HCMC University Of Technology
Faculty of Computer Science & Engineering

Lab 4 Scheduling
Course: Operating Systems

Author: Toan V. Tran, PT Kien, Phuong-Duy N, Duc-Hai N, Minh TC
Email: viettoantran98@gmail.com

April 15, 2021

Goal: This lab helps student to practice the scheduling algorithms in Operating System.

Content In detail, this lab requires student implement one of the scheduling algorithms from the given source codes. We use these source code to emulate the scheduling algorithm using **FIFO**. For other algorithms, student will practice with exercises by hand., including:

- SJF
- Priority Scheduling
- Round-Robin

Result After doing this lab, student can understand the principle of each scheduling algorithm in practice.

Requirement Student need to review the theory of scheduling.

1 Workload Assumptions

Before getting into the range of possible policies, let us first make a number of simplifying assumptions about the processes running in the system, sometimes collectively called the **workload**. Determining the workload is a critical part of building scheduling policies.

We will make the following assumptions about the processes, sometimes called jobs, that are running in the system:

1. Each job runs for the same amount of time.
2. All jobs only use the CPU (i.e., they do not perform I/O)
3. The run-time of each job is known.

Scheduling criteria:

- CPU utilization
- Throughput
- Turnaround time: The interval from the time of submission of a process to the time of completion is the turnaround time.

$$T_{turnaround} = T_{completion} - T_{arrival} \tag{1.1}$$

- Waiting time: It affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.
- Response time: The time from the submission of a request until the first response is produced. This is called response time, is the time it takes to start responding, not the time it takes to output the response.

$$T_{response} = T_{firstrun} - T_{arrival} \tag{1.2}$$

2 Scheduling algorithms

2.1 First In, First Out (FIFO)

The most basic algorithm a scheduler can implement is known as **First In, First Out (FIFO)** scheduling or sometimes **First Come, First Served**. FIFO has a number of positive properties: it is clearly very simple and thus easy to implement. Given our assumptions, it works pretty well.

Process	Arrive	Burst Time
P1	0	24
P2	0	3
P3	0	3

If the processes arrive in the order P1, P2, P3, and are served in FCFS order, we get the result shown in the following Gantt chart, which is a bar chart that illustrates a particular schedule, including the start and finish times of each of the participating processes:

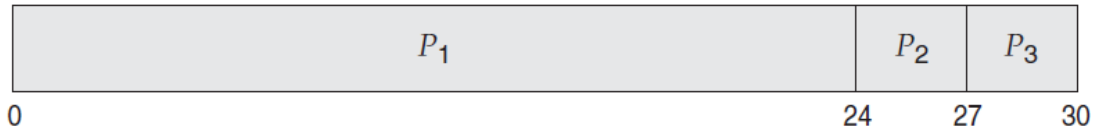

Figure 2.1: FIFO algorithm.

2.2 Shortest Job First Scheduling

A different approach to CPU scheduling is the **shortest-job-first** (**SJF**) scheduling algorithm. This algorithm associates with each process the length of the process's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst.

As an example of SJF scheduling, consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Arrive Time	Burst Time
P1	0	6
P2	0	8
P3	0	7
P4	0	3

Using SJF scheduling, we would schedule these processes according to the following Gantt chart:

2.3 Priority scheduling

The **SJF** algorithm is a special case of the general priority-scheduling algorithm. A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.

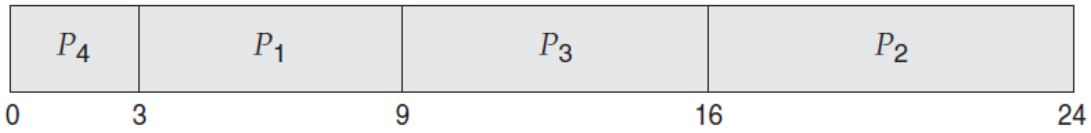

Figure 2.2: SJF algorithm.

2.4 Round-robin scheduling

The round-robin (RR) scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes. A small unit of time, called **a time quantum** or **time slice**, is defined. A time quantum is generally from 10 to 100 milliseconds in length.

3 Exercise

In this lab, you are given a source code that is described as bellow. The main purpose of the source code is to simulate CPU scheduling algorithms. This lab requires students implementing scheduling algorithms including "Shortest Job First", "Priority Scheduling", and "Round Robin". Students are allowed to change anything in the source code, but do not change in print function to make sure correct outputs.

Process structure In the source code, a process is defined as:

```
typedef struct _process_{
    char id[256];
    int arrive_time;
    int burst;
    int tmp_burst_;
    int priority;
    int return_time;
    int waiting_time;
    int turnaround_time;
    int response_time;
    bool completed;
    bool executing;
}Process;
```

Load processes The input of processes is loaded from "process.txt". For example:

```
4
P0 2 2 2
P1 0 3 3
```

```
P2  4  2  2
P3  10  3  4
2
```

in which, the first number (4) is the number of processes. The next 4 lines contains information of processes, each line for a process. For example, in the second line, P0, 2, 2, and 2 are the id of the process, the arrive time, the burst time and the priority, respectively. The last number (2) in the last line is the quantum time.

TASKS The source code autonomously loads processes and sorts them in the order of arrive time. Your tasks is to implement algorithms in file "schedulers.h". Uncompleted functions include SJF(), Priority_Scheduling(), RR(). Note that you can change other files if you want. These function will input a pointer p as Process* and a number of processes. After these function, processes of the array (which p point to) are updated to correct values including:

- return_time: when the process finished
- waiting_time: total waiting time of the process
- response_time: the time from arriving to the first execution time.
- turnaround_time: the time from arriving to completing.

SUBMISSION **Any cheating code detected will get 0 score. Students are required to show their running program during lab hours.**

NOTE: As these exercises are graded automatically, thereby, you need to implement the program by the requirements mentioned above. you must compress all of files (.c, .h, Makefile) into a zip file as:

```
Program
├── Makefile
├── main.c
└── process.h
```