VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEM (CO2017)

Assignment

SIMPLE OPERATING SYSTEM

Advisors:
Lê Thanh Vân
Trần Việt Toản

Group members:

Nguyễn Luật Gia Khôi	1952079	CC01
Phạm Bùi Minh Huân	1952056	CC01
Phạm Khánh Trình	1953044	CC02

HO CHI MINH CITY, MAY 2021

Contents

1 Answers to questions in implementation section

Question 1: What are the advantages of using priority feedback queue in comparison with other scheduling algorithms you have learned?

In priority feedback queue, processes can move between the queues. The system keeps analyzing the behavior (time of execution) of processes and according to which it changes its priority. Thus it yields the following advantages:

- This is a flexible scheduling algorithm.
- This scheduling algorithm allows different processes to move between different queues.
- A process that waits too long in a lower priority queue may be moved to a higher priority queue which helps preventing starvation.
- Minimize response time without a prior knowledge of job length.

Question 2: What are the advantages and disadvantage of segmentation with paging?

Segmented Paging Advantages

- No external fragmentation.
- Reduced memory requirements as no. of pages limited to segment size.
- Page table size is smaller just like segmented paging,
- Similar to segmented paging, the entire segment need not be swapped out.

Segmented Paging Disadvantages

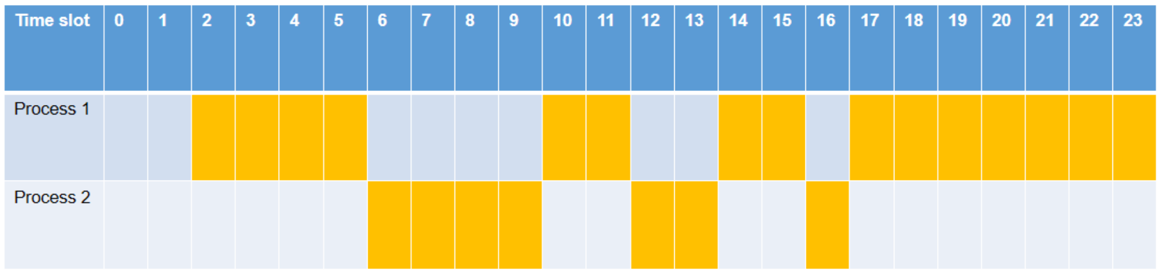
- Internal fragmentation remains a problem.
- Hardware is more complicated due to segmented paging.
- Extra level of paging at first stage adds to the delay in memory access.

2 Interpreting results of running tests

2.1 Scheduling

2.1.a) sched_0

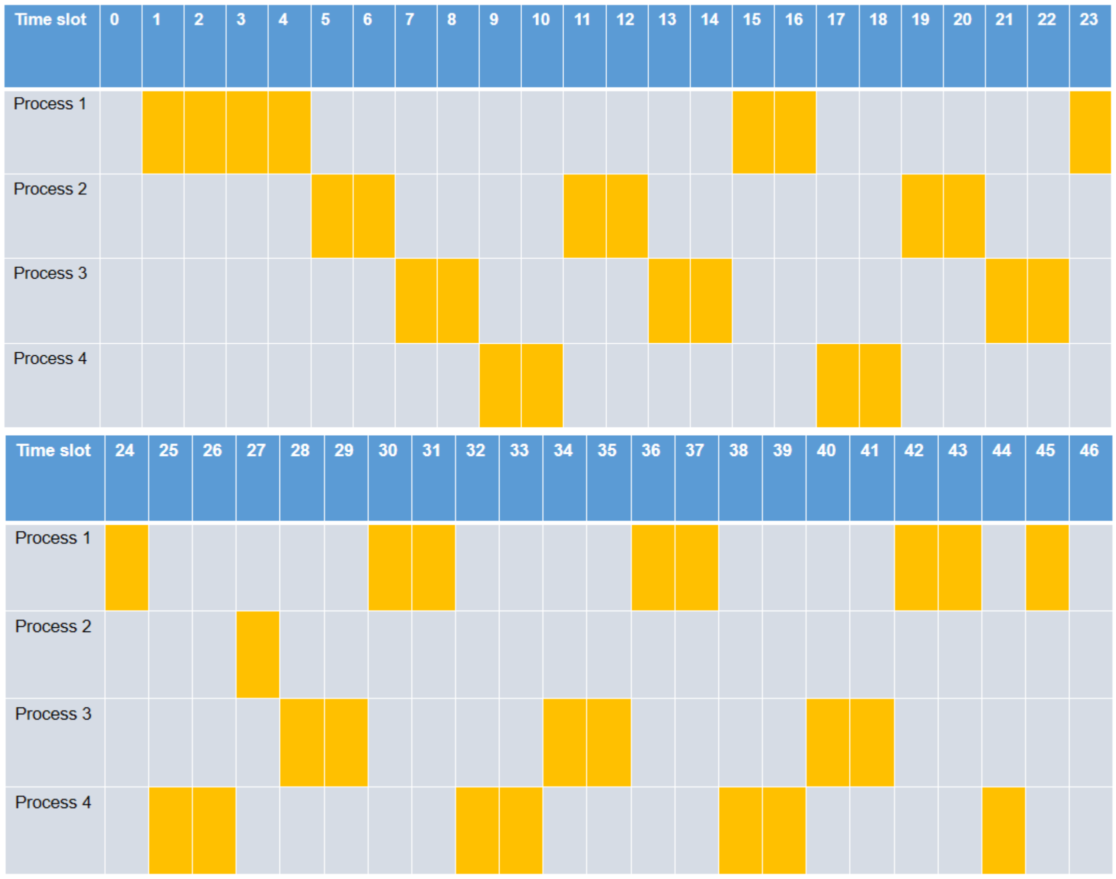
The scheduling of 2 processes are illustrated in the following Gantt chart. A yellow cell means that the corresponding process is currently running at a given time slot.



Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Process 1			🟨	🟨	🟨	🟨					🟨	🟨			🟨	🟨		🟨	🟨	🟨	🟨	🟨	🟨	🟨
Process 2							🟨	🟨	🟨	🟨			🟨	🟨			🟨							

2.1.b) sched_1

The scheduling of 4 processes are illustrated in the following Gantt charts. A yellow cell means that the corresponding process is currently running at a given time slot.



Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Process 1		🟨	🟨	🟨	🟨											🟨	🟨							🟨
Process 2						🟨	🟨					🟨	🟨							🟨	🟨			
Process 3								🟨	🟨					🟨	🟨							🟨	🟨	
Process 4										🟨	🟨							🟨	🟨					

Time slot	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
Process 1	🟨						🟨	🟨					🟨	🟨					🟨	🟨		🟨	
Process 2				🟨																			
Process 3					🟨	🟨					🟨	🟨					🟨	🟨					
Process 4		🟨	🟨						🟨	🟨					🟨	🟨					🟨		

2.2 Memory

1. **m_0**:
   - Content of m_0:

   - 1 7

- alloc 13535 0
- alloc 1568 1
- free 0
- alloc 1386 2
- alloc 4564 4
- write 102 1 20
- write 21 2 1000

- Status of RAM:

- After alloc 13535 0:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
  - 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
  - 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
  - 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
  - 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
  - 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
  - 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
  - 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
  - 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
  - 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
  - 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
  - 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
  - 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)

  Explanation:
  To allocate 13535 bytes, we need $\lceil 13535/1024 = 14 \rceil$ page frames. The virtual address of the first byte, whose corresponding physical address is 0x00000, is then stored into register 0 of the process.
- After alloc 1568 1:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
  - 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
  - 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
  - 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
  - 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
  - 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
  - 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
  - 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
  - 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
  - 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
  - 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
  - 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)

  - 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
  - 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation:
  To allocate 1568 bytes, we need $\lceil 1568/1024 \rceil = 2$ page frames. The virtual address of the first byte, whose corresponding physical address is 0x03800, is then stored into register 1 of the process.

- After free 0:
  - 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation
  The OS now deallocates the amount of allocated memory whose first byte is stored in register 0. In this case, register 0 stored the address of the first byte of an allocated memory of 14 page frames.

- After alloc 1386 2:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
  - 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation To allocate 1386 bytes, we need page frames. The virtual address of the first byte, whose corresponding physical address is 0x00000, is then stored into register 2 of the process.

- After alloc 4564 4:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 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)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation To allocate 4564 bytes, we need $\lceil 4564/1024 \rceil = 5$ page frames. The virtual address of the first byte, whose corresponding physical address is 0x00800, is then stored into register 4 of the process.

2. **m_1**
   - Content of m_1:
   - 1 8
   - alloc 13535 0
   - alloc 1568 1
   - free 0

- alloc 1386 2
- alloc 4564 4
- free 2
- free 4
- free 1

- Status of RAM:

- After alloc 13535 0:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
  - 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
  - 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
  - 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
  - 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
  - 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
  - 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
  - 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
  - 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
  - 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
  - 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
  - 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
  - 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)

  Explanation To allocate 13535 bytes, we need $\lceil 13535/1024 \rceil = 14$ page frames. The virtual address of the first byte, whose corresponding physical address is 0x00000, is then stored into register 0 of the process.
- After alloc 1568 1:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
  - 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
  - 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
  - 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
  - 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
  - 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
  - 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
  - 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
  - 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
  - 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
  - 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
  - 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
  - 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
  - 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)

  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation To allocate 1568 bytes, we need $\lceil 1568/1024 \rceil = 2$ page frames. The virtual address of the first byte, whose corresponding physical address is 0x03800, is then stored into register 1 of the process.

- After free 0:
  - 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation The OS now deallocates the amount of allocated memory whose first byte is stored in register 0. In this case, register 0 stored the address of the first byte of an allocated memory of 14 page frames.

- After alloc 1386 2:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
  - 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation To allocate 1386 bytes, we need $\lceil 1386/1024 \rceil = 2$ page frames. The virtual address of the first byte, whose corresponding physical address is 0x00000, is then stored into register 2 of the process.

- After alloc 4564 4:
  - 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
  - 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)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation To allocate 4564 bytes, we need $\lceil 4564/1024 \rceil = 5$ page frames. The virtual address of the first byte, whose corresponding physical address is 0x00800, is then stored into register 4 of the process.

- After free 2:
  - 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)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation The OS now deallocates the amount of allocated memory whose first byte is stored in register 2. In this case, register 2 stored the address of the first byte of an allocated memory of 2 page frames.

- After free 4:
  - 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
  - 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)

  Explanation The OS now deallocates the amount of allocated memory whose first byte is stored in register 4. In this case, register 4 stored the address of the first byte of an allocated memory of 5 page frames.
- After free 1:
  * Empty memory *
  Explanation The OS now deallocates the amount of allocated memory whose first byte is stored in register 1. In this case, register 1 stored the address of the first byte of an allocated memory of 2 page frames. Therefore, the memory is now completely empty.

2.3 Overall

2.3.a) os_0

Configuration:

```
6 2 4
0 p0
2 p1
```

Comment: According to the configuration, the os will run 4 processes on 2 CPU simultaneously. There is one process specified in p0 and three processes specified p1.
Ouput:

```
Time slot   0
        Loaded a process at input/proc/p0, PID: 1
        CPU 1: Dispatched process  1
Time slot   1
        Loaded a process at input/proc/p1, PID: 2
Time slot   2
        CPU 0: Dispatched process  2
Time slot   3
        Loaded a process at input/proc/p1, PID: 3
Time slot   4
        Loaded a process at input/proc/p1, PID: 4
Time slot   5
Time slot   6
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  3
Time slot   7
Time slot   8
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  4
Time slot   9
Time slot  10
Time slot  11
Time slot  12
```

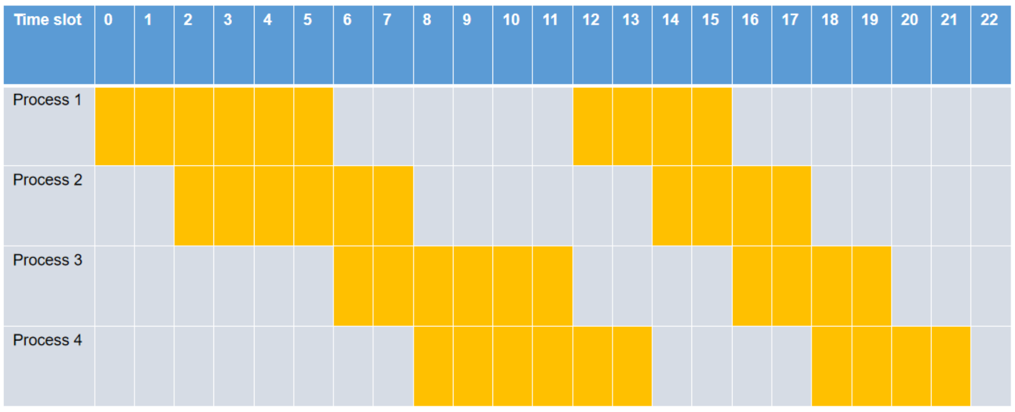
```
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  1
Time slot  13
Time slot  14
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  15
Time slot  16
        CPU 1: Processed  1 has finished
        CPU 1: Dispatched process  3
Time slot  17
Time slot  18
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  4
Time slot  19
Time slot  20
        CPU 1: Processed  3 has finished
        CPU 1 stopped
Time slot  21
Time slot  22
        CPU 0: Processed  4 has finished
        CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 03 (idx 000, nxt: 001)
001: 00400-007ff - PID: 03 (idx 001, nxt: 002)
002: 00800-00bff - PID: 03 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 03 (idx 003, nxt: -01)
004: 01000-013ff - PID: 04 (idx 000, nxt: 005)
005: 01400-017ff - PID: 04 (idx 001, nxt: 006)
006: 01800-01bff - PID: 04 (idx 002, nxt: 012)
007: 01c00-01fff - PID: 02 (idx 000, nxt: 008)
008: 02000-023ff - PID: 02 (idx 001, nxt: 009)
009: 02400-027ff - PID: 02 (idx 002, nxt: 010)
        025e7: 0a
010: 02800-02bff - PID: 02 (idx 003, nxt: 011)
011: 02c00-02fff - PID: 02 (idx 004, nxt: -01)
012: 03000-033ff - PID: 04 (idx 003, nxt: -01)
014: 03800-03bff - PID: 03 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 03 (idx 001, nxt: 016)
016: 04000-043ff - PID: 03 (idx 002, nxt: 017)
        041e7: 0a
017: 04400-047ff - PID: 03 (idx 003, nxt: 018)
018: 04800-04bff - PID: 03 (idx 004, nxt: -01)
023: 05c00-05fff - PID: 02 (idx 000, nxt: 024)
024: 06000-063ff - PID: 02 (idx 001, nxt: 025)
025: 06400-067ff - PID: 02 (idx 002, nxt: 026)
026: 06800-06bff - PID: 02 (idx 003, nxt: -01)
047: 0bc00-0bfff - PID: 01 (idx 000, nxt: -01)
```

```
        0bc14: 64
057: 0e400-0e7ff - PID: 04 (idx 000, nxt: 058)
058: 0e800-0ebff - PID: 04 (idx 001, nxt: 059)
059: 0ec00-0efff - PID: 04 (idx 002, nxt: 060)
        0ede7: 0a
060: 0f000-0f3ff - PID: 04 (idx 003, nxt: 061)
061: 0f400-0f7ff - PID: 04 (idx 004, nxt: -01)
```

Remark

- First, let us draw a Gantt chart illustrating the scheduling and executions of 4 processes:



Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Process 1	🟨	🟨	🟨	🟨	🟨	🟨							🟨	🟨	🟨	🟨							
Process 2			🟨	🟨	🟨	🟨	🟨	🟨							🟨	🟨	🟨	🟨					
Process 3							🟨	🟨	🟨	🟨	🟨	🟨					🟨	🟨	🟨	🟨			
Process 4									🟨	🟨	🟨	🟨	🟨	🟨					🟨	🟨	🟨	🟨	

- It can be seen that there were at most 2 instructions executed at the same time, which matches the configuration of having 2 CPUs running simultaneously. Moreover, the number of instructions executed (the number of yellow cells) in each process also matches the number of instructions specified in the process discription files. Now let's talk about the memory content.
- Process 1 executed the instructions specified in p0 and process 2, 3, 4 executed the instructions specified in p1.
- As a matter of fact, process 1 allocated 8 pages and deallocated 7 pages so that the final memory content contained only 1 page allocated for process 1.
- Processes 2, 3, and 4 allocated 50 pages and deallocated 41 pages each. As the result, each process got 9 pages allocated divided into 2 blocks (5 and 4 pages respectively).
- Therefore, the number of allocated pages after running all the instructions were $8 - 7 + (50 - 41) * 3 = 28$.
- In addition, process 1 wrote 100 (which is 64 in hexadecimal) and processes 2, 3, and 4 wrote 10 (which is 0a in hexadecimal) into the memory.

We can also represent these remarks as a table for a more compact view

Process (specification file)	#pages allocated	#pages deallocated	#pages remain	#blocks divided	Written
1 (p0)	8	7	1	1	0x64
2 (p1)	50	41	9	2 (5 and 4 pages respectively)	0x0a
3 (p1)	50	41	9	2 (5 and 4 pages respectively)	0x0a
4 (p1)	50	41	9	2 (5 and 4 pages respectively)	0x0a
Total	158	130	28	7	0x64, 0x0a, 0x0a, 0x0a

2.3.b) os_1

Configuration:

```
2 4 8
1 p0
2 s3
4 m1
6 s2
7 m0
9 p1
11 s0
16 s1
```

Comment: According to the configuration, the os will run 8 processes on 4 CPU simultaneously. The instructions of the 8 processes specified in the files p0, s3, m1, s2, m0, p1, s0, and s1.
Ouput:

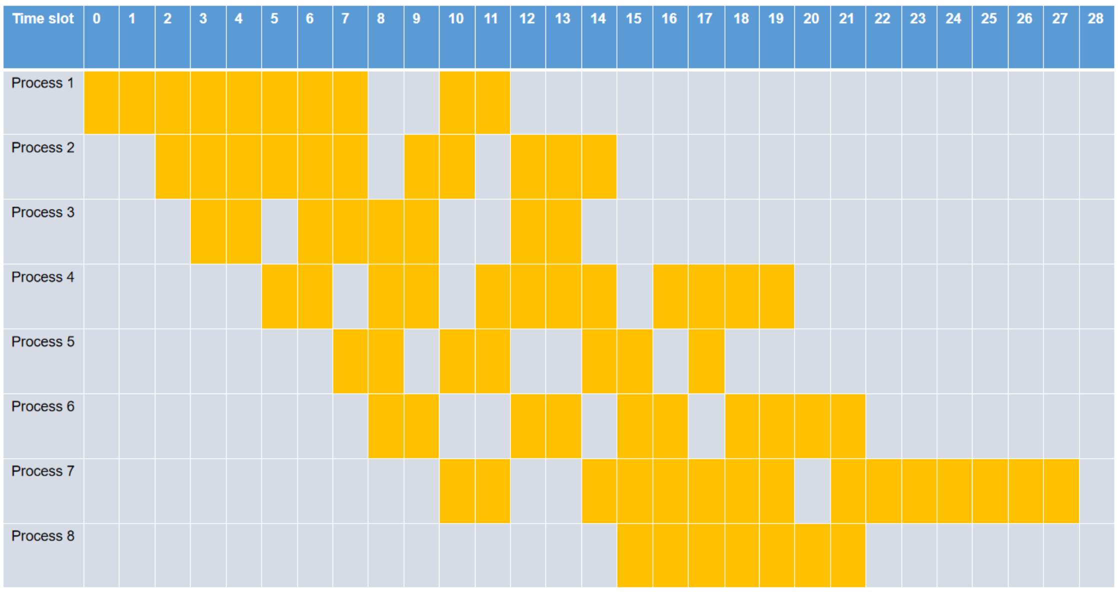
```
Time slot   0
        Loaded a process at input/proc/p0, PID: 1
        CPU 3: Dispatched process  1
Time slot   1
Time slot   2
        Loaded a process at input/proc/s3, PID: 2
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  2
        CPU 2: Dispatched process  1
Time slot   3
        Loaded a process at input/proc/m1, PID: 3
        CPU 1: Dispatched process  3
Time slot   4
        CPU 3: Put process  2 to run queue
        CPU 3: Dispatched process  2
        CPU 2: Put process  1 to run queue
        CPU 2: Dispatched process  1
```

```
Time slot   5
        Loaded a process at input/proc/s2, PID: 4
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  4
Time slot   6
        CPU 0: Dispatched process  3
        CPU 3: Put process  2 to run queue
        CPU 3: Dispatched process  2
        CPU 2: Put process  1 to run queue
        CPU 2: Dispatched process  1
Time slot   7
        Loaded a process at input/proc/m0, PID: 5
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  5
Time slot   8
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
        Loaded a process at input/proc/p1, PID: 6
        CPU 3: Put process  2 to run queue
        CPU 3: Dispatched process  3
        CPU 2: Put process  1 to run queue
        CPU 2: Dispatched process  6
Time slot   9
        CPU 1: Put process  5 to run queue
        CPU 1: Dispatched process  2
Time slot  10
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/s0, PID: 7
        CPU 3: Put process  3 to run queue
        CPU 3: Dispatched process  7
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  5
Time slot  11
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  4
Time slot  12
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  2
        CPU 3: Put process  7 to run queue
        CPU 3: Dispatched process  3
        CPU 2: Put process  5 to run queue
        CPU 2: Dispatched process  6
Time slot  13
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
Time slot  14
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  7
        CPU 3: Processed  3 has finished
        CPU 3: Dispatched process  5
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  2
Time slot  15
        Loaded a process at input/proc/s1, PID: 8
        CPU 2: Processed  2 has finished
        CPU 2: Dispatched process  8
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  6
Time slot  16
        CPU 0: Put process  7 to run queue
        CPU 0: Dispatched process  4
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  7
Time slot  17
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 1: Put process  6 to run queue
        CPU 1: Dispatched process  5
Time slot  18
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
        CPU 3: Put process  7 to run queue
        CPU 3: Dispatched process  6
        CPU 1: Processed  5 has finished
        CPU 1: Dispatched process  7
Time slot  19
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
Time slot  20
        CPU 0: Processed  4 has finished
        CPU 0 stopped
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot  21
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
Time slot  22
        CPU 3: Processed  6 has finished
        CPU 3 stopped
        CPU 2: Processed  8 has finished
Time slot  23
        CPU 2 stopped
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  24
Time slot  25
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  26
Time slot  27
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  28
        CPU 1: Processed  7 has finished
        CPU 1 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 06 (idx 000, nxt: 001)
001: 00400-007ff - PID: 06 (idx 001, nxt: 031)
002: 00800-00bff - PID: 05 (idx 000, nxt: 003)
        00be8: 15
003: 00c00-00fff - PID: 05 (idx 001, nxt: -01)
004: 01000-013ff - PID: 05 (idx 000, nxt: 005)
005: 01400-017ff - PID: 05 (idx 001, nxt: 006)
006: 01800-01bff - PID: 05 (idx 002, nxt: 007)
007: 01c00-01fff - PID: 05 (idx 003, nxt: 008)
008: 02000-023ff - PID: 05 (idx 004, nxt: -01)
013: 03400-037ff - PID: 06 (idx 000, nxt: 014)
014: 03800-03bff - PID: 06 (idx 001, nxt: 015)
015: 03c00-03fff - PID: 06 (idx 002, nxt: 016)
016: 04000-043ff - PID: 06 (idx 003, nxt: -01)
019: 04c00-04fff - PID: 01 (idx 000, nxt: -01)
        04c14: 64
029: 07400-077ff - PID: 05 (idx 000, nxt: 030)
        07414: 66
030: 07800-07bff - PID: 05 (idx 001, nxt: -01)
031: 07c00-07fff - PID: 06 (idx 002, nxt: 032)
        07de7: 0a
032: 08000-083ff - PID: 06 (idx 003, nxt: 033)
033: 08400-087ff - PID: 06 (idx 004, nxt: -01)
```

Remark:

- First, let us draw a Gantt chart illustrating the scheduling and executions of 4 processes:



Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Process 1	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨			🟨	🟨																	
Process 2			🟨	🟨	🟨	🟨	🟨	🟨		🟨	🟨		🟨	🟨	🟨														
Process 3				🟨	🟨		🟨	🟨	🟨	🟨			🟨	🟨															
Process 4						🟨	🟨		🟨	🟨		🟨	🟨	🟨	🟨		🟨	🟨	🟨	🟨									
Process 5								🟨	🟨		🟨	🟨			🟨	🟨		🟨											
Process 6									🟨	🟨			🟨	🟨		🟨	🟨		🟨	🟨	🟨	🟨							
Process 7											🟨	🟨			🟨	🟨	🟨	🟨	🟨	🟨		🟨	🟨	🟨	🟨	🟨	🟨	🟨	
Process 8																🟨	🟨	🟨	🟨	🟨	🟨	🟨							

- It can be seen that there were at most 4 instructions executed at the same time, which matches the configuration of having 4 CPUs running simultaneously. Moreover, the number of instructions executed (the number of yellow cells) in each process also matches the number of instructions specified in the process discription files. Now let's talk about the memory content.
- Since os_1 has significantly more tasks than os_0, the memory allocation of processes in os_1 will be illustrated in table format for a more compact and eligible representation

Process (specification file)	**#pages allocated**	**#pages deallocated**	**#pages remain**	**#blocks divided**	**Written**
1 (p0)	8	7	1	1	0x64
2 (s3)	0	0	0	0	None
3 (m1)	23	23	0	0	None
4 (s2)	0	0	0	0	None
5 (m0)	23	14	9	3 (2, 2, and 5 pages respectively)	0x66, 0x15
6 (p1)	50	41	9	2 (5 and 4 respectively)	0x0a
7 (s0)	0	0	0	0	None
8 (s1)	0	0	0	0	None
Total	104	85	19	6	0x64, 0x66, 0x15, 0x0a

2.3.c) Conclusion

In summary, both of the executions of os_0 and os_1 returned results that matched expected statuses of the memory, which proves the correctness of our implementation.