HO CHI MINH CITY, UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEER

Operating System

Assignment #2

Simple Operating System

Instructor : MEng. Nguyen Manh Thin
Students: Tran Ngoc Minh Diep - 1952610.
Duong Gia An - 1952163.
Thieu Quang Trung - 1953051.

HO CHI MINH CITY, 05 / 2022

Content

1 Introduction

1.1 An overview

1.2 Source Code

1.3 Processes

1.4 How to Create a Process?

1.5 How to Run the Simulation

2 Implementation

2.1 Scheduler

Question: What is the advantage of using **priority feedback queue** in comparison with other scheduling algorithms you have learned?
These are all scheduling algorithms we have learned:

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Priority Scheduling (PS)
- Shortest Remaining Time First (SRTF)
- Round Robin (RR)
- Multilevel Queues Scheduling (MLQS)
- Multilevel Feedback Queue Scheduling (MLFS)

Compared to others, **priority feedback queue** has some advantages as follows:

- The CPU executes processes in **round-robin** style. Every process gets an equal share of the CPU. Because **round-robin** is cyclic in nature, there is **no starvation**.
- Using 2 priority queues with each assigned priority processes, it is based on **multilevel queues scheduling** and **multilevel feedback queue scheduling**. Because the processes are permanently assigned to the queue, it has advantage of low scheduling overhead, moreover, it prevents starvation by moving a process that waits too long for lower priority queue (`run_queue`) to the higher priority queue (`ready_queue`).

2.2 Memory Management

Question: What is the advantage and disadvantage of segmentation with paging?

Advantages of segmentation with paging:

- The memory usage reduction

- No internal fragmentation
- Page table size is limited by the segment size
- Segment table has only one entry corresponding to one actual segment
- It simplifies memory allocation

Disadvantages of segmentation with paging:

- Internal fragmentation
- External fragmentation
- The complexity level will be much higher as compare to paging
- Page tables need to be contiguously stored in the memory.

2.3 Put It All Together

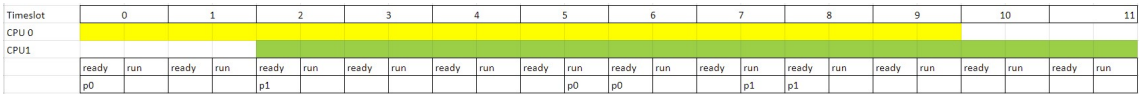
3 Scheduling

Draw Gantt diagram describing how processes are executed by the CPU

Depending on 4 files os_0, os_1, sched_0, sched_1 (input section), we have 4 Gantt charts.

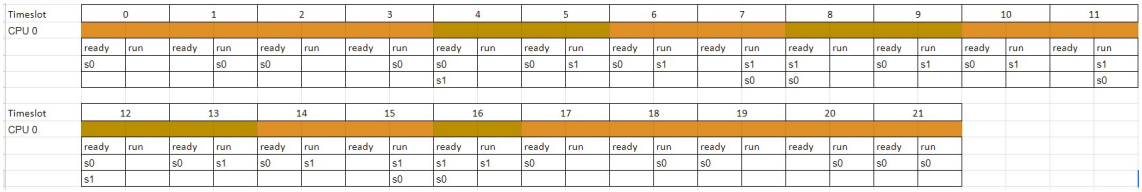
Gantt chart 1 from file os_0 :

os_0	[time slice	[No. CPU	[No. Processes]		[Process]	[Color]	[Init Time	[Priority]	[No. Instructions]
	6	2	2	Just 2 (p0 and p1)	p0		0	1	10
					p1		2	1	10



Gantt chart 2 from file sched_0 :

sched_0	[time slice	[No. CPU	[No. Processes]		[Process]	[Color]	[Init Time	[Priority]	[No. Instructions]
	2	1	2		s0		0	12	15
					s1		4	20	7



Gantt chart 3 from file sched_1 :

sched_1	[time slice	[No. CPU]	[No. Processes]		[Process]	[Color]	[Init Time	[Priority]	[No. Instructions]
	2	1	4		s0		0	12	15
					s1		4	20	7
					s2		6	20	12
					s3		7	7	11

Timeslot	0		1		2		3		4		5		6		7		8		9		10		11	
CPU 0																								
	ready	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*
	s0			s0	**s0**			s0	s0		s0	s1	s0	s1	s0	s1	**s0**	s1	s3	s1	**s3**	s1		s1
									s1				**s2**	s1	s3	s2	s3	s2		s2		s2		s2
																				s0		s0		s0
																								s3

Timeslot	**12**		13		14		15		16		17		18		19		**20**		21		22		23	
CPU 0																								
	ready	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*
	s1		s2	s1	**s2**	s1	s0	s1	s0	s1	s0	s1	**s0**	s1		s1	**s1**		s2	s1	**s2**	s1	s0	s1
	s2		s0		s0		s3	s2	**s3**	s2		s2		s2		s2	s2		s0		s0		s3	s2
	s0		s3		s3							s3		s3		s3	s0		s3		s3			
	s3															s0	s3							

Timeslot	24		25		26		27		**28**		29		30		31		24		25		26	
CPU 0																						
	ready	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*
	s0	s1	s0	s1	**s0**	s1		s1	**s1**	**s1**	**s2**		s3	s2	**s3**	s2	s0	s2	**s0**	s2		s2
	s3	s2		s2		s2		s2	s2		s3		s0		s0			s3		s3		s3
				s3		s3		s3	s0		s0											s0
								s0	s3													

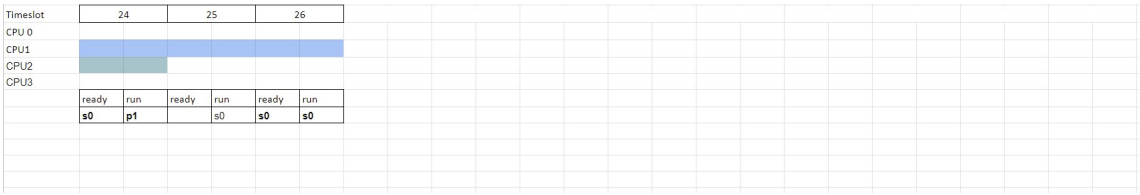
Timeslot	27		28		29		30		31		32		33		34		35		36	
CPU 0																				
	ready	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*
	s2		s3	s2	s3	s2	s3	s2	**s3**	s2		s2	**s2**		s3	**s2**	s3	**s0**	**s3**	**s3**
	s3		s0		**s0**			s0		s0		s3	s3		s0		**s0**			
	s0											s0	s0							

Gantt chart 4 from file os_1 :

os_1	[time slice	[No. CPU]	[No. Processes]		[Process]	[Color]	[Init Time	[Priority]	[No. Instructions]
	2	4	8		p0		1	1	10
					s3		2	7	11
					m1		4	1	8
					s2		6	20	12
					m0		7	1	7
					p1		9	1	10
					s0		11	12	15
					s1		16	20	7

Timeslot	0		1		2		3		4		5		6		7		8		9		10		11	
CPU 0																								
CPU1																								
CPU2																								
CPU3																								
	ready	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*
			p0		**s3**	p0	**p0**	s3	**m1**	p0	**p0**	m1	**m1**	p0	**p0**	s2	**m0**	s2	**s2**	m0	**p0**	m0	**p1**	m0
									s3			s3	**s3**		m0	s3		s3	**s3**		p1	s2	**s0**	s2
													s2			m1		m1	**m1**			s3		s3
																		p0	p0			m1		m1
																			p1					**p0**

Timeslot	12		13		14		15		16		17		18		19		20		21		22		23	
CPU 0																								
CPU1																								
CPU2																								
CPU3																								
	ready	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*	*ready*	*run*
	m0	s0	**m0**	s0	**s0**	m0	**p1**	m0	m0	p1	**m0**	p1	p1	m0	**p1**	m0	**m0**	p1	**p1**	s1	**s1**	p1	**p1**	s0
	s2	p1	**m1**	p1	p1	**m1**	**s3**	s2	**s2**			s2	**s2**			**s2**	**s1**	**m0**		s0	**s0**	**s1**		
	s3			s2	**s2**			s0	**s0**			s1	**s1**			s1	**s0**							
	m1			s3	s3			**s3**	**s1**			s0	**s0**			s0								



Timeslot	24		25		26	
CPU 0						
CPU1						
CPU2						
CPU3						
	ready	run	ready	run	ready	run
	s0	**p1**		s0	**s0**	**s0**

4 Memory

Show the status of RAM after each memory allocation and deallocation function call

Depending on 2 files m0 and m1 (input section), there are 2 processes m0 and m1.

4.1 Process m0

1 7

alloc 13535 0

alloc 1568 1

free 0

alloc 1386 2

alloc 4564 4

write 102 1 20

write 21 2 1000

1 page = 1024 byte

Register	Byte
reg 0	0
reg 1	14336
reg 2	2048
reg 3	
reg 4	5120
reg 5	
reg 6	
reg 7	
reg 8	
reg 9	

16		
15		1
14		
13		
12		
11		
10		
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		0
Page		Reg

After allocating ’13535 0’ and ’1568 1’

16		
15		1
14		
13		
12		
11		
10		
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		
Page		Reg

After ’free 0’

16		
15		1
14		
13		
12		
11		
10		
9		
8		
7		
6		
5		
4		
3		
2		4
1		
0		2
Page		Reg

After 'allocating '1386 2' and '4564 4'

4.2 Process m1

1 8

alloc 13535 0

alloc 1568 1

free 0

alloc 1386 2

alloc 4564 4

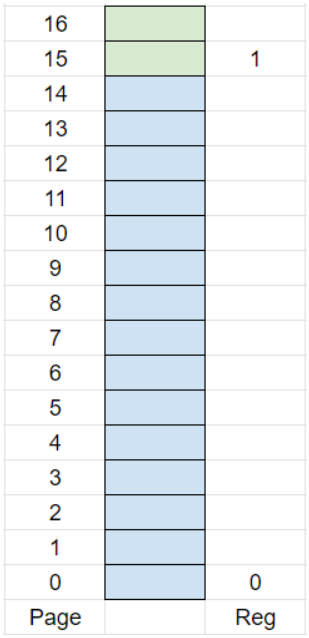
free 2

free 4

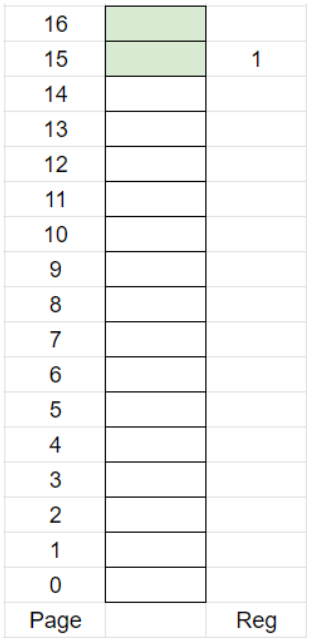
free 1

1 page = 1024 byte

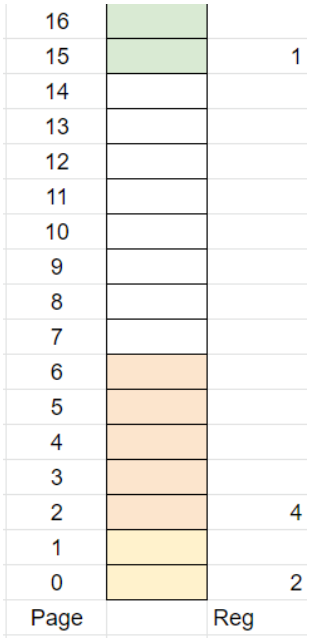
Register	Byte
reg 0	0
reg 1	14336
reg 2	2048
reg 3	
reg 4	5120
reg 5	
reg 6	
reg 7	
reg 8	
reg 9	



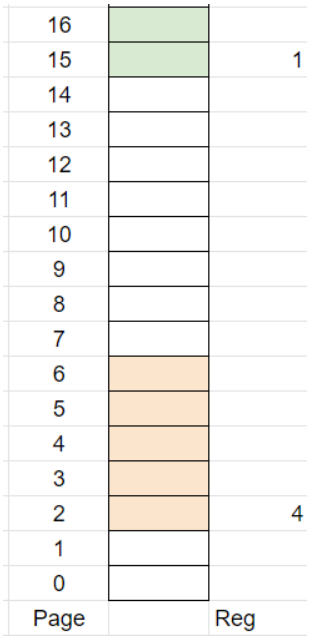
After allocating ’13535 0’ and ’1568 1’



After ’free 0’



After ’allocating ’1386 2’ and ’4564 4’



After ’free 2’

16		
15		1
14		
13		
12		
11		
10		
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		
Page		Reg

After 'free 4'

16		
15		
14		
13		
12		
11		
10		
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		
Page		Reg

After 'free 1'

5 Overall

Interpreting the results of simulation

This is the output after finishing coding section

```
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
       003e8: 15
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
       03814: 66
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
```

Comparing with m0 file(output section), there are differences.

```
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
   003e8: 14
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
   03814: 64
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
```

There is a mistake at line 2 '**003e8: 14**' of file m0 because of depending on command of process m0.

1 7

alloc 13535 0

alloc 1568 1

free 0

alloc 1386 2

alloc 4564 4

write 102 1 20

write 21 2 1000

At line 7, the command **'write 102 1 20'** the output will be 15(hex)

6 Member Workload

No.	Full Name	Student ID	Problem	Evaluation
1	Duong Gia An	1952163	- Funtion enqueue(), dequeue() and get_proc() - Scheduling	100%
2	Thieu Quang Trung	1953051	- Function get_page_table() and translate() - 2 questions on report	100%
3	Tran Ngoc Minh Diep	1952610	- Funtion alloc_mem() and free_mem() - Memory status	100%

*** Evalution on Completion of Assigned task**

References

[Web] Segmented Paging https://www.javatpoint.com/os-segmented-paging

[Web] Segmented Paging https://www.geeksforgeeks.org/advantages-and-disadvantages-of-various-cpu-scheduling-algorithms/