option so that I could create an instance of the threads before I wanted to run them which makes it easier to simulate the booting process.
There were multiple ways I could have applied threading to my system for example I could have used an I/O Manager thread but I decided there was not enough I/O in my system that would require its own manager since I did not include an I/O subsystem. I also could have included a separate scheduler thread for each processor, but my OS worked effectively with just one scheduler for both CPUs. Instead I chose to implement ProcessCreationThread which produces a buffer of PCBs created from processes in the input file or created by other processes. The DispatcherThread consumes this buffer, schedules the processes then produces them to a running buffer. The CPUs then consumes a PCB from the buffer and processes the process. I made the processor threads a lower priority than the others so that PCBs are scheduled before they are processed. Both buffers must be synchronised which is explained in the next section. This uses polled I/O since ProcessCreationThread continually checks for new input and the other threads continually check if more PCBs have been produced to the synchronised buffers.
In theory the processes in the running buffer will be dispatched to whichever CPU is currently free allowing both processors to be working at the same time however since the processes I tested are executed before the next process is scheduled both end up being free so one is randomly chosen.

Name: Imogen Hay URN: 6579619 Username: ih00264


FirstProcessorThread
~frequency : String
~lock : ReentrantLock
~con : Condition
~running_buffer : PCB
~mmu : MemoryManagementUnit
~ : Instructions
~burstTime : int
~clock : int
~overflow : int
+run() : void
+updateClock(requestedCPUburst : int) : void
+incrementClock(value : int) : void
+processing(program : PCB) : void
1
SecondProcessorThread
~frequency : String
~lock : ReentrantLock
~con : Condition
~running_buffer : PCB
~mmu : MemoryManagementUnit
~ : Instructions
~burstTime : int
~clock : int
~overflow : int
+run() : void
+updateClock(requestedCPUburst : int) : void
+incrementClock(value : int) : void
+processing(program : PCB) : void
1
Instructions
-theBank : Bank = null
-mmu : MemoryManagementUnit = null
- : ProcessCreationThread = null
+decode(instructions : ArrayList<String>) : void
+getRequests(instruction : String[]) : int[]
+add(input1 : int, input2 : int) : int
+subtract(input1 : int, input2 : int) : int
1
ProcessCreationThread
~lock : ReentrantLock
~con : Condition
~background_queue : PCB
#input_list : PCB
~buffer_size : int
-removeComments(dir : String) : String
+Read(dir : String) : void
+test_print() : void
+addProcess(process : PCB) : void
+run() : void
DispatcherThread
-contextswitching : float
-timequantum : float
+eventlist : Vector<Event>
+processlist : PCB
-fifo : boolean
-algorithm : int
~lock : ReentrantLock
~conBackground : Condition
~conRunning : Condition
~input_queue : PCB
+RQC : ReadyQueueComparator
+background_queue : PCB
-ready_queue : PCB
-ready_queue_fifo : PCB
~running_buffer : PCB
-MMU : MemoryManagementUnit
+run() : void
-clearAll() : void
-incrementPriority_ReadyQueue(dpriority : int) : void
+set_BackGroundQueue(plist : Vector<PCB>) : void
+setTimeQuantum(tq : float) : void
+setContextSwitching(c : float) : void
+addEvent(id : int, t : float, et : eventtype) : void
+updateReady(time : float) : void
+terminateProcess(p : PCB, time : float) : void
+runAlgorithm(index : int) : void
+update_running_buffer(process : PCB) : void
+FCFS() : void
+SJF() : void
+RR() : void
+HPFSN() : void
+d_HPFSP() : void
+HPFSP() : void
+_HPFSP() : void
+HPFD() : void
+WRR() : void
+printEventList() : void

ProcessCreationThread – Creates PCBs, produces to background_queue until buffer full.

DispatcherThread – Consumes PCBs from buffer, produces to running_buffer when scheduled. Signals to ProcessCreationThread when more can be produced.

FirstProcessorThread – Consumes PCBs from running_buffer, fetches, decodes, and executes program. Signals to dispatcher when finished.

SecondProcessorThread – Same function as FirstProcessorThread but for second CPU.

Instruction – Used by processors to decode and execute instructions in program. This may involve input to the ProcessCreationThread, Bank or MMU which is why they are connected.

Each class is commented to further explain the purpose of each method.

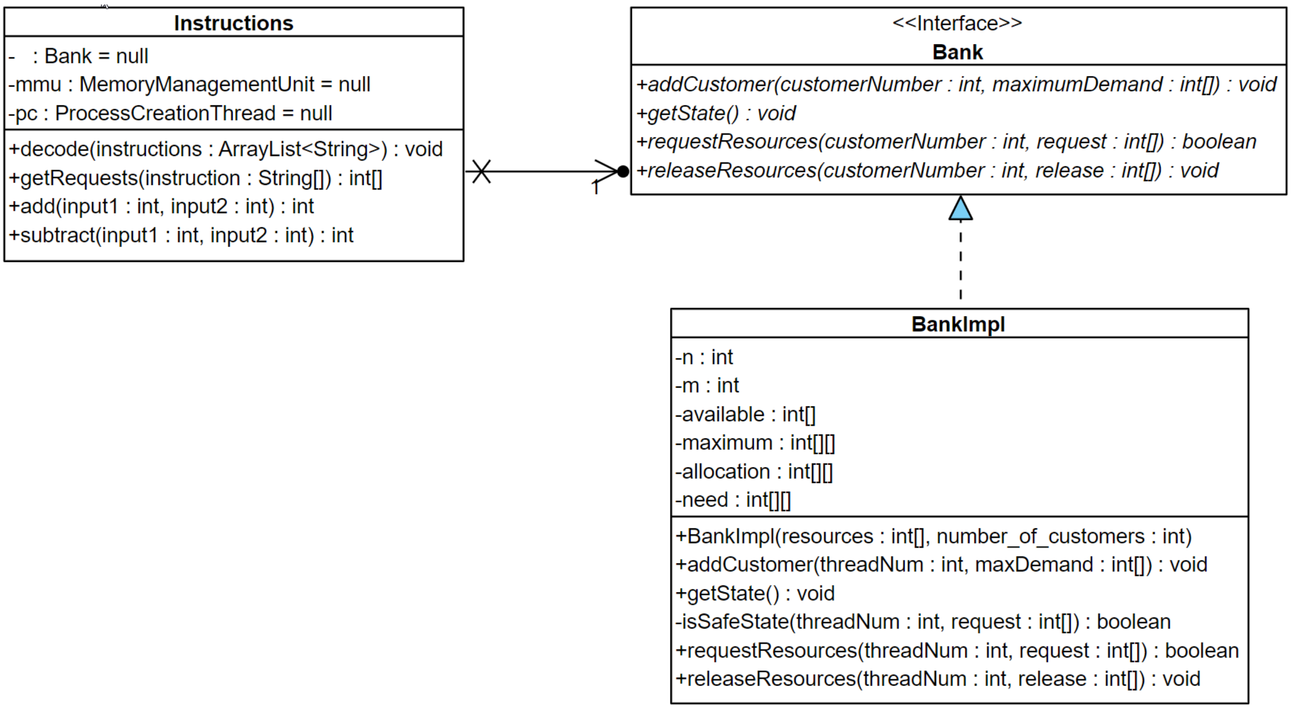
Synchronization and Deadlock Prevention:

I decided to make process execution synchronous instead of asynchronous which involved locking the DispatcherThread until a process was complete. I chose this option because although synchronisation is slower the processes are completed in the correct order instead of getting mixed up. This helps to achieve good parallelism.

I synchronized the background_queue and running_buffer since they are both used by multiple threads which add and remove values to them. I could have achieved this using semaphores or using the synchronised keyword as a monitor however I preferred using lock and conditions since this method gives me more control when passing object ownership between threads. I implemented one lock and two conditions. So, one condition for each producer/consumer pair and object to synchronise.

Deadlock occurs when a set of processes competing for system resources or communicating with each other are permanently blocked. This can be caused when each of my processes is awaiting an event that can only be triggered by another blocked process in the set. It is permanent since no events are ever triggered. To prevent this from happening in my system I had to make sure my threads with synchronised elements (e.g. consumable buffers) were locked and unlocked at correct times and only wait when another thread has been signalled which can continue or when input is needed so it is not permanently blocked.

Deadlocks could also occur from processes using too many reusable resources leaving the OS in an unsafe state. To demonstrate this kind of deadlock prevention, I used banker's algorithm from lab 6. It was recommended to apply this method to an I/O thread, but I did not implement this thread, so the requests and releases are inputted as instructions which are part of different processes programs. Since the order of execution of these programs will vary sometimes a release command could be executed before a process had been allocated anything. This meant I had to adapt the banker's algorithm given to allow releases only if the customer currently holds those resources. I decided to use this algorithm instead on an alternative such as a resource allocation graph because my OS is a multi-instance resource system not single instance.


Instructions
-   : Bank = null
-mmu : MemoryManagementUnit = null
-pc : ProcessCreationThread = null
+decode(instructions : ArrayList<String>) : void
+getRequests(instruction : String[]) : int[]
+add(input1 : int, input2 : int) : int
+subtract(input1 : int, input2 : int) : int
1
<<Interface>>
Bank
+addCustomer(customerNumber : int, maximumDemand : int[]) : void
+getState() : void
+requestResources(customerNumber : int, request : int[]) : boolean
+releaseResources(customerNumber : int, release : int[]) : void
BankImpl
-n : int
-m : int
-available : int[]
-maximum : int[][]
-allocation : int[][]
-need : int[][]
+BankImpl(resources : int[], number_of_customers : int)
+addCustomer(threadNum : int, maxDemand : int[]) : void
+getState() : void
-isSafeState(threadNum : int, request : int[]) : boolean
+requestResources(threadNum : int, request : int[]) : boolean
+releaseResources(threadNum : int, release : int[]) : void

Bank – Adapted from sample code [1]. Interface to solution to banker's algorithm.
BankImpl – Adapted from sample code [1]. Solution to banker's algorithm to prevent deadlocks
Instructions – Decodes instruction. If they are resource request or release instruction send to Bank to be checked and approved or denied.

Each class is commented to further explain the purpose of each method. Bankers Algorithm further explained in lab 6.

4 OS Architecture

Instruction Set:

This OS supports multiple instructions which are recognised and executed by the CPUs. Some are part of programs to be executed as part of a process, others can be entered as user input. The decoding for these instructions can be seen in Instructions.java.

1. **X ADD Y** where x and why are both integers and the sum of both is produced.
2. **X SUB Y** where x and why are both integers and the y is subtracted from x.
3. **RQ <customer> <amount of R1> <amount of R2> ect.** requests to obtain resources for customer (process)
4. **RL <customer> <amount of R1> <amount of R2> ect.** requests to release resources for customer (process)
5. ***** displays the current resource state.
6. **READ x** reads physical address of virtual address x.
7. **NEW <ID> <arrival time> <CPU burst> <priority> <program size>** creates a new PCB for this process using values inputted.

All of these instructions can be entered by the user to be decoded and executed however a NEW process will be produced to the buffer but not consumed since the user can only enter input when the scheduling algorithm in the DispatcherThread has finished.

Additionally, the scheduler recognises the numbers 0-6 as instructions of which algorithm to use. What each number represents is detailed in the user manual. The memory management unit recognises 'lru' and 'fifo' as instructions referring to which page sorting algorithm to use and allocateMemory(program), deallocateMemory(program) and read(program) as instructions for memory allocation and deallocation.

Assumed Architecture:

To make an OS simulation work I had to make some assumptions for example that my OS uses two CPU cores. I decided to not allow the amount of CPU cores to be configured since two is enough to demonstrate how different CPU frequencies effect their clocks and simulate multiprocessing. Since there are not that many processes used by my system my OS is efficient enough without the need for more which is also why I assumed that both CPUs share one dispatcher thread.

My OS can be simulated using 3 different CPU frequencies. Fast, moderate, and slow. The system assumes a fast CPUs clock increases by half of CPU burst, moderate is the same, slow is double. I did this because it effectively demonstrates the effect of different CPU frequencies on the clock without adding too much complexity.

The OS also assumes frame size and page size are the same. This simplifies reading from main memory and page swapping since each frame can contain exactly one frame. RAM size, Frame size, amount of VM and TLB size are all assumed by the configuration file however the config file can be changed to simulate different operating systems.

Compatibility:

Since my configuration file can be changed my OS design is compatible with hardware of any RAM and virtual memory size if the amount of RAM is smaller than VM size. The frame and page size can also be any value if both are the same. It also supports any amount of reusable resources and processes. It is compatible with processes of any size or CPU burst.

This means my OS architecture is compatible with computers using simple hardware of any size with two processors of any speed that require memory management and process scheduling subsystems using a range of algorithms. It does not support for example file management or I/O hardware devices. I decided not to include these systems since they can easily be replaced in my design by user I/O using a console and basic input output system and my OS design does not require files so a file manager is not needed. Since my OS included a memory management unit and process scheduler it would be used for devices requiring lots of processing but with only a limited amount of memory and not much user input or file usage e.g. if lots of tasks needed to be completed but size or cost is limited.

This design would not be compatible with the architecture of a mobile phone for instance since they contain enough more expensive memory, enough to not require page swapping so it is not supported. I could have designed my OS to use more cores to increase processing speed however 2 seemed like the optimum value since the process scheduler may not schedule processes quickly enough to make use of more CPUs since they would execute programs far more quickly than they could be scheduled resulting in lots of wasted processor time from waiting and it is likely that not all of them would be used. The overhead time from the CPU scheduling and communicating with each other would also waste time.

5 Bibliography

[1] *A. Silberschatz, P. B. Galvin, and G. Gagne, Operating system concepts with Java*, 8th ed. Hoboken, N.J: John Wiley & Sons, 2010.

[2] *MMayla* (2019) *Process-Scheduling-Simulator* (Version 5.0)[SourceCode]. https://github.com/MMayla/Process-Scheduling-Simulator