ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÀI TẬP LỚN

HỆ ĐIỀU HÀNH (TN) – CO2018

ĐỀ TÀI

SIMPLE OPERATING SYSTEM

Giảng viên hướng dẫn:	ThS. Hoàng Lê Hải Thanh	
Giảng viên lí thuyết:	TS. Lê Thanh Vân	
Lớp:	L01	
Nhóm:	hehe	
Sinh viên thực hiện:	A	22
	B	22
	C	21
	D	22

THÀNH PHỐ HỒ CHÍ MINH – 11/2024

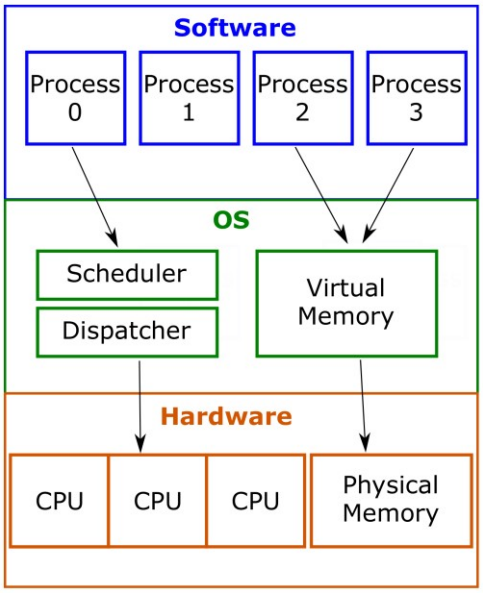
Danh sách thành viên nhóm

STT	Họ tên	MSSV	Tỉ lệ đóng góp
1	A	22	100%
2	B	22	100%
3	C	21	100%
5	Lê Minh Thế	22	100%

Mục lục

1 Dẫn nhập

Bài báo cáo này trình bày quá trình và kết quả thực hiện bài tập lớn môn Hệ điều hành (TN) – CO2018 của nhóm. Mục tiêu của bài báo cáo là hiện thực, mô phỏng một hệ điều hành đơn giản. Qua đó, giúp sinh viên nắm được kiến trúc tổng thể của một hệ điều hành cũng như các khái niệm cơ bản có trong hệ điều hành: định thời (scheduing), đồng bộ (synchronization), quản lí bộ nhớ (memory management).



Hình 1.1: *Minh họa tổng quản các module chính có trong hệ điều hành sẽ xây dựng*

Minh họa trên cho thấy hệ điều hành ta sắp xây dựng sẽ cho phép nhiều tiến trình do người dùng tạo ra được chia sẻ và sử dụng các tài nguyên điện toán ảo. Do đó, trong bài tập này, ta sẽ tập trung vào việc triển khai bộ định thời/bộ điều phối và công cụ bộ nhớ ảo.

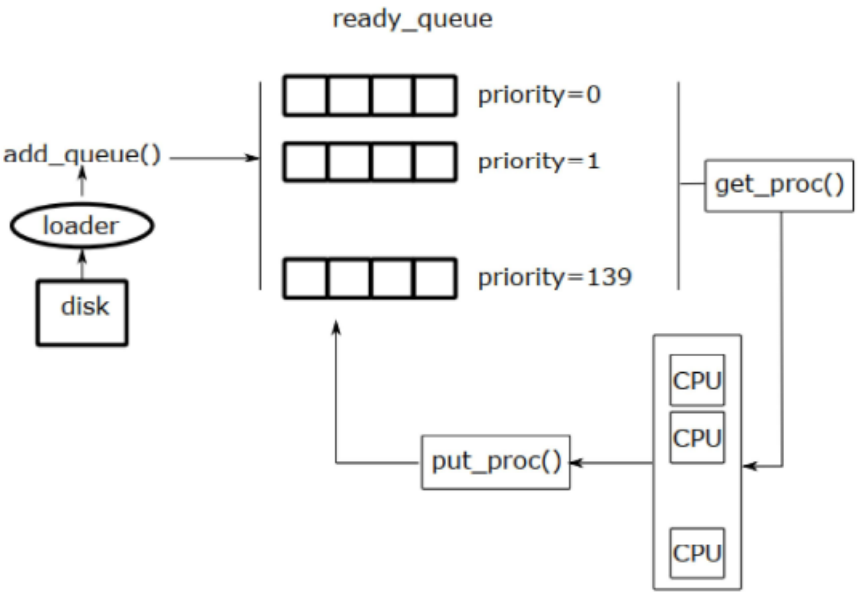
Bộ định thời đóng vai trò quan trọng trong việc quản lý và phân phối tài nguyên hệ thống một cách có quy tắc cho các tiến trình. Nhiệm vụ chính của bộ định thời là lựa chọn một tiến trình được cấp phát CPU để thực thi tại một thời điểm và trong một khoảng thời gian xác định. Mục tiêu của bộ định thời là tối ưu hóa việc sử dụng CPU, tránh các xung đột ảnh hưởng xấu đến hệ điều hành.

1.1 Cơ sở lý thuyết

Hệ điều hành được thế kế để làm việc trên nhiều bộ xử lí (CPU). Để thực hiện việc đó, người ta hiện thực một hàng đợi gọi là *ready queue* – hàng đợi lưu trữ các tiến trình cần được định thời. Khi một CPU trống, bộ định thời sẽ dùng thuật toán xác định một tiến trình trong *ready queue* sẽ được cấp phát tài nguyên và nạp vào CPU để thực thi.

Ở bài tập lớn này, ta sử dụng thuật toán Multilevel Queue (MLQ) kết hợp Round-Robin để xây dựng bộ định thời cho hệ điều hành đơn giản. **Thuật toán MLQ:** Bộ định thời sẽ gồm `MAX_PRIO` các hàng đợi được đánh số độ ưu tiên khác nhau (số càng nhỏ, độ ưu tiên càng lớn). Các tiến trình khi xuất hiện sẽ được đưa vào hàng đợi có mức độ ưu tiên bằng với độ ưu tiên của tiến trình đó.

Các quá trình thuộc hàng đợi có độ ưu tiên cao hơn sẽ được bộ định thời ưu tiên lựa chọn cho thực thi trước. Mỗi một quá trình khi được nạp vào CPU sẽ được thực thi tối đa trong một khoảng thời gian cố định, gọi là *time slice*. Khi kết thúc *time slide*, quá trình (nếu chưa hoàn tất) sẽ được đưa trở lại hàng đợi **ban đầu**.



Hình 1.2: *Hoạt động của quá trình định thời*

Chính sách MLQ: Để tránh tình trạng trì hoãn thực thi (delayed execution) của các tiến trình có độ ưu tiên thấp do phải chờ đợi quá lâu, bên cạnh độ ưu tiên, mỗi hàng đợi có thêm một thông số cố định *slot* với:

$$slot_i = (MAX\_PRIO - prio_i)$$

Có thể thấy, hàng đợi có độ ưu tiên càng lớn (giá trị $prio_i$ càng nhỏ) thì có thông số $slot_i$ tương ứng càng lớn. Thông số $slot_i$ này cho biết các quá trình trong hàng đợi i được phép sử dụng CPU trong tối đa $slot_i$ *time slice*. Khi dùng hết, hệ thống phải dành tài nguyên cho các quá trình ở hàng đợi khác và để lại các tiến trình chưa hoàn thành xong ở hàng đợi ban đầu cho lượt sau trong tương lai.

```
prio = 0        |         1          | ....    | MAX_PRIO - 1
slot = MAX_PRIO |   MAX_PRIO - 1     | ....    |      1
```

Hình 1.3: *Ví dụ về slot của các hàng đợi với bộ định thời có MAX_ PRIO=140*

1.2 Trả lời câu hỏi:

Câu hỏi: Ưu điểm của việc sử dụng thuật toán định thời được mô tả trong bài tập này so với các thuật toán định thời khác đã học là gì?

Trả lời: Thuật toán định thời sử dụng Multilevel Queue kết hợp Round-Robin và cơ chế *time slice slot* trong bài tập lớn này có một số ưu điểm so với các thuật toán định thời đã học như First-Come - First-Served (FCFS), Shortest-Job-First (SJF), Shortest-Remaining-Time-First (SRTF), Round-Robin (RR), Priority Queue và các Multilevel Queue cơ bản:

- **Xử lý các tiến trình theo ưu tiên:** Thuật toán MLQ trong bài tập lớn này cho phép phân cấp các tiến trình hiệu quả hơn dựa trên độ ưu tiên của hàng đợi lưu trữ chúng. Thuật toán này sử dụng nhiều hàng đợi, mỗi hàng đợi được chỉ định một mức ưu tiên cụ thể. Điều này có lợi khi so sánh với các thuật toán FCFS hoặc SJF đơn giản.
- **Cơ chế Độ ưu tiên cố định không phản hồi** (Fixed Priority without Feedback)**:** Thuật toán định thời MLQ sử dụng trong bài tập này chỉ định các quá trình được xếp vào hàng đợi với độ ưu tiên cụ thể và sẽ ở hàng đợi đó trong toàn bộ vòng đời thực thi của chúng, không sử dụng cơ chế phản hồi cho phép di chuyển các quá trình giữa các hàng đợi. Việc này giúp đơn giản hóa thiết kế và triển khai so với các thuật toán định thời phản hồi. Nó đảm bảo rằng các quá trình có mức ưu tiên cao được chọn thực thi trước các quá trình có mức ưu tiên thấp hơn mà không cần tính toán lại hoặc di chuyển các quy trình giữa các hàng đợi. Đặc điểm này mang lại tính ổn định, có lợi cho các tác vụ thời gian thực hoặc có mức ưu tiên cao, trong đó cần tuân thủ nghiêm ngặt về sự ưu tiên.
- **Cơ chế duyệt các hàng đợi:** Cơ chế này tính toán số lượng *slot slice time* CPU được phân bổ cho mỗi hàng đợi dựa trên mức độ ưu tiên của hàng đợi đó, trong đó các hàng đợi có mức độ ưu tiên cao hơn (prio gần 0 hơn) sẽ nhận được nhiều *slot time slice* CPU hơn. Ưu điểm: Cơ chế duyệt này phân bổ thời gian CPU theo tỷ lệ có lợi cho các quy trình có mức độ ưu tiên cao. Bằng cách phân bổ $slot_i = MAX\_PRIO - prio_i$, các hàng đợi có mức độ ưu tiên cao hơn (giá trị prio nhỏ hơn) sẽ được truy cập CPU thường xuyên hơn. Điều này giúp giải quyết tình trạng delayed execution đối với các quá trình ở hàng đợi có độ ưu tiên thấp thông qua việc các hàng đợi khi đã hết *slot time slice* sẽ được hệ thống chuyển tài nguyên sang dùng cho các quá trình có trong hàng đợi khác, đảm bảo các quá trình trong các hàng đợi sẽ không bị đình trệ quá lâu.

So sánh tổng thể:

- So với FCFS, định thời MLQ này đảm bảo phản hồi tốt hơn cho các tác vụ có mức độ ưu tiên cao, tránh định trệ khi các tác vụ dài làm chậm tất cả các quy trình theo sau.
- So với SJF và SRTF, thuật toán dùng trong bài tập duy trì được thứ tự thực thi có thể dự đoán được dựa trên mức độ ưu tiên.
- So với định thời ưu tiên với phản hồi động, cơ chế ưu tiên cố định sử dụng trong thuật toán này làm giảm độ phức tạp và đảm bảo tính ổn định.
- So với các hệ thống Multilevel Queue cơ bản, cơ chế *slot* trong MLQ này đảm bảo phân bổ thời gian CPU hiệu quả hơn trên các mức độ ưu tiên khác nhau, tránh hiệu ứng deleyed execution đối với các quá trình có độ ưu tiên thấp.

1.3 Công việc và Hiện thực

- Trong `queue.h`:

  Bổ sung trường `slot` vào `struct queue_t`: Thể hiện cơ chế *slot time slice* cho trình định thời.

```c
#define MAX_QUEUE_SIZE 10

struct queue_t {
    struct pcb_t * proc[MAX_QUEUE_SIZE];
    int size;
    int slot;
};
```

- Trong `queue.c`:
  - Hiện thực hàm `enqueue()`: Đưa một tiến trình mới vào một hàng đợi.

```c
void enqueue(struct queue_t * q, struct pcb_t * proc) {
    /* TODO: put a new process to queue [q] */
    if (full(q)){
        printf("Queue is full, cannot enqueue process.\n");
        return;
    }

    q->proc[q->size++] = proc;
}
```

  - Hiện thực hàm `dequeue()`: Lấy và trả về tiến trình ở đầu một hàng đợi.

```c
struct pcb_t * dequeue(struct queue_t * q) {
    if (empty(q)) {
        printf("Queue is empty, cannot dequeue process.\n");
        return NULL;
    }

    struct pcb_t * ret_proc = q->proc[0];

    // left-shifting the rest queue
    for(int i = 1; i < q->size; ++i){
            q->proc[i-1] = q->proc[i];
    }

    q->size--;
    return ret_proc;
}
```

– Hiện thực hàm `update_slot()`: Cập nhật giá trị slot mới cho một hàng đợi.

```c
void update_slot(struct queue_t * q, int new_val) {
    if(new_val < 0){
        new_val = 0;
    }
    q->slot = new_val;
}
```

- Trong `sched.h`:

  Bổ sung thêm/chỉnh sửa lại các nguyên mẫu hàm trong trường hợp có định nghĩa `MLQ_SCHED`:

```c
    /* decrease slot of queue having priority [prio] to [used_slot] units */
    void decrease_slot(int used_slot, unsigned long prio);

    /* Get the next process from ready queue and return the slot of
       queue containing this process to [slot_of_queue_of_proc] */
    struct pcb_t * get_proc(int *slot_of_queue_of_proc);
```

- Trong `sched.c`:

  – Viết lại hàm `init_scheduler()`: Bổ sung khởi tạo giá trị `slot` ban đầu cho từng hàng đợi của bộ định thời.

```c
void init_scheduler(void) {
#ifdef MLQ_SCHED
    for (unsigned long prio = 0; prio < MAX_PRIO; ++prio){
        mlq_ready_queue[prio].size = 0;
```

```
        // Addition: init slot for each queue (slot time slice mechanism)
        mlq_ready_queue[prio].slot = MAX_PRIO - prio;
    }
#endif
    ready_queue.size = 0;
    run_queue.size = 0;
    pthread_mutex_init(&queue_lock, NULL);
}
```

– Hiện thực hàm `check_no_more_slots(void)`: Kiểm tra liệu các hàng đợi không rỗng đều đã hết `slot` hay không. Nếu đúng, gán lại giá trị `slot` của từng hàng đợi về giá trị ban đầu ($MAX\_PRIO - prio$).

```
void check_no_more_slots(void){
    /* if all non-empty queues have no more slot, reset their slot value */

    for(unsigned long prio = 0; prio < MAX_PRIO; ++prio)
        if(!empty(&mlq_ready_queue[prio]) &&
                        mlq_ready_queue[prio].slot > 0){
            /* exist a non-empty queue that has slot => no need to update => return */
            return;
        }

    /* reset all slots in all queues */
    for(unsigned long prio = 0; prio < MAX_PRIO; ++prio){
        update_slot(&mlq_ready_queue[prio], MAX_PRIO - prio);
    }
}
```

– Hiện thực hàm `get_mlq_proc(int *slot_of_queue_of_proc)`: Lấy và trả về một quá trình đang đợi thực thi trong hệ thống định thời, đồng thời trả về giá trị `slot` hiện tại của hàng đợi chứa tiến trình đó.

```
struct pcb_t * get_mlq_proc(int* slot_of_queue_of_proc) {
    /*TODO: get a process from PRIORITY [ready_queue].
     * Remember to use lock to protect the queue.
     * */
    struct pcb_t * ret_proc = NULL;

    pthread_mutex_lock(&queue_lock);

    /* first reset their slot value if all non-empty queues have no more slot */
    check_no_more_slots();

    for (unsigned long prio = 0; prio < MAX_PRIO; ++prio) {
        if (!empty(&mlq_ready_queue[prio]) &&
                        mlq_ready_queue[prio].slot > 0) {
            ret_proc = dequeue(&mlq_ready_queue[prio]);
```

```
            // return parameter
            *slot_of_queue_of_proc = mlq_ready_queue[prio].slot;
            break;
        }
    }

    pthread_mutex_unlock(&queue_lock);
    return ret_proc;
}
```

- Trong `os.c`:

  Viết lại hàm `cpu_routine()`: Bổ sung thêm các xử lý liên quan đến *slot time slice*.

```
static void * cpu_routine(void * args) {
    struct timer_id_t * timer_id = ((struct cpu_args*)args)->timer_id;
    int id = ((struct cpu_args*)args)->id;
    /* Check for new process in ready queue */
    int time_left = 0;
    struct pcb_t * proc = NULL;
    // Addition: slot_of_queue_of_proc is the slot of queue containing the process
    int slot_of_queue_of_proc = 0;
    while (1) {
        /* Check the status of current process */
        if (proc == NULL) {
            /* No process is running, the we load new process from
            * ready queue */
            proc = get_proc(&slot_of_queue_of_proc);
            if (proc == NULL) {
                next_slot(timer_id);
                continue; /* First load failed. skip dummy load */
            }
        }else {
            /* process has finished its job or run out of time slice */

            /* Addition: actual time used by the process in current time slot */
            int used_time = time_slot - time_left;

            if (proc->pc == proc->code->size) {
            /* The porcess has finish it job */
            printf("\tCPU %d: Processed %2d has finished\n", id ,proc->pid);

            /* Addition: decrease slot of queue containing process */
            decrease_slot(used_time, proc->prio);
            free(proc);
            proc = get_proc(&slot_of_queue_of_proc); // Load new process
            time_left = 0;
        }else if (time_left == 0) {
            /* The process has done its job in current time slot */
```

```
36
37              printf("\tCPU %d: Put process %2d to run queue\n", id, proc->pid
    );
38
39              /* Addition: decrease slot of queue containing process */
40              decrease_slot(used_time, proc->prio);
41              put_proc(proc); // Put process back to run queue
42              proc = get_proc(&slot_of_queue_of_proc); // Load new process
43          }}
44
45          /* Recheck process status after loading new process */
46          if (proc == NULL && done) {
47              /* No process to run, exit */
48              printf("\tCPU %d stopped\n", id);
49              break;
50          }else if (proc == NULL) {
51              /* There may be new processes to run in
52               * next time slots, just skip current slot */
53              next_slot(timer_id);
54              continue;
55          }else if (time_left == 0) {
56              printf("\tCPU %d: Dispatched process %2d\n", id, proc->pid);
57
58              /* Addition: calculate the time left for the process.
59                 In case slot of queue containing process is smaller than time
     slot,
60                 the process will only run in that slot */
61              time_left = time_slot <= slot_of_queue_of_proc ? time_slot :
    slot_of_queue_of_proc;
62          }
63
64          /* Run current process */
65          run(proc);
66          time_left--;
67          next_slot(timer_id);
68      }
69      detach_event(timer_id);
70      pthread_exit(NULL);
71  }
```

1.3.1 Kiểm thử và Phân tích

Ta sẽ kiểm tra module định thời đã cài đặt bằng việc chạy một số test case.

1.3.1.1 Test case 1:

Input:

Nội dung tệp tin sched1	time slice	**Số lượng CPU**	**Số lượng quá trình**	MAX_PRIO (thiết lập tại os-cfg.h)
4 1 4 0 p1s 1 1 p2s 0 2 p3s 0 4 p4s 1	4	1	4	140

Bảng 1.1: *Tệp tin mô tả và giải thích các thông số của test case*

Tệp tin	p1s	p2s	p3s	p4s
Nội dung	1 10 calc calc calc calc calc calc calc calc calc	20 6 calc calc calc calc calc calc	7 17 calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc	20 13 calc calc calc calc calc calc calc calc calc calc calc calc calc calc

Bảng 1.2: *Các tệp tin chứa nội dung các quá trình sử dụng trong test case*

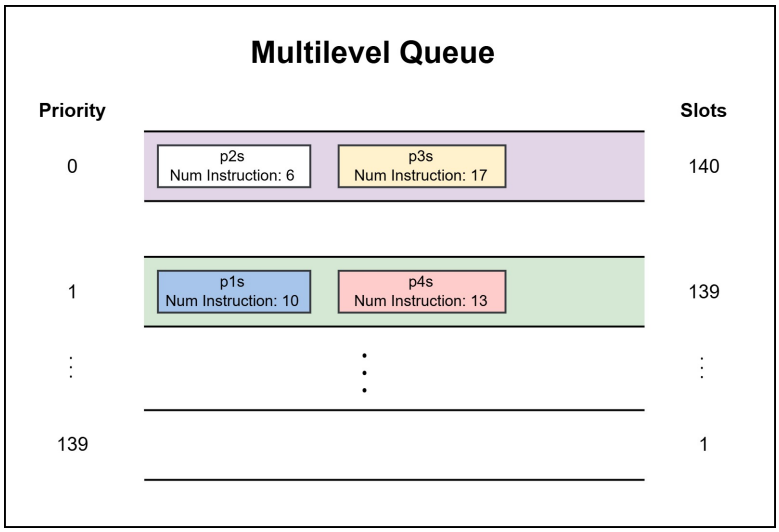
Output:

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p1s, PID: 1 PRIO: 1
Time slot   1
        Loaded a process at input/proc/p2s, PID: 2 PRIO: 0
        CPU 0: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 0
Time slot   3
Time slot   4
        Loaded a process at input/proc/p4s, PID: 4 PRIO: 1
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   6
Time slot   7
Time slot   8
Time slot   9
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot  14
Time slot  15
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  3
Time slot  16
Time slot  17
Time slot  18
Time slot  19
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  20
Time slot  21
Time slot  22
Time slot  23
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
```

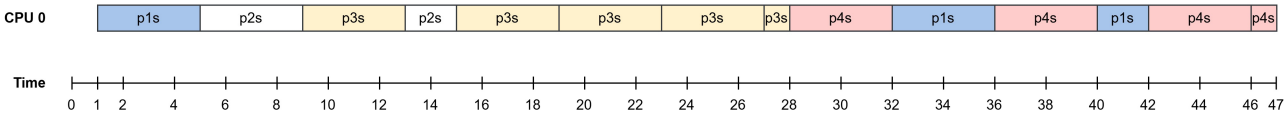
```
Time slot  24
Time slot  25
Time slot  26
Time slot  27
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  28
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  4
Time slot  29
Time slot  30
Time slot  31
Time slot  32
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  1
Time slot  33
Time slot  34
Time slot  35
Time slot  36
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  37
Time slot  38
Time slot  39
Time slot  40
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  1
Time slot  41
Time slot  42
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  4
Time slot  43
Time slot  44
Time slot  45
Time slot  46
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  47
        CPU 0: Processed  4 has finished
        CPU 0 stopped
```

Hình 1.4: *Kết quả khi thực thi chương trình với test case sched1*

Phân tích:



Hình 1.5: *Mô phỏng cấu trúc Multilevel Queue ban đầu*



Hình 1.6: *Biểu đồ Gantt ứng với output*

- Tại *time=0*: Quá trình `p1s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=1*:
  - Quá trình `p2s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
  - Bộ định thời thực hiện tìm quá trình sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, chỉ có duy nhất `p1s` nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.

  Hai tác vụ trên là song hành với nhau do thực thi trên hai luồng khác nhau.
- Tại *time=2*: Quá trình `p3s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=4*: Quá trình `p4s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=5*:
  - Quá trình `p1s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p2s` nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.
- Tại *time=9*:
  - Quá trình `p2s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p3s` nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.
- Tại *time=13*:
  - Quá trình `p3s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p2s` nên đây sẽ là quá trình được chọn. Do trước đó đã thực thi trong 4 time slice, `p2s` chỉ còn 6 - 4 = 2 instruction cần thực thi, nên sẽ mất 2 time slice cho lần thực thi này.
- Tại *time=15*:

  - Quá trình `p2s` dừng và được giải phóng do đã thực thi xong.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p3s` nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.

- Tại *time=19*:
  - Quá trình `p3s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p3s` nên đây sẽ là quá trình được chọn (vừa mới được thêm lại nhưng chỉ có duy nhất và ở hàng đợi ưu tiên cao nên tiếp tục được chọn), thực thi trong 4 time slice.

- Tại *time=23* và *time=27*: Tương tự như trên, `p3s` tiếp tục được thêm lại vào hàng đợi và ngay lập tức nạp vào CPU. Riêng tại *time=27*, `p3s` chỉ cần thực thi trong 1 time slice trước khi hoàn thành.

- Tại *time=28*:
  - Quá trình `p3s` dừng và được giải phóng do đã thực thi xong.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, hàng đợi *prio=0* không còn quá trình nào nữa. Bộ định thời chuyển sang hàng đợi có độ ưu tiên thấp liền kề. Tại đầu hàng đợi *prio=1* có quá trình `p4s` nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.

- Tại *time=32*:
  - Quá trình `p4s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Tại đầu hàng đợi *prio=1* có quá trình `p1s` nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.

- Tại *time=36*, *time=40*, *time=42*, và *time=46*: Hai quá trình `p1s` và `p4s` xen kẽ nhau được nạp và thực thi cho đến khi kết thúc (tại *time=42* đối với `p1s` và *time=47* đối với `p4s`).

Nhận xét:

- Kết quả thu được là phù hợp và đúng với lý thuyết, dự đoán.
- Turnaround time, waiting time của từng quá trình cũng như giá trị trung bình:

Quá trình	**Turnaround time** (time slice)	**Waiting time** (time slice)
`p1s`	$TT_1 = 42 - 1 = 41$	$WT_1 = 41 - 10 = 31$
`p2s`	$TT_2 = 15 - 5 = 10$	$WT_2 = 10 - 6 = 4$
`p3s`	$TT_3 = 28 - 9 = 19$	$WT_3 = 19 - 17 = 2$
`p4s`	$TT_4 = 47 - 28 = 19$	$WT_4 = 19 - 13 = 6$
	$\overline{TT} = 22.25$	$\overline{WT} = 10.75$

TT = thời điểm kết thúc – thời điểm ban đầu
$WT = TT$ – thời gian thực thi

Bảng 1.3: *Turnaround time, waiting time của từng quá trình và giá trị trung bình*

- Do thiết kế này đang sử dụng số hàng đợi quá nhiều (`MAX_PRIO` = 140), nhưng số lượng quá trình cũng như số instruction của mỗi quá trình không quá lớn, dẫn đến lượng *slot time slice* cấp cho các hàng đợi đầu tiên là dư thừa. Test case này không thể hiện được cơ chế duyệt các hàng đợi dựa trên giá trị *slot time slice*. Từ đó có thể nhìn thấy tình trạng deleyed execution với việc các quá trình ở hàng đợi *prio=0* (`p2s` và `p3s`) được bộ định thời ưu tiên luân phiên chọn thực hiện trước. Các quá trình ở hàng đợi *prio=1* (`p1s` và `p4s`) phải chờ các tiến trình ưu tiên cao hơn thực hiện xong mới được duyệt tới (riêng trường hợp `p1s` được thực thi đầu tiên là do đây là quá trình xuất hiện trước nhất, dẫn đến TT và WT của quá trình này rất lớn).

1.3.1.2 Test case 2

Để kiểm tra cơ chế duyệt các hàng đợi dựa trên thông số *slot time slice*, ta sẽ thực hiện kiểm thử chương trình với lượng `MAX_PRIO` nhỏ hơn (mục đích để giá trị *slot* của các hàng đợi cũng theo đó là giá trị nhỏ, giảm được xuống về 0), các nội dung khác về tệp tin đầu vào được giữ nguyên.

Input:

Nội dung tệp tin `sched1`	`time slice`	**Số lượng CPU**	**Số lượng quá trình**	`MAX_PRIO`

`4 1 4`
`0 p1s 1`
`1 p2s 0`
`2 p3s 0`
`4 p4s 1`
	`4`	`1`	`4`	**7**

Bảng 1.4: *Tệp tin mô tả và giải thích các thông số của test case*

Tệp tin	`p1s`	`p2s`	`p3s`	`p4s`
Nội dung	1 10 calc calc calc calc calc calc calc calc calc	20 6 calc calc calc calc calc calc	7 17 calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc	20 13 calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc

Bảng 1.5: *Các tệp tin chứa nội dung các quá trình sử dụng trong test case*

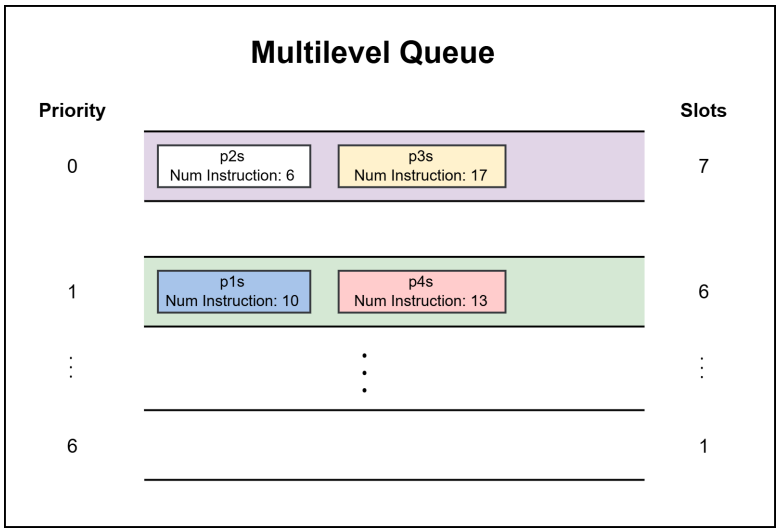
Output:

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p1s, PID: 1 PRIO: 1
Time slot   1
        Loaded a process at input/proc/p2s, PID: 2 PRIO: 0
        CPU 0: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 0
Time slot   3
Time slot   4
        Loaded a process at input/proc/p4s, PID: 4 PRIO: 1
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   6
Time slot   7
Time slot   8
Time slot   9
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  10
Time slot  11
Time slot  12
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
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 0: Processed  2 has finished
        CPU 0: Dispatched process  3
Time slot  17
Time slot  18
Time slot  19
Time slot  20
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  21
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
```

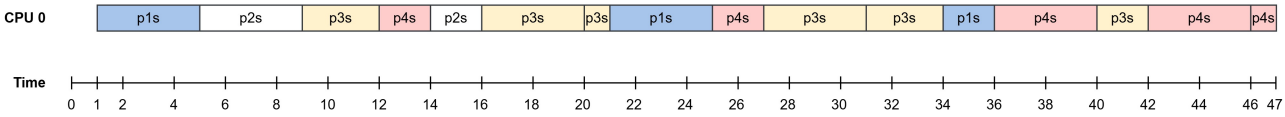
```
Time slot  22
Time slot  23
Time slot  24
Time slot  25
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  26
Time slot  27
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot  28
Time slot  29
Time slot  30
Time slot  31
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  32
Time slot  33
Time slot  34
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  35
Time slot  36
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  4
Time slot  37
Time slot  38
Time slot  39
Time slot  40
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot  41
Time slot  42
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  4
Time slot  43
Time slot  44
Time slot  45
Time slot  46
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  47
        CPU 0: Processed  4 has finished
        CPU 0 stopped
```

Hình 1.7: *Kết quả khi thực thi chương trình với test case sched1*

Phân tích:



Hình 1.8: *Mô phỏng cấu trúc Multilevel Queue ban đầu*



Hình 1.9: *Biểu đồ Gantt ứng với output*

- Tại *time=0*: Quá trình `p1s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=1*:
  - Quá trình `p2s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
  - Bộ định thời thực hiện tìm quá trình sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, chỉ có duy nhất `p1s`, đồng thời hàng đợi chứa quá trình này vẫn còn *slot* nên đây sẽ là quá trình được chọn, thực thi trong $\min(\textit{time slice}, slot_1) = \min(4, 6) = 4$ time slice.

  Hai tác vụ trên là song hành với nhau do thực thi trên hai luồng khác nhau.
- Tại *time=2*: Quá trình `p3s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=4*: Quá trình `p4s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=5*:
  - Quá trình `p1s` dừng và được thêm lại vào hàng đợi. $slot_1$ giảm đi 4 đơn vị (do quá trình `p1s` thuộc hàng đợi *prio=1*). Giá trị slot hiện tại của các hàng đợi không rỗng:

    $$slot_0 = 7 \qquad\qquad slot_1 = 2$$

  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p2s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn, thực thi trong $\min(\textit{time slice}, slot_0) = \min(4, 7) = 4$ time slice.
- Tại *time=9*:
  - Quá trình `p2s` dừng và được thêm lại vào hàng đợi. $slot_0$ giảm đi 4 đơn vị. Giá trị slot hiện tại của các hàng đợi không rỗng:

    $$slot_0 = 3 \qquad\qquad slot_1 = 2$$

  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p3s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn, thực thi trong $\min(\textit{time slice}, slot_0) = \min(4, 3) = 3$ time slice.

- Tại *time=12*:
  - Quá trình `p3s` dừng và được thêm lại vào hàng đợi. $slot_0$ giảm đi 3 đơn vị. Giá trị slot hiện tại của các hàng đợi không rỗng:

$$slot_0 = 0 \qquad slot_1 = 2$$

  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p2s`. *Tuy nhiên*, *slot* của hàng đợi này đã hết. Hệ thống chuyển sang duyệt hàng đợi tiếp theo. Tại đầu hàng đợi *prio=1* có quá trình `p4s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn, thực thi trong $\min(\textit{time slice}, slot_1) = \min(4, 2) = 2$ time slice.

- Tại *time=14*:
  - Quá trình `p4s` dừng và được thêm lại vào hàng đợi. $slot_1$ giảm đi 2 đơn vị. Giá trị slot hiện tại của các hàng đợi không rỗng:

$$slot_0 = 0 \qquad slot_1 = 0$$

  - Bộ định thời, trước khi tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên, sẽ kiểm tra và thấy rằng tất cả *slot* của các hàng đợi không rỗng đều đã hết và sẽ tiến hành gán lại giá trị ban đầu:

$$slot_0 = 7 \qquad slot_1 = 6$$

    Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p2s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn.vDo trước đó đã thực thi trong 4 time slice, `p2s` chỉ còn 6 - 4 = 2 instruction cần thực thi, nên sẽ mất 2 time slice cho lần thực thi này.

- Tại *time=16*:
  - Quá trình `p2s` dừng và được giải phóng do đã thực thi xong. $slot_0$ giảm đi 2 đơn vị. Giá trị slot hiện tại của các hàng đợi không rỗng:

$$slot_0 = 5 \qquad slot_1 = 6$$

  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p3s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.

- Tại *time=20*:

  - Quá trình `p3s` dừng và được thêm lại vào hàng đợi. $slot_0$ giảm đi 4 đơn vị. Giá trị slot hiện tại của các hàng đợi không rỗng:

$$slot_0 = 1 \qquad\qquad slot_1 = 6$$

  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p3s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn, thực thi trong $\min(\textit{time slice}, slot_0) = \min(4, 1) = 1$ time slice.

- Tại *time=21*:

  - Quá trình `p3s` dừng và được thêm lại vào hàng đợi. $slot_0$ giảm đi 1 đơn vị. Giá trị slot hiện tại của các hàng đợi không rỗng:

$$slot_0 = 0 \qquad\qquad slot_1 = 6$$

  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p3s`. *Tuy nhiên*, *slot* của hàng đợi này đã hết. Hệ thống chuyển sang duyệt hàng đợi tiếp theo. Tại đầu hàng đợi *prio=1* có quá trình `p1s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn, thực thi trong 4 time slice.

- Tại *time=25*:

  - Quá trình `p1s` dừng và được thêm lại vào hàng đợi. $slot_1$ giảm đi 4 đơn vị. Giá trị slot hiện tại của các hàng đợi không rỗng:

$$slot_0 = 0 \qquad\qquad slot_1 = 2$$

  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=1* có quá trình `p4s` và *slot* của hàng đợi này vẫn còn nên đây sẽ là quá trình được chọn, thực thi trong 2 time slice – 2 slot còn lại của hàng đợi.

Ở các bước kế tiếp, bộ định thời sẽ cập nhật lại giá trị cho các *slot* (do tất cả slot của các hàng đợi không rỗng đều đã hết), sau đó tiếp tục xác định các quá trình được thực thi như đã trình bày bên trên.

Nhận xét:

- Kết quả thu được là phù hợp và đúng với lý thuyết, dự đoán.
- Turnaround time, waiting time của từng quá trình cũng như giá trị trung bình:

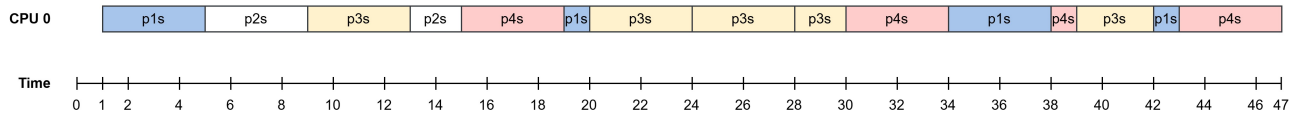
Quá trình	**Turnaround time** (time slice)	**Waiting time** (time slice)
p1s	$TT_1 = 36 - 1 = 35$	$WT_1 = 35 - 10 = 25$
p2s	$TT_2 = 16 - 5 = 11$	$WT_2 = 11 - 6 = 5$
p3s	$TT_3 = 42 - 9 = 33$	$WT_3 = 33 - 17 = 16$
p4s	$TT_4 = 47 - 12 = 35$	$WT_4 = 35 - 13 = 22$
	$\overline{TT} = 28.50$	$\overline{WT} = 17.00$

TT = thời điểm kết thúc – thời điểm ban đầu
$WT = TT$ – thời gian thực thi

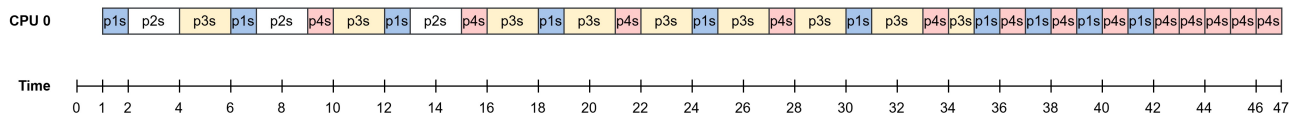
Bảng 1.6: *Turnaround time, waiting time của từng quá trình và giá trị trung bình*

- Với input đầu vào này, việc chọn `MAX_PRIO` $= 7$ là không tối ưu, do cả TT và WT ở trường hợp này đều tăng hơn so với trước. Tuy nhiên, với test case này, ta đã kiểm thử được cơ chế duyệt các hàng đợi bằng giá trị *slot time slice*. Từ đó, cần có thêm bước đánh giá input đầu vào để lựa chọn số lượng hàng đợi sử dụng cài đặt bộ định thời sao cho tối ưu.

Tham khảo:



Hình 1.10: *Biểu đồ Gantt ứng với input trên và MAX_PRIO=10*



Hình 1.11: *Biểu đồ Gantt ứng với input trên và MAX_PRIO=2*

1.3.1.3 Test case 3

Test case này sẽ kiểm thử bộ định thời đã hiện thực với nhiều CPU.

Input:

Nội dung tệp tin sched1	time slice	**Số lượng CPU**	**Số lượng quá trình**	MAX_PRIO
5 3 6 0 p1s 2 1 p2s 1 2 p3s 0 4 p4s 1 10 p5s 2 12 p6s 1	5	3	6	140

Bảng 1.7: *Tệp tin mô tả và giải thích các thông số của test case*

Tệp tin	p1s	p2s	p3s	p4s	p5s	p6s
Nội dung	1 10 calc calc calc calc calc calc calc calc calc calc	20 6 calc calc calc calc calc calc	7 17 calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc calc	20 13 calc calc calc calc calc calc calc calc calc calc calc	6 5 calc calc calc calc calc	9 8 calc calc calc calc calc calc calc calc

Bảng 1.8: *Các tệp tin chứa nội dung các quá trình sử dụng trong test case*

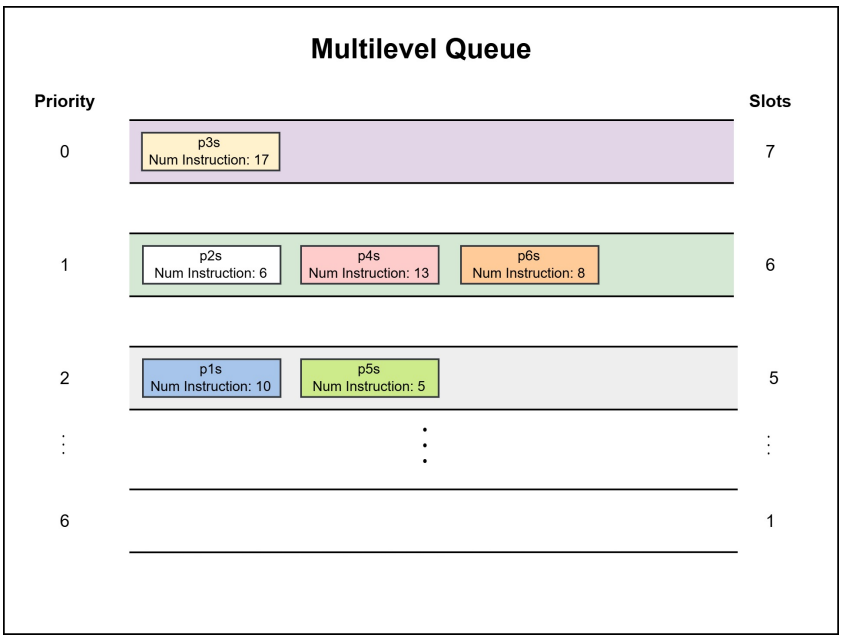
Output:

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p1s, PID: 1 PRIO: 2
Time slot   1
        CPU 2: Dispatched process  1
        Loaded a process at input/proc/p2s, PID: 2 PRIO: 1
Time slot   2
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 0
Time slot   3
        CPU 1: Dispatched process  3
Time slot   4
        Loaded a process at input/proc/p4s, PID: 4 PRIO: 1
Time slot   5
Time slot   6
        CPU 2: Put process  1 to run queue
        CPU 2: Dispatched process  4
Time slot   7
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   8
        CPU 0: Processed  2 has finished
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
        CPU 0: Dispatched process  1
Time slot   9
Time slot  10
         Loaded a process at input/proc/p5s, PID: 5 PRIO: 2
Time slot  11
        CPU 2: Put process  4 to run queue
        CPU 2: Dispatched process  4
Time slot  12
        Loaded a process at input/proc/p6s, PID: 6 PRIO: 1
```

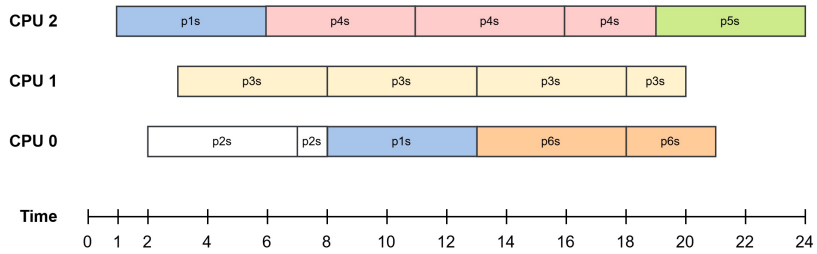
```
Time slot  13
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  6
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot  14
Time slot  15
Time slot  16
        CPU 2: Put process  4 to run queue
        CPU 2: Dispatched process  4
Time slot  17
Time slot  18
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  6
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot  19
        CPU 2: Processed  4 has finished
        CPU 2: Dispatched process  5
Time slot  20
        CPU 1: Processed  3 has finished
        CPU 1 stopped
Time slot  21
        CPU 0: Processed  6 has finished
        CPU 0 stopped
Time slot  22
Time slot  23
Time slot  24
        CPU 2: Processed  5 has finished
        CPU 2 stopped
```

Hình 1.12: *Kết quả khi thực thi chương trình với test case sched1*

Phân tích:



Hình 1.13: *Mô phỏng cấu trúc Multilevel Queue ban đầu*



Hình 1.14: *Biểu đồ Gantt ứng với output*

- Tại *time=0*: Quá trình `p1s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=1*:
  - Quá trình `p2s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
  - Bộ định thời thực hiện tìm quá trình sẽ được nạp vào CPU2 và cấp phát tài nguyên. Lúc này, chỉ có duy nhất `p1s` nên đây sẽ là quá trình được chọn, thực thi trong 5 time slice.

  Hai tác vụ trên là song hành với nhau do thực thi trên hai luồng khác nhau.
- Tại *time=2*:
  - Quá trình `p3s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
  - Bộ định thời thực hiện tìm quá trình sẽ được nạp vào CPU0 và cấp phát tài nguyên. Lúc này, chỉ có duy nhất `p2s` nên đây sẽ là quá trình được chọn, thực thi trong 5 time slice.
- Tại *time=3*: Bộ định thời thực hiện tìm quá trình sẽ được nạp vào CPU0 và cấp phát tài nguyên. Lúc này, chỉ có duy nhất `p3s` nên đây sẽ là quá trình được chọn, thực thi trong 5 time slice.
- Tại *time=4*: Quá trình `p4s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=6*:
  - Quá trình `p1s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU2 và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=1* có quá trình `p4s` nên đây sẽ là quá trình được chọn, thực thi trong 5 time slice.
- Tại *time=7*:
  - Quá trình `p2s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU0 và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=1* có quá trình `p2s` nên đây sẽ là quá trình được chọn, thực thi trong 1 time slice trước khi hoàn thành.

- Tại *time=8*:
  - Quá trình `p2s` dừng và được giải phóng do đã thực thi xong.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU0 và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=3* có quá trình `p1s` nên đây sẽ là quá trình được chọn, thực thi trong 5 time slice.
  - Quá trình `p3s` dừng và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU1 và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=1* có quá trình `p3s` nên đây sẽ là quá trình được chọn, thực thi trong 5 time slice.
- Tại *time=10*: Quá trình `p5s` xuất hiện và được thêm vào hàng đợi tương ứng, mất 1 time slice.
- Tại *time=11*:
  - Quá trình `p4s` dừng và và được thêm lại vào hàng đợi.
  - Bộ định thời tìm quá trình tiếp theo sẽ được nạp vào CPU2 và cấp phát tài nguyên. Lúc này, tại đầu hàng đợi *prio=0* có quá trình `p4s` nên đây sẽ là quá trình được chọn, thực thi trong 5 time slice.

Bộ định thời sẽ tiếp tục hoạt động như vậy cho phần còn lại của các quá trình.

Nhận xét:

- Kết quả thu được là phù hợp và đúng với lý thuyết, dự đoán.
- Turnaround time, waiting time của từng quá trình cũng như giá trị trung bình:

Quá trình	**Turnaround time** (time slice)	**Waiting time** (time slice)
`p1s`	$TT_1 = 13 - 1 = 12$	$WT_1 = 12 - 10 = 2$
`p2s`	$TT_2 = 8 - 2 = 6$	$WT_2 = 6 - 6 = 0$
`p3s`	$TT_3 = 20 - 3 = 17$	$WT_3 = 17 - 17 = 0$
`p4s`	$TT_4 = 19 - 6 = 13$	$WT_4 = 13 - 13 = 0$
`p5s`	$TT_5 = 24 - 19 = 5$	$WT_5 = 5 - 5 = 0$
`p6s`	$TT_6 = 21 - 13 = 8$	$WT_6 = 8 - 8 = 0$
	$\overline{TT} = 10.17$	$\overline{WT} = 0.33$

TT = thời điểm kết thúc – thời điểm ban đầu
$WT = TT$ – thời gian thực thi

Bảng 1.9: *Turnaround time, waiting time của từng quá trình và giá trị trung bình*

- Việc sử dụng nhiều CPU để thực thi các quá trình cho thấy sự hiệu quả rõ rệt về tốc độ cũng như thời.