HCMC UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & ENGINEERING

OPERATING SYSTEM ASSIGNMENT

Simple Operating System

Student: Nguyễn Ngọc Phú – 2114417

MAY 17, 2023

Table of Contents

1 Preface

1. **Source code**: https://github.com/ngyngcphu/Simple-Operating-System
2. **Presentation Slides** (In progress...) : https://ngyngcphu.github.io/Simple-Operating-System/

2 Scheduler

2.1 Question

Question: What is the advantage of using priority queue in comparison with other scheduling algorithms you have learned?
Answer:
Disadvantages of other scheduling algorithms:

1. **First Come First Served (FCFS):**
   - Because of using the nonpreemptive strategy, if a process starts, the CPU executes the process until it finishes.
   - If a process has a long CPU burst, the processes behind (probably processes with a short CPU burst) in the queue have to wait a long time -> It's not fair to the processes.
2. **Shortest Job First (SJF):**
   - It is necessary to estimate the process's next CPU time in advance when this is often not possible in practice.
   - Processes with long CPU bursts have more waiting time or delay indefinitely when multiple processes with short CPU bursts simultaneously enter queue ->**starvation**.
3. **Round Robin (RR):**
   - If the time quantum is too large: **RR->FCFS**.
   - If the time quantum is too small, the context switch of the CPU will increase greatly causing overhead, reduce CPU utilization.
4. **Priority Scheduling (PS-normal):**
   - Must have scheduler with processes with equal priority.
   - Processes with lower priority may not have a chance to execute -> **starvation**.

Advantages of using **Multi-Level Queue (MFQ)** scheduling algorithms:

1. **Response time:** Because processes with the same priority are placed in the same queue, high-priority processes are processed first.
2. **Fairly:** Ensuring fairness for all processes in the same queue is executed by applying Round Robin "style".
3. **Not starvation:** Each queue have only fixed slot to use the CPU (**MAX_PRIO - prio**) and when it is used up, the system must change the resource to the other process in the next queue.

2.2 Gantt Diagram

1. **TEST 0**
   Input: /input/sched
   Output: /output/sched.output

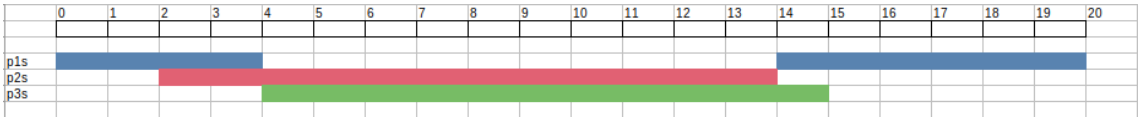

Figure 1: *Gantt scheme: 2 CPU executing 3 processes - test 0*

2. **TEST 1**
   Input: /input/sched_0
   Output: /output/sched_0.output

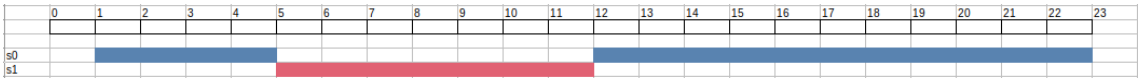

Figure 2: *Gantt scheme: 1 CPU executing 2 processes - test 1*

3. **TEST 2**
   Input: /input/sched_1
   Output: /output/sched_1.output

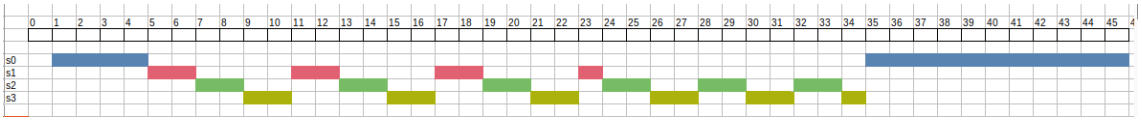

Figure 3: *Gantt scheme: 1 CPU executing 4 processes - test 2*

2.3 Implementation

1. **Priority Queue**

   - *enqueue()*: Put a new process at the end of the queue.
   - *dequeue()*: Get a process at the begin of the queue.

```
void enqueue(struct queue_t *q, struct pcb_t *proc)
{
        /* TODO: put a new process to queue [q] */
        if (q == NULL)
        {
                perror("Queue is NULL !\n");
                exit(1);
        }
        if (q->size == MAX_QUEUE_SIZE)
        {
```

```
                perror("Queue is full !\n");
                exit(1);
        }
        q->proc[q->size] = proc;
        q->size++;
}

struct pcb_t *dequeue(struct queue_t *q)
{
        /* TODO: return a pcb whose prioprity is the highest
         * in the queue [q] and remember to remove it from q
         * */
        if (q == NULL || q->size == 0)
        {
                return NULL;
        }
        struct pcb_t *temp = q->proc[0];

#ifdef MLQ_SCHED
        int length = q->size - 1;
        for (int i = 0; i < length; ++i)
        {
                q->proc[i] = q->proc[i + 1];
        }
        q->proc[length] = NULL;
        q->size--;
        return temp;
#else
        .
        .
        .
#endif
}
```

2. **Scheduler**

- *get_mlq_proc()*: get a process from PRIORITY [ready_queue]

```
struct pcb_t *get_mlq_proc(void)
{
 struct pcb_t *proc = NULL;
 /*TODO: get a process from PRIORITY [ready_queue].
  * Remember to use lock to protect the queue. */
 pthread_mutex_lock(&queue_lock);
 int i;
 for (i = 0; i < MAX_PRIO; ++i)
 {
    if (empty(&mlq_ready_queue[i]) || slot[i] == 0)
    {
       slot[i] = MAX_PRIO - i;
       continue;
    }
    proc = dequeue(&mlq_ready_queue[i]);
```

```
        slot[i]--;
        break;
    }
    pthread_mutex_unlock(&queue_lock);
    return proc;
}
```

3 Memory Management

3.1 Question

1. **Question 1:** In this simple OS, we implement a design of multiple memory segments or memory areas in source code declaration. What is the advantage of the proposed design of multiple segments?
   Answer:
   Advantage of the proposed design of multiple segments:

   - **Memory Organization**: By dividing memory into multiple segments, you can have a more structured and organized memory layout. Each segment can be dedicated to a specific purpose or type of data (data segment, code segment, stack segment, heap segment,...), making it easier to manage and access information.
   - **Efficient Memory Allocation**: By dividing memory into segments, you can optimize memory allocation. Different segments can be allocated based on specific requirements, such as stack segment for storing local variables and function calls, heap segment for dynamic memory allocation, and code segment for executable instructions. This segmentation can improve memory utilization and reduce fragmentation.
   - **Context Switching**: When performing a context switch between processes or threads, having segmented memory can simplify the process. Only the necessary segments need to be swapped, rather than the entire memory space, which can be more efficient and faster.

2. **Question 2:** What will happen if we divide the address to more than 2-levels in the paging memory management system?
   Answer:
   Here's what would happen if we introduce more than 2 levels in the paging system:

   - **Reduced Memory Overhead**: With more levels, each individual page table becomes smaller, reducing the memory overhead required to store the page tables. This is especially beneficial in systems with large address spaces.
   - **Improved Page Table Access Time**: In a multi-level paging scheme, the page table is divided into multiple smaller tables. This reduces the size of each table, making it faster to access and traverse during the address translation process. It can help mitigate the increase in access time that would occur with a single large page table.
   - **Flexible Page Table Allocation**: With multiple levels, it becomes easier to allocate and manage page tables dynamically. You can allocate page tables on-demand as required by the processes, reducing memory wastage.
   - **Efficient Use of Memory**: A multi-level paging scheme allows for efficient use of memory resources. Instead of having one large page table that needs to be allocated contiguously, you can allocate smaller page tables in a fragmented manner, utilizing available memory more effectively.

3. **Question 3:** What is the advantage and disadvantage of segmentation with paging?
   Answer:
   Advantage:

   - Optimize memory, use memory effectively.
   - Simple allocation of non-consecutive memory.

- Fix page size too large by paging in each segment.
- No external fragmentation occurs.

Disadvantage:

- Internal fragmentation still occurs.
- Combining segmentation and paging increases the complexity of the memory management system.
- When using segmentation with paging, each segment may contain multiple pages. As a result, the number of entries in the page tables can increase, leading to larger page tables.

4. **Question 4:** What will happen if the synchronization is not handled in your simple OS? Illustrate by example the problem of your simple OS if you have any
Answer:
If synchronization is not handled in a simple operating system, it can lead to various problems, including race conditions, data inconsistency, and deadlock.
Let's consider a simple example to illustrate these problems. Suppose our simple OS allows multiple processes to access and modify a shared variable simultaneously without any synchronization mechanism in place:
Process A and Process B are two concurrent processes that increment and decrement a shared variable count. Both processes start with count initialized to 0.
Process A executes the following code: $count = count + 1$
Process B executes the following code: $count = count - 1$
Without synchronization, the following sequence of events can occur:

   - Process A reads the current value of count (0) into a register.
   - Process B reads the current value of count (0) into a register.
   - Process A increments its local register value by 1 (0 + 1 = 1).
   - Process B decrements its local register value by 1 (0 - 1 = -1).
   - Process A writes its updated value (1) back to count.
   - Process B writes its updated value (-1) back to count.

   In this scenario, both processes have executed their operations concurrently without any synchronization. As a result, the final value of count is incorrect (-1 instead of 0). This problem is known as a race condition, where the output of the program depends on the timing and interleaving of concurrent operations.
   Additionally, without proper synchronization mechanisms, there is a risk of deadlock. Deadlock can occur when two or more processes are waiting indefinitely for each other to release resources. If there are shared resources that processes need exclusive access to, without proper synchronization, it's possible for a situation to arise where multiple processes are waiting for a resource that will never be released.

3.2 Show the status of RAM

Below is the result of the logging process after each allocation, deallocation, reading and writing command in the program in a testcase.
Input: /input/os_1_singleCPU_mlq
Output: /output/os_1_singleCPU_mlq

```
Time slot  0
ld_routine
Time slot  1
  Loaded a process at input/proc/s4, PID: 1 PRIO: 4
Time slot  2
  CPU 0: Dispatched process 1
  Loaded a process at input/proc/s3, PID: 2 PRIO: 3
Time slot  3
Time slot  4
  CPU 0: Put process 1 to run queue
  Loaded a process at input/proc/m1s, PID: 3 PRIO: 2
  CPU 0: Dispatched process 2
Time slot  5
Time slot  6
  CPU 0: Put process 2 to run queue
  CPU 0: Dispatched process 3
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
  Loaded a process at input/proc/s2, PID: 4 PRIO: 3
Time slot  7
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
  Loaded a process at input/proc/m0s, PID: 5 PRIO: 3
Time slot  8
  CPU 0: Put process 3 to run queue
  CPU 0: Dispatched process 3
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region=0
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
Time slot  9
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=2 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
```

```
Page Number: 1 -> Frame Number: 0
================================================================
  Loaded a process at input/proc/p1s, PID: 6 PRIO: 2
Time slot 10
  CPU 0: Put process 3 to run queue
  CPU 0: Dispatched process 6
Time slot 11
  Loaded a process at input/proc/s0, PID: 7 PRIO: 1
Time slot 12
  CPU 0: Put process 6 to run queue
  CPU 0: Dispatched process 7
Time slot 13
Time slot 14
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
Time slot 15
Time slot 16
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
  Loaded a process at input/proc/s1, PID: 8 PRIO: 0
Time slot 17
Time slot 18
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 8
Time slot 19
Time slot 20
  CPU 0: Put process 8 to run queue
  CPU 0: Dispatched process 8
Time slot 21
Time slot 22
  CPU 0: Put process 8 to run queue
  CPU 0: Dispatched process 8
Time slot 23
Time slot 24
  CPU 0: Put process 8 to run queue
  CPU 0: Dispatched process 8
Time slot 25
  CPU 0: Processed 8 has finished
  CPU 0: Dispatched process 7
Time slot 26
Time slot 27
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
Time slot 28
Time slot 29
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
Time slot 30
Time slot 31
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
Time slot 32
Time slot 33
```

```
  CPU 0: Put process 7 to run queue
  CPU 0: Dispatched process 7
Time slot 34
  CPU 0: Processed 7 has finished
  CPU 0: Dispatched process 3
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region=2
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
Time slot 35
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region=1
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
Time slot 36
  CPU 0: Processed 3 has finished
  CPU 0: Dispatched process 6
Time slot 37
Time slot 38
  CPU 0: Put process 6 to run queue
  CPU 0: Dispatched process 6
Time slot 39
Time slot 40
  CPU 0: Put process 6 to run queue
  CPU 0: Dispatched process 6
Time slot 41
Time slot 42
  CPU 0: Put process 6 to run queue
  CPU 0: Dispatched process 6
Time slot 43
Time slot 44
  CPU 0: Processed 6 has finished
  CPU 0: Dispatched process 2
Time slot 45
Time slot 46
  CPU 0: Put process 2 to run queue
  CPU 0: Dispatched process 4
Time slot 47
Time slot 48
  CPU 0: Put process 4 to run queue
  CPU 0: Dispatched process 5
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region=0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000003
```

```
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
Time slot 49
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region=1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
Time slot 50
  CPU 0: Put process 5 to run queue
  CPU 0: Dispatched process 2
Time slot 51
Time slot 52
  CPU 0: Put process 2 to run queue
  CPU 0: Dispatched process 4
Time slot 53
Time slot 54
  CPU 0: Put process 4 to run queue
  CPU 0: Dispatched process 5
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=5 - Region=0
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
Time slot 55
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region=2 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
Time slot 56
  CPU 0: Put process 5 to run queue
  CPU 0: Dispatched process 2
Time slot 57
Time slot 58
  CPU 0: Put process 2 to run queue
  CPU 0: Dispatched process 4
Time slot 59
Time slot 60
  CPU 0: Put process 4 to run queue
  CPU 0: Dispatched process 5
===== PHYSICAL MEMORY AFTER WRITING =====
```

```
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000240: 102
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot 61
===== PHYSICAL MEMORY AFTER WRITING =====
write region=2 offset=1000 value=1
print_pgtbl: 0 - 512
00000000: c0000000
00000004: 80000002
Page Number: 0 -> Frame Number: 0
Page Number: 1 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000240: 102
BYTE 000003e8: 1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
Time slot 62
  CPU 0: Processed 5 has finished
  CPU 0: Dispatched process 2
Time slot 63
Time slot 64
  CPU 0: Put process 2 to run queue
  CPU 0: Dispatched process 4
Time slot 65
Time slot 66
  CPU 0: Put process 4 to run queue
  CPU 0: Dispatched process 2
Time slot 67
  CPU 0: Processed 2 has finished
  CPU 0: Dispatched process 4
Time slot 68
Time slot 69
  CPU 0: Put process 4 to run queue
  CPU 0: Dispatched process 4
Time slot 70
Time slot 71
  CPU 0: Processed 4 has finished
  CPU 0: Dispatched process 1
Time slot 72
Time slot 73
  CPU 0: Put process 1 to run queue
  CPU 0: Dispatched process 1
Time slot 74
Time slot 75
```

```
  CPU 0: Put process 1 to run queue
  CPU 0: Dispatched process 1
Time slot 76
  CPU 0: Processed 1 has finished
  CPU 0 stopped
```

3.3 Explanation

Input: /input/os_1_singleCPU_mlq

```
2 1 8
1 s4  4
2 s3  3
4 m1s 2
6 s2  3
7 m0s 3
9 p1s 2
11 s0 1
16 s1 0
```

Notes: All of processes in /input/proc.
Output explanation:

- **Time slot 6**
  CPU executes instruction **alloc 300 0** of **m1s**. Result:

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- **Time slot 7**
  CPU executes instruction **alloc 100 1** of **m1s**. Result:

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- **Time slot 8**
  CPU executes instruction **free 0** of **m1s**. Result:

```
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
```

```
PID=3 - Region=0
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- **Time slot 9**
  CPU executes instruction **alloc 100 2** of **m1s**. Result:

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=3 - Region=2 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- **Time slot 34**
  CPU executes instruction **free 2** of **m1s**. Result:

```
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region=2
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- **Time slot 35**
  CPU executes instruction **free 1** of **m1s**. Result:

```
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=3 - Region=1
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Page Number: 0 -> Frame Number: 1
Page Number: 1 -> Frame Number: 0
================================================================
```

- **Time slot 48**
  CPU executes instruction **alloc 300 0** of **m0s**. Result:

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region=0 - Address=00000000 - Size=300 byte
print_pgtbl: 0 - 512
```

```
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
```

- **Time slot 49**
  CPU executes instruction **alloc 100 1** of **m0s**. Result:

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region=1 - Address=0000012c - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
```

- **Time slot 54**
  CPU executes instruction **free 0** of **m0S**. Result:

```
===== PHYSICAL MEMORY AFTER DEALLOCATION =====
PID=5 - Region=0
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
```

- **Time slot 55**
  CPU executes instruction **alloc 100 2** of **m0s**. Result:

```
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=5 - Region=2 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
```

- **Time slot 60**
  CPU executes instruction **write 102 1 20** of **m0s**. Result:

```
===== PHYSICAL MEMORY AFTER WRITING =====
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: 80000003
00000004: 80000002
```

```
Page Number: 0 -> Frame Number: 3
Page Number: 1 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000240: 102
===== PHYSICAL MEMORY END-DUMP =====
================================================================
```

- **Time slot 61**
  CPU executes instruction **write 1 2 1000** of **m0s**. Result:

```
===== PHYSICAL MEMORY AFTER WRITING =====
write region=2 offset=1000 value=1
print_pgtbl: 0 - 512
00000000: c0000000
00000004: 80000002
Page Number: 0 -> Frame Number: 0
Page Number: 1 -> Frame Number: 2
================================================================
===== PHYSICAL MEMORY DUMP =====
BYTE 00000240: 102
BYTE 000003e8: 1
===== PHYSICAL MEMORY END-DUMP =====
================================================================
```