ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN HỆ ĐIỀU HÀNH

Đề tài

Simple Operating System

GVHD: Nguyễn Minh Tâm

STT	Full name	Students ID	Class
1	Dương Khôi Nguyên	2312333	L01
2	Nguyễn Hoàng Minh Phú	2312655	L01
3	Nguyễn Võ Anh Quân	2312847	L01
4	Đoàn Duy Khanh	2311488	L01
5	Nguyễn Thái Hoàng	2311062	L01

Ho Chi Minh City, 12/2025

wwwwwwwwwwwwwww

Table of Contents

1 Bộ lập lịch (Scheduler)

1.1 Trả lời câu hỏi

Câu hỏi: Những ưu điểm của việc sử dụng thuật toán lập lịch được mô tả trong bài tập này (Multi-Level Queue - MLQ) so với các thuật toán lập lịch khác là gì?

Trả lời: Thuật toán Multi-Level Queue (MLQ) được sử dụng trong bài tập này mang lại nhiều ưu điểm so với các thuật toán đơn giản như FCFS hay Round Robin thuần túy:

- **Phân loại tiến trình:** MLQ cho phép phân loại các tiến trình dựa trên độ ưu tiên (priority). Các tiến trình quan trọng hoặc cần phản hồi nhanh có thể được đặt vào hàng đợi có độ ưu tiên cao hơn.
- **Đảm bảo độ phản hồi:** Bằng cách luôn chọn tiến trình từ hàng đợi có độ ưu tiên cao nhất (nếu có), hệ thống đảm bảo các tác vụ quan trọng được xử lý ngay lập tức.
- **Cơ chế Slot (Time Slicing):** Việc cấp phát một số lượng slot (thời gian CPU) nhất định cho mỗi hàng đợi (trong bài tập là `MAX_PRIO - prio`) giúp ngăn chặn tình trạng "đói tài nguyên" (starvation) hoàn toàn cho các tiến trình độ ưu tiên thấp, đồng thời vẫn duy trì được sự ưu tiên cho các tiến trình quan trọng.

1.2 Hiện thực

Hệ thống sử dụng mảng `mlq_ready_queue` gồm `MAX_PRIO` (140) hàng đợi. Cơ chế lập lịch chính được hiện thực trong hàm `get_mlq_proc` tại file `sched.c`.

1.2.1 Cấu trúc dữ liệu

```
1 static struct queue_t mlq_ready_queue[MAX_PRIO];  // 140 priority
    queues
2 static int slot[MAX_PRIO];                 // Time slots per priority
    level
3 static int current_slot[MAX_PRIO];         // Current slot usage tracking
4 static int current_prio = 0;               // Current serving priority
```

```
5 static pthread_mutex_t queue_lock;      // Mutex for thread-safety
```

Trong đó:

- `mlq_ready_queue[i]`: Hàng đợi chứa các tiến trình có độ ưu tiên `i`.
- `slot[i] = MAX_PRIO - i`: Số time slots được cấp cho mỗi priority level. Priority cao hơn (giá trị nhỏ hơn) được nhiều slots hơn.
- `current_slot[i]`: Đếm số slots đã sử dụng cho priority `i` trong round hiện tại.
- `current_prio`: Priority level đang được phục vụ (chỉ dùng để tracking).

1.2.2 Giải thuật get_mlq_proc

Giải thuật được chia thành 2 giai đoạn:

Giai đoạn 1: Lựa chọn dựa trên vị trí

1. **Luôn duyệt từ priority 0 (highest) đến priority 139 (lowest)** để đảm bảo priority preemption.
2. Kiểm tra điều kiện: hàng đợi không rỗng VÀ còn slots available (`current_slot[prio] < slot[prio]`).
3. Nếu thỏa mãn: dequeue process, tăng `current_slot[prio]`, cập nhật `current_prio`.
4. Dừng ngay khi tìm thấy process phù hợp.

Giai đoạn 2: Tái lập & Thử lại

Nếu Giai đoạn 1 không tìm thấy process (tất cả các vị trí đã hết):

1. Tái thiết lập tất cả `current_slot[]` về 0.
2. Thử lại từ priority 0, lần này bỏ qua điều kiện slot.
3. Chọn process đầu tiên tìm được từ hàng đợi không rỗng.

Synchronization: Sử dụng `pthread_mutex_lock(&queue_lock)` để đảm bảo thread-safety khi nhiều CPUs truy cập queues đồng thời.

```
struct pcb_t * get_mlq_proc(void) {
    struct pcb_t * proc = NULL;
    pthread_mutex_lock(&queue_lock);
    for (int prio = 0; prio < MAX_PRIO; prio++) {
        if (!empty(&mlq_ready_queue[prio]) &&
            current_slot[prio] < slot[prio]) {

            proc = dequeue(&mlq_ready_queue[prio]);
            if (proc != NULL) {
                current_slot[prio]++;
                current_prio = prio;
                break;
            }
        }
    }
    if (proc == NULL) {
        for (int i = 0; i < MAX_PRIO; i++) {
            current_slot[i] = 0;
        }
        for (int prio = 0; prio < MAX_PRIO; prio++) {
            if (!empty(&mlq_ready_queue[prio])) {
                proc = dequeue(&mlq_ready_queue[prio]);
                if (proc != NULL) {
                    current_slot[prio] = 1;
                    current_prio = prio;
                    break;
                }
            }
        }
    }
    if (proc != NULL) {
        enqueue(&running_list, proc);
    }
    pthread_mutex_unlock(&queue_lock);
```

```
35      return proc;
36  }
```

Đặc điểm thuật toán:

- **Priority preemption:** Mỗi lần chọn process, thuật toán duyệt từ priority 0 (highest) đến priority 139 (lowest), đảm bảo processes có mức độ ưu tiên cao hơn luôn được xử lý trước.
- **Slot fairness:** Giai đoạn 2 được kích hoạt khi tất cả queues có processes đã hết vị trí, reset bộ đếm và bắt đầu round mới, ngăn đói tài nguyên (starvation) cho các processes có mức ưu tiên thấp hơn.
- **Time complexity:** O(MAX_PRIO) = O(140) = O(1) thời gian là một hằng số cho mỗi lần get process.
- **Thread-safe:** Sử dụng mutex lock bảo vệ critical section khi truy cập shared queues trong multi-CPU environment.

1.3 Kết quả chạy thử nghiệm

1.3.1 Test Case 1: sched_0 (Priority Preemption)

Mục đích: Kiểm tra cơ chế priority preemption - process có priority cao hơn phải preempt process có priority thấp hơn ngay lập tức.

Input: `sched_0`

- Time 0: Load P1 (PID=1, PRIO=4)
- Time 4: Load P2 (PID=2, PRIO=0)

Output (rút gọn):

```
Time slot   0
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
Time slot   1
```

```
        CPU 0: Dispatched process  1
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/s1, PID: 2 PRIO: 0
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2    <-- P2 (PRIO=0) preempts P1 (PRIO=4)
...
Time slot  12
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1    <-- P1 resume after P2 finished
...
Time slot  23
        CPU 0: Processed  1 has finished
```

Phân tích kết quả:

- **Time 0-4:** P1 (PRIO=4) được load và chạy trên CPU 0.
- **Time 4:** P2 (PRIO=0) được load vào hệ thống.
- **Time 5:** P2 được dispatch ngay lập tức, thay thế P1 - đây là behavior của priority preemption.
- **Time 5-12:** P2 chạy liên tục (không bị preempt vì có priority cao nhất) và hoàn thành tại time 12.
- **Time 12-23:** P1 được dispatch lại và chạy cho đến khi hoàn thành.
- **Kết luận:** Cơ chế priority preemption hoạt động đúng - process có priority cao hơn (giá trị nhỏ hơn) sẽ preempt process có priority thấp hơn ngay lập tức.

1.3.2 Test Case 2: sched_1 (MLQ Round-Robin)

Mục đích: Kiểm tra FIFO scheduling trong cùng priority level và MLQ round-robin behavior.

Input: `sched_1`

- Time 0: Load P1 (PID=1, PRIO=4)
- Time 4: Load P2 (PID=2, PRIO=0)
- Time 6: Load P3 (PID=3, PRIO=0)
- Time 7: Load P4 (PID=4, PRIO=0)

Output (rút gọn):

```
Time slot   5
        CPU 0: Dispatched process  2
Time slot   7
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   9
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2    <-- Round-robin: P2->P3->P2
Time slot  11
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  4    <-- Continue: P2->P3->P2->P4
Time slot  13
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3    <-- Pattern: P2->P3->P4->P3
...
Time slot  22
        CPU 0: Processed  2 has finished
Time slot  34
```

```
        CPU 0: Processed  3 has finished
Time slot  35
        CPU 0: Processed  4 has finished
        CPU 0: Dispatched process  1    <-- P1 chỉ chạy sau khi all PRIO=0 finish
```

Phân tích kết quả:

- **Time 0-4:** P1 (PRIO=4) được load và bắt đầu chạy.
- **Time 4-7:** P2, P3, P4 (tất cả PRIO=0) được load vào hệ thống.
- **Time 5:** P2 preempt P1 do có priority cao hơn.
- **Scheduling pattern:** P2→P3→P2→P4→P3→P2→P4→P3... - đây là round-robin giữa các processes cùng priority.
- **FIFO behavior:** Trong cùng priority 0, các processes được dispatch theo thứ tự vào queue (P2 trước P3 trước P4).
- **Time 22-35:** P2, P3, P4 hoàn thành lần lượt.
- **Time 35-46:** P1 mới được CPU sau khi tất cả processes PRIO=0 đã hoàn thành.
- **Kết luận:** MLQ đảm bảo FIFO trong cùng priority level và round-robin scheduling với time slice.

1.3.3 Test Case 3: sched (Multi-CPU)

Mục đích: Kiểm tra MLQ scheduling trên môi trường multi-CPU.
Input: `sched` (2 CPUs)

- Time 0: Load P1 (PID=1, PRIO=1)
- Time 1: Load P2 (PID=2, PRIO=0)
- Time 2: Load P3 (PID=3, PRIO=0)

Output (rút gọn):

```
Time slot   1
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/p2s, PID: 2 PRIO: 0
Time slot   2
        CPU 1: Dispatched process  2    <-- P2 on CPU1
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 0
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  3    <-- P3 preempts P1 (0 < 1)
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  2
...
Time slot  13
        CPU 1: Processed  2 has finished
Time slot  14
        CPU 1: Dispatched process  1    <-- P1 gets CPU1 after P2 finished
Time slot  16
        CPU 0: Processed  3 has finished
Time slot  20
        CPU 1: Processed  1 has finished
```

Phân tích kết quả:

- **Time 1:** CPU 0 dispatch P1 (PRIO=1), P2 (PRIO=0) được load.
- **Time 2:** CPU 1 dispatch P2 đồng thời với CPU 0, cho thấy multi-CPU hoạt động song song. P3 (PRIO=0) được load.
- **Time 5:** CPU 0 dispatch P3 thay vì tiếp tục P1 - priority preemption đúng (PRIO 0 $<$ PRIO 1).

- **Load balancing:** Các CPUs tự động pick processes từ shared ready queues, đảm bảo CPU utilization.
- **Time 13-20:** P2 và P3 hoàn thành, P1 được dispatch lại trên CPU 1.
- **Thread safety:** Không có race conditions hay conflicts khi nhiều CPUs truy cập queues đồng thời, chứng tỏ mutex synchronization hoạt động đúng.
- **Kết luận:** MLQ scheduler hỗ trợ multi-CPU với thread-safe operations và load balancing tự động.

1.3.4 Tổng kết kết quả test

Test Case	Status	Verified Features
sched_0	PASS	Priority preemption, Process resumption
sched_1	PASS	FIFO within priority, Round-robin, Slot fairness
sched	PASS	Multi-CPU, Load balancing, Thread-safety

Table 1: Kết quả test scheduler

Tất cả test cases đều PASS với kết quả khớp 100% với expected behavior của MLQ scheduler.