ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA


# HỆ ĐIỀU HÀNH (CO2018)

## LAB 4
## SCHEDULING & CONTIGUOUS MEMORY ALLOCATION

GVHD: Nguyễn Quang Hùng

Sinh viên: Hoàng Tiến Hải - 2011152

Tp. Hồ Chí Minh, Tháng 4/2022

# Mục lục

# 1 INTRODUCTION

# 2 PRACTICE

# 3 EXERCISES

## 3.1 Problem1

Write two programs for CPU scheduling using **FCFS** and **Round Robin**. Measure the total turnaround time for all processes.(read the section 1.3)

SOLVE

– Đầu tiên chúng ta sẽ hoàn thiện các hàm để hiện thực một queue hoàn chỉnh trước.

```
struct pcb_t * de_queue(struct pqueue_t * q) {
        struct pcb_t * proc = NULL;
        // TODO: return q->head->data and remember to update the queue's
        // head and tail if necessary. Remember to use 'lock' to avoid
        // race condition

        // YOUR CODE HERE
        if (q->head != NULL)
        {
                pthread_mutex_lock(&q->lock);
                proc = q->head->data;
                if (q->head != q->tail) q->head = q->head->next;
                else {
                        q->head = NULL;
                        q->tail = NULL;
                }
                pthread_mutex_unlock(&q->lock);
        }

        return proc;
}
```

– Thực hiện lấy một phần tử bên trong hàng đợi ra ngoài, ta sẽ xét xem hàng đợi đó đã có phần tử nào hay chưa. Nếu có sẽ tiến hành khoá mutex và thực hiện lấy phần tử đó ra khỏi queue. Tránh race condition với việc thêm vào hàng đợi bằng mutex lock. Nếu đã lấy hết phần tử trong hàng đợi ra ngoài thì head và tail của hàng đợi sẽ trả về NULL.

– Tương tự với khi thêm vào hàng đợi một phần tử:

```
void en_queue(struct pqueue_t * q, struct pcb_t * proc) {
        // TODO: Update q->tail.
        // Remember to use 'lock' to avoid race condition

        // YOUR CODE HERE

        struct qitem_t * block = (struct qitem_t *)malloc(sizeof(struct qitem_t));
        block->data = proc;
        block->next = NULL;


        pthread_mutex_lock(&q->lock);
```

```
        if (q->head == NULL)
        {
                q->head = block;
                q->tail = block;
        }
        else{
                q->tail->next = block;
                q->tail = block;
        }

        pthread_mutex_unlock(&q->lock);
}
```

– Tạo một biến *block* để chứa dữ liệu kiểu qitem_t sẽ được thêm vào hàng đợi. Khi tiến hành thêm phần tử vào hàng đợi, sử dụng mutex lock để đảm bảo đồng bộ giữa các quá trình với nhau tránh xung đột dữ liệu khi truy xuất.

– Tiến hành chỉnh sửa chương trình trong file *sched.c*

- Round Robin

```
// TODO: Calculate exec_time from process's PCB
if (proc->burst_time > timeslot)
{
        exec_time = timeslot;
        proc->burst_time -= timeslot;
}
else
{
        exec_time = proc->burst_time;
        proc->burst_time = 0;
}
```

– Và

```
// TODO: Check if the process has terminated (i.e. its
// burst time is zero. If so, free its PCB. Otherwise,
// put its PCB back to the queue.

if (proc->burst_time == 0)
{
        printf("Turnaround time of process %d: %d\n", id, timestamp - proc->arri
        free(proc);
}
else
{
        en_queue(&ready_queue, proc);
}

// YOUR CODE HERE

/* Track runtime status */
printf("%2d--%2d: Execute %d\n", start, timestamp, id);
```

– Cách tính *exec_time* khi sử dụng phương pháp round robin: Nếu burst time của process đang được chạy không được xử lý hết trong một chu kỳ timeslot (timeslice) thì sẽ được đưa lại về cuối

hàng đợi để xử lý tiếp cho lần sau. Hàng đợi tiếp tục lấy phần tử tiếp theo của hàng đợi. Burst time của process sẽ giảm qua thời gian chạy. Nếu burst time nhỏ hơn hoặc bằng timeslot thì process sẽ được xử lý hết, không cần đưa về cuối hàng đợi và giải phóng process đã được xử lý hết. Thời gian xoay vòng turnaround sẽ được tính bằng thời gian hoàn thành process trừ cho thời gian tới arrival.

```
pi@osboxes:~/lab4_os/Roundrobin $ echo "Input file:"
Input file:
pi@osboxes:~/lab4_os/Roundrobin $ cat input.txt
2 5
0 6
2 4
4 2
5 4
6 5
pi@osboxes:~/lab4_os/Roundrobin $ echo "Results"
Results
pi@osboxes:~/lab4_os/Roundrobin $ gcc sched.c queue.c -o sched -lpthread
pi@osboxes:~/lab4_os/Roundrobin $ cat input.txt | ./sched
 1- 3: Execute 0
 3- 5: Execute 0
 5- 7: Execute 1
Turnaround time of process 0: 9
 7- 9: Execute 0
Turnaround time of process 2: 7
 9-11: Execute 2
Turnaround time of process 1: 11
11-13: Execute 1
13-15: Execute 3
15-17: Execute 4
Turnaround time of process 3: 14
17-19: Execute 3
19-21: Execute 4
Turnaround time of process 4: 16
21-22: Execute 4
```

- FCFS
  - Tương tự như hiện thực round robin, FCFS được làm như sau:

```
// TODO: Calculate exec_time from process's PCB
exec_time += proc->burst_time;
proc->burst_time = 0;
```

  - Và:

```
// TODO: Check if the process has terminated (i.e. its
// burst time is zero. If so, free its PCB. Otherwise,
// put its PCB back to the queue.

if (proc->burst_time == 0)
{
        printf("Turnaround time of process %d: %d\n", id, timestamp - proc->arri
        free(proc);
}

// YOUR CODE HERE

/* Track runtime status */
printf("%2d-%2d: Execute %d\n", start, timestamp, id);
```

  - Exec_time của phương pháp FCFS sẽ được tính khác và đơn giản hơn so với round robin. Một process đến trước sẽ được xử lý trước cho đến khi nào thực hiện xong process đó mới chuyển qua process khác và không có sự xen lẫn và nhường CPU cho các process khác.

```
pi@osboxes:~/lab4_os/FCFS $ echo "Input file:"
Input file:
pi@osboxes:~/lab4_os/FCFS $ cat input.txt
2 5
0 6
2 4
4 2
5 4
6 5
pi@osboxes:~/lab4_os/FCFS $ echo "Results"
Results
pi@osboxes:~/lab4_os/FCFS $ gcc sched.c queue.c -o sched -lpthread
pi@osboxes:~/lab4_os/FCFS $ cat input.txt | ./sched
Turnaround time of process 0: 7
 1- 7: Execute 0
Turnaround time of process 1: 9
 7-11: Execute 1
Turnaround time of process 2: 9
11-13: Execute 2
Turnaround time of process 3: 12
13-17: Execute 3
Turnaround time of process 4: 16
17-22: Execute 4
pi@osboxes:~/lab4_os/FCFS $
```

## 3.2 Problem2