Programming Assignment #3
CPU Scheduling

Spring, 2022

Deadline: May 16th before 24:00

Overview

Write a C-program to implement a simulator with various CPU scheduling algorithms discussed in our classes.

The selected scheduling algorithms to implement in this assignment are

1) First Come First Serve (**FCFS**)
2) Round Robin (**RR**)
3) Preemptive Priority Scheduling with Aging

Outline

- In this assignment you will write your own scheduler (to simulate a real process scheduler)
  - The simulator selects a task to run from ready queue based on the scheduling algorithm.
  - Since the project intends to **simulate** a CPU scheduler, it does **not** require any actual process creation or execution. When a task is scheduled, the simulator will simply print out what task is selected to run at a time.
- The output of your simulator will be similar to a Gantt chart.

Outline

You will have to complete this assignment using the C language in Linux command line environment.

- since most existing operating system kernels are written in C/C++.

This assignment also aims to improve your C programming skill.

- Learn how to use “command line arguments” in the C

가천대학교 Gachon University

Detail

Design Hint

It is recommended that you start this assignment by defining a data structure similar to a **Process Control Block (PCB)** for each task.

You need to implement **linked lists** as the process queue structure.

System Model

Assume that there are **10 processes** in the **job queue** of your (virtual) system.

Initially, all the processes (10 processes) are in NEW state.

A process may enter the **ready queue** if it arrives according to the task information which is determined by user's input

The input is delivered through a "file" that specifies each process's arrival time and CPU burst requests.

1. User's Input

The task (process) information will be read from an input file.

The file format:

- Each line of the file has the following format:

```
pid  priority  arrival_time  burst_time
```

All of fields are integer types, where

- **pid** is a unique numeric process ID (**1** to **10**)
  - (maximum value is 10)
- **priority** is an integer value
  - A larger value implies a higher priority
- **arrival_time** is the time when the task arrives in the unit of milliseconds
- **burst_time** the is the CPU time requested by a task, in the unit of milliseconds

Assume the time units for **arrival_time**, **burst_time** and **interval** are in millisecond units.

Sample input : input.dat

```
1      50      10      50
2      50      0       40
3      30      20      50
5      10      7       35
```

2. Requirements

Simulate (i.e., implement) the following three CPU scheduling algorithms

- First-Come, First-Served (FCFS) Scheduling
- Round-Robin (RR) Scheduling
- **Preemptive** Priority Scheduling (with Aging)

The program should schedule tasks and print progress of task every unit time (millisecond) (as shown in sample outputs)

For each algorithm, calculate

1) Average Waiting time of each process (**pid**)
2) Average Response time of each process (**pid**)
3) Average Turnaround time of each process
4) overall CPU usage (0~100%)

3. Output

Print time-flow (like Gantt chart) and statistical information for each algorithm

Sample output (the output is saved into a file or printed out on the screen)

```
Scheduling : FCFS
==========================================
<time 0> ---- system is idle ----
<time 1> ---- system is idle ----
<time 2> [new arrival] process 1
<time 2> process 1 is running
<time 3> process 1 is running
<time 4> process 1 is running
<time 5> process 1 is running
<time 6> [new arrival] process 2
<time 6> process 1 is running
<time 7> process 1 is finished
------------------------------- (Context-Switch)
<time 8> process 2 is running
…
<time 50> all processes finish
==========================================
Avarage cpu usage : 92.00 %
Avarage waiting time :  15.2
Avarage response time : 12.1
Avarage turnaround time: 21.0
```

```
Scheduling : RR
==========================================
<time 0> ---- system is idle ----
<time 1> ---- system is idle ----
<time 2> [new arrival] process 1
<time 2> process 1 is running
<time 3> process 1 is running
<time 4> process 1 is running
<time 5> process 1 is running
<time 6> [new arrival] process 2
<time 6> process 1 is running
<time 7> process 1 is finished
------------------------------- (Context-Switch)
<time 8> process 2 is running
…
<time 50> all processes finish
==========================================
Avarage cpu usage : 92.00 %
Avarage waiting time :  7.2
Avarage response time : 7.1
Avarage turnaround time: 18.0
```

4. Details on Priority Scheduling

- The priority level of a process is represented by the priority field
  - (**pid priority arrival_time burst_time**)

  The priority scheduler selects the process with the highest priority value to run next.

  Aging

  To prevent the starvation, you will implement an aging scheme. The priority of a process varies over time according to:

The priority of a process =
Base priority + f(waiting time **t** of the process)

f(**t**) = **alpha** * **t**

1’. User’s input

Task(job) Information is delivered to your simulator using “command line arguments”

In the C, it is very common to use the command line arguments for parameter configuration and/or acceptance of user’s input.

The usage are as follow:

```
your_sched_name   [input_filename] [output_filename]
                  [time_quantum_for_RR] [alpha_for_PRIO]
```

Example:

```
> os22 scenario1.dat output1.txt  5  0.2
```

```
1  50 10  50
2  50 0   40
3  30 20  50
5  10 7   35
```

Output file name

Time quantum for Round Robin scheduler

Alpha value for aging in priority scheduling

Command line argument in the C ?

A way for accepting program's arguments in C using `argc` and `argv`

Your main() function may look like:

```
int main ( int argc, char *argv[] )
```

- the **arg**ument **c**ount.
- It is the number of arguments passed into the program from the command line, including the name of the program.

char* argv[]

- the listing of all the arguments.
- argv[0] is the name of the program, or an empty string if the name is not available. After that, every element number less than argc is a command line argument.
- You can use each argv element just like a string, or use argv as a two dimensional array. argv[argc] is a null pointer.

Example

```
> os22 scenario1.dat output1.txt  5  0.2
```

argc = ?

argv[3] = ?

가천대학교 Gachon University

Command line argument in the C ?

Example code

```
#include <stdio.h>

int main ( int argc, char *argv[] )
{

    if ( argc != 2 ) /* argc should be 2 for correct execution */
    {
        /* We print argv[0] assuming it is the program name */
        printf( "usage: %s filename", argv[0] );
    }
    else
    {
        // We assume argv[1] is a filename to open
        FILE *file = fopen( argv[1], "r" );

        /* fopen returns 0, the NULL pointer, on failure */
        if ( file == 0 )
        {
            printf( "Could not open file\n" );
        }
        else
        {
            int x;
            /* read one character at a time from file, stopping at EOF, which
               indicates the end of the file.  Note that the idiom of "assign
               to a variable, check the value" used below works because
               the assignment statement evaluates to the value assigned. */
            while  ( ( x = fgetc( file ) ) != EOF )
            {
                printf( "%c", x );
            }
            fclose( file );
        }
    }
}
```

5. Design Hints

Recommendation

You can use a data structure similar to a Process Control Block (PCB) for each task, though yours will be much simpler.

Two queues are maintained; one for the job queue and one for the ready queue

- The job queue may be implemented simply with an Array!
- and the ready queue with a linked-list

Simulator

The simulator first initializes the job queue and the data structure of PCB information for 10 processes (which are set to be in the NEW state – this is our assumption)

The simulator then reads the task information from the **input file** and stores all data in a data structure.

Then, start simulating a scheduling algorithm in a **time-driven** manner.

At each time unit (or slot), it adds any newly arrived task(s) into the ready queue and runs a specific scheduler algorithm in order to select appropriate task from ready queue.

When a task is chosen to run, the simulator prints out a message indicating what process ID is chosen to execute for this time slot. If no task is running (i.e. empty ready queue), it prints out an “idle” message.

Before advancing to the next time unit, the simulator should make all necessary changes in job and ready queue status.

Submission

- Program Report using POWER-POINT(PPT)
  - **3-5 pages explaining your Program (data structure, functions etc.)**
  - **Screen capture of execution for 3 or more command-line input strings (입력)**
  - **Last page**
    - **Performance results (성능평가) of each scheduling algorithm**
      - **waiting time, …**
    - **You may use graphs to show your results**

- Submission
  - A zip file, which contains
    - **All source code files (.c)**, your test input files
    - PPT file
  - File naming: Student#_YourName.zip (ex: 20212232_홍길동.zip)

Grading policy

50 points (Approx. 5% of your semester grades).

- FCFS (20), RR (10), Preemptive Priority Scheduling with Aging (10), Documentation(PPT) (10)

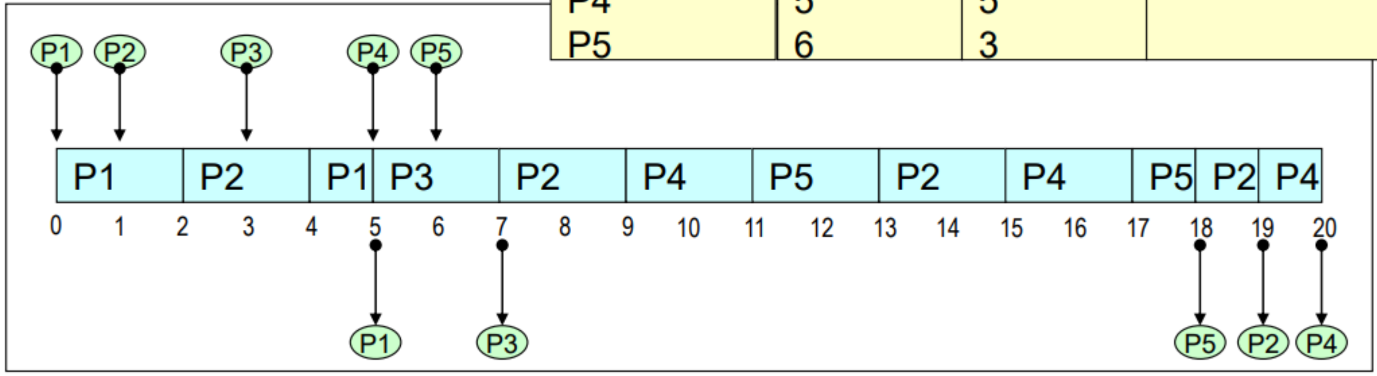
Degradation

- If you do not follow the requirements:
  - File naming, contents (-10%)
- Delayed submission: at least -50% points
- Compile error : -50%
- **Copied from previous year submissions, classmates: 0 points**
- Individual assignment ! (No collaboration)

Appendix (RR)

Example:
RR Scheduling

Process ID	Arrival time	Burst time	Wait Time
P1	0	3	
P2	1	7	
P3	3	2	
P4	5	5	
P5	6	3	



Gantt Chart

Questions

Contact the TA for questions

김동민: ihaha424@gachon.ac.kr

임채윤: lcu1027@naver.com