HCMC University Of Technology
Faculty of Computer Science & Engineering

Lab 7 Scheduling
Course: Operating Systems

April 9, 2018

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 arrive at the same time.
3. All jobs only use the CPU (i.e., they perform no I/O)
4. 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	Burst Time
P1	24
P2	3
P3	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:

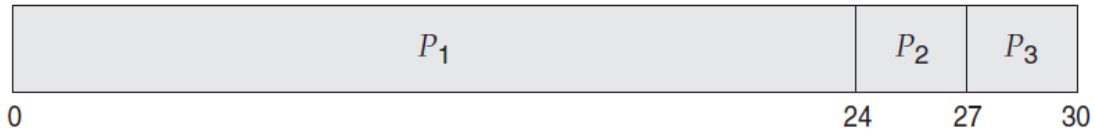

Figure 2.1: FIFO algorithm.

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	Burst Time
P1	6
P2	8
P3	7
P4	3

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

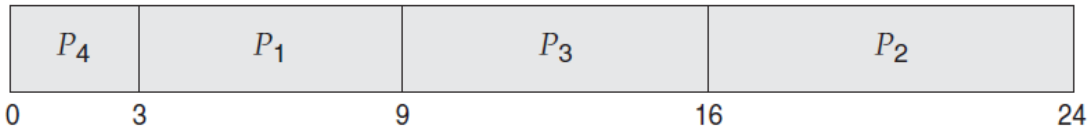

Figure 2.2: SJF algorithm.

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.

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 (Required)

1. Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

a) What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b) What is the average turnaround time for these processes with the SJF scheduling algorithm?

c) The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be called future-knowledge scheduling.

2. Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P1	8	4
P2	6	1
P3	1	2
P4	9	2
P5	3	3

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority), and RR (quantum = 1). Calculate the average waiting time and turnaround time of each scheduling algorithm.

Note: Please remember to write your answer to a single PDF file and name it after your MSSV before submitting this file to Sakai.

A. Bonus

This lab requires student implement a scheduling algorithm at class, **FIFO**. You are marked at the class with the given source code and only need to write your code in "*// TO DO*" parts. The source code emulates the scheduling process of OS using **FIFO** or (**FCFS**). This process is illustrated below:

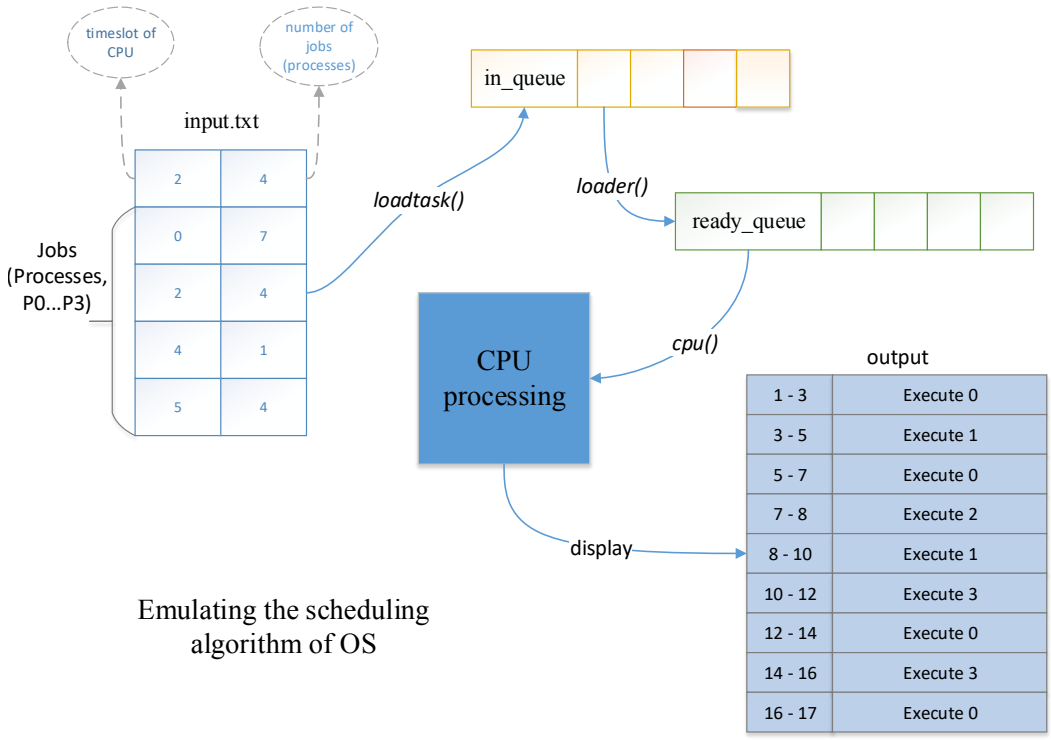

Figure A.1: Model of emulating the scheduling algorithm in OS.

In the source code, you need to understand the content of each source file. With the `main()` function in `sched.c`, we need to do;

```
1  ////////// sched.c ////////////
2
3  #include "queue.h"
4  #include <pthread.h>
5  #include <unistd.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  #define TIME_UNIT        100 // In microsecond
10
```

```
11 static struct pqueue_t in_queue; // Queue for incomming processes
12 static struct pqueue_t ready_queue; // Queue for ready processes
13
14 static int load_done = 0;
15
16 static int timeslot; // The maximum amount of time a process is allowed
17 // to be run on CPU before being swapped out
18
19 // Emulate the CPU
20 void * cpu(void * arg);
21
22 // Emulate the loader
23 void * loader(void * arg);
24
25 /* Read the list of process to be executed from stdin */
26 void load_task();
27
28 int main(){
29         pthread_t cpu_id;        // CPU ID
30         pthread_t loader_id;     // LOADER ID
31
32         /* Initialize queues */
33         initialize_queue(&in_queue);
34         initialize_queue(&ready_queue);
35
36         /* Read a list of jobs to be run */
37         load_task();
38
39         /* Start cpu */
40         pthread_create(&cpu_id, NULL, cpu, NULL);
41         /* Start loader */
42         pthread_create(&loader_id, NULL, loader, NULL);
43
44         /* Wait for cpu and loader */
45         pthread_join(cpu_id, NULL);
46         pthread_join(loader_id, NULL);
47
48         pthread_exit(NULL);
49 }
```

Note: You need to consider the data structure which is declared in `structs.h`. This file declares attributes of `process` and `queue`.

```
1  /////////// structs.c ////////////
2  #ifndef STRUCTS_H
3  #define STRUCTS_H
4
5  #include <pthread.h>
6
7  /* The PCB of a process */
8  struct pcb_t {
9          /* Values initialized for each process */
10         int arrival_time; // The timestamp at which process arrives
11                           // and wishes to start
12         int burst_time; // The amount of time that process requires
13                          // to complete its job
14         int pid;        // process id
15 };
16
17 /* 'Wrapper' of PCB in a queue */
18 struct qitem_t {
19         struct pcb_t * data;
20         struct qitem_t * next;
21 };
22
23 /* The 'queue' used for both ready queue and in_queue (e.g. the list of
24  * processes that will be loaded in the future) */
25 struct pqueue_t {
26         /* HEAD and TAIL for queue */
27         struct qitem_t * head;
28         struct qitem_t * tail;
29         /* MUTEX used to protect the queue from
30          * being modified by multiple threads*/
31         pthread_mutex_t lock;
32 };
33
34 #endif
```

Finally, that is the source file used to implement the `queue` operations, such as `en_queue()`, `de_queue()`.

```
1  /////////// queue.h ////////////
2
3  #ifndef QUEUE_H
4  #define QUEUE_H
5
6  #include "structs.h"
7
```

```
 8 /* Initialize the process queue */
 9 void initialize_queue(struct pqueue_t * q);
10
11 /* Get a process from a queue */
12 struct pcb_t * de_queue(struct pqueue_t * q);
13
14 /* Put a process into a queue */
15 void en_queue(struct pqueue_t * q, struct pcb_t * proc);
16
17 int empty(struct pqueue_t * q);
18
19 #endif
```