Spring 2022 COMP 3511 Homework Assignment 2 (HW2)

Handout Date: Mar. 11, 2022, Due Date: Mar. 25, 2022

Name	
Student ID	
ITSC email	@connect.ust.hk

Please read the following instructions carefully before answering the questions:

- You should finish the homework assignment **individually**.
- This homework assignment contains **two** parts:
  1) Multiple choices, 2) Question & Answer
- All programs should be executed in a CS Lab 2 machine.
- **Homework Submission:** submitted to **Homework #2** on **Canvas**.
- TA responsible for HW2: Decang Sun (dsunak@cse.ust.hk)

1. (20 points) Multiple choices.

Please write down your answers in the boxes below:

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
D	C	A	C/D	C	D	B	C	B	D

1) Which of the following is the correct description of the Process Control Block (PCB)?

   A. It is an example of a process queue
   B. The address of the next instruction to be processed by a different process is stored in the PCB
   C. Determine the process to be executed next
   D. It contains information associated with a process

   Answer: D (see 3.10)

2) Under certain conditions, the process transits between states. Which of the following process state transitions does not occur?

   A. waiting state → ready state
   B. ready state → running state
   C. ready state → waiting state
   D. running state → waiting state

   Answer: C (see 3.28)

3) Run the following in our CS Lab 2 Linux machines:

```
#include <stdio.h>
#include <unistd.h>

int main(){
 pid_t pid;
 int a=100;
 pid = fork();
 if (pid==0)
      printf ("This is the child process, a=%d ", --a);
 else
      printf ("This is the parent process, a=%d ", ++a);
}
```

Assuming that the compilation and linking process is correct and the program is executed correctly, the running result is

A. This is the parent process, a=101 This is the child process, a=99
B. This is the child process, a=99
C. This is the parent process, a=101
D. This is the child process, a=101 This is the parent process, a=99
Answer: A (see 3.34)

4) Which of the following statements is true?

A. Named pipes are deleted immediately after the communication ends.
B. Only parent and child processes can communicate using named pipes.
C. Communication can be either directional or bidirectional for named pipes
D. Named pipes can only be used by communicating processes on the same machine.
Answer: C/D (see 3.56)
D: Named pipes on Windows or UNIX is different. If you choose this option, I will give you points. Remember to check 3.56.

5) Which of the following statements about processes and threads is correct?

A. Process is the basic unit of processor scheduling
B. Thread is the basic unit of resource allocation
C. Thread is the basic unit of processor scheduling
D. Threads can exist independently of processes
Answer: C (see 4.6)

6) Which of the following would be an acceptable signal handling scheme for a multithreaded program? _____.

A. Deliver the signal to the thread to which the signal applies.
B. Deliver the signal to every thread in the process.
C. Deliver the signal to only certain threads in the process.

D. All of the above
Answer: D (see 4.29)

7) The following are the function names and meanings provided by the pthread library commonly used in multithreaded programming under Linux. Which statement is wrong?

A. pthread_create: creates a thread
B. pthread_join: Multiple threads can call the pthread_join() function to get the execution result of the same thread
C. pthread_mutex_init: initializes a thread mutex
D. pthread_exit: ends a thread
Answer: B (see 4.32)

8) Which of the following process scheduling algorithm may lead to starvation?

A. FIFO
B. Round Robin
C. Shortest Job First
D. None of the above
Answer: C (see 5.23)

9) Assume process P0 and P1 are the process before and after a context switch, and PCB0 and PCB1 are the process control block of P0 and P1 respectively. Which of the following actions is NOT handed by the dispatcher?

A. Save state into PCB0, and restore state from PCB1 (switching context)
B. Selecting a process among the available ones in the ready queue
C. Switching to user mode
D. Jumping to the proper location in the user program to resume that program
Answer: B (see 5.7)

10) Which of the following statements about the MLFQ scheduling algorithm is correct?

A. Process cannot move from a low-level queue up to a high-level queue
B. Scheduling algorithms for each individual queue should be the same
C. Similar to SJF, it also needs knowledge of the next CPU burst time in advance
D. It could well handle interactive jobs by delivering similar performance as that of SJF or SRTF
Answer: D

11) **(25 points) Short answer. Please answer the following questions in several clear sentences.**

a. (5 points) Explain the concept of context switching in multithreaded process and how the kernel handles when a context switch occurs?

Answer: Whenever the CPU starts executing a new process, the old process's state must be preserved. The context of a process is represented by its process control block. Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.(3 points) This task is known as a context switch. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saves context of the new process scheduled to run.(2 points) (see 4.14)

b. (3 points) Why should a web server not run as a single-threaded process?

Answer: For a web server that runs as a single-threaded process, only one client can be serviced at a time. This could result in potentially enormous wait times for a busy server. (see 4.4)

c. (5 points) According to Amdahl's Law, what is the speedup gain for an application that is 25% parallel and we run it on a machine with 4 processing cores? What about running on 10 processing cores?

Answer: With 4 cores, S=0.75 and N=4.

$$speedup \leq \frac{1}{S + \frac{1-S}{N}} = \frac{1}{0.75 + \frac{1-0.75}{4}} = 1.23(16/13)$$

With 10 cores, S=0.75 and N=10.

$$speedup \leq \frac{1}{S + \frac{1-S}{N}} = \frac{1}{0.75 + \frac{1-0.75}{10}} = 1.29(40/31)$$

(see 4.12)

d. (5 points) What is the response time? What is the turnaround time? What is the difference between them?

Answer: Turnaround time is the sum of the periods that a process is spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O. Turnaround time essentially measures the amount of time it takes to execute a process. Response time, on the other hand, is a measure of the time that elapses between a request and the first response produced. (see 5.8)

e. (2 points) What are the disadvantages of the many-to-one model in the multi-threading model?

The main disadvantage of the many-to-one model is that if one thread blocks while accessing the core, the entire process is blocked. Multiple threads may not run in parallel on multicore system because only one kernel thread can be active at a time. (see 4.23)

Multiple threads may not be able to run in parallel because only one kernel thread can be active at a time. The waiting time could be long on average.

f. (5 points) Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

$\alpha = 0$ and $\tau_0 = 50$ milliseconds

$\alpha = 0.99$ and $\tau_0 = 20$ milliseconds

Answer:

When $\alpha = 0$ and $\tau_0 = 50$ milliseconds, the formula always makes a prediction of 50 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $\tau_0 = 20$ milliseconds, the most recent behaviour of the process is given much higher weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memoryless, and simply predicts the length of the previous burst for the next quantum of CPU execution. (see 5.19)

12) **(30 points) CPU Scheduling**

a. (10 points) This is a dynamically changing priority scheduling algorithm. In this algorithm, larger number means higher priority. The initial priority of all processes is 0. The formula of the scheduling algorithm is $P(t) = M \times (t - T0)$. T0 is the time at which the process joins the ready queue. When a process is in the CPU ready queue (not running), its priority changes at a rate of M. Similarly, when it is running, its priority changes at a rate N. We can modify parameters M and N to get different scheduling algorithms. (Hint: The process that requests the CPU later gets the CPU allocation first. We call it Last-come First-serve (LCFS))
1) What is the algorithm that results from N > M > 0?
2) What is the algorithm that results from M < N < 0?

Answer:
FCFS (First-come, first-serve). (2 points)
All the processes in the ready queue have the same inital priorty 0. Their priority increases at the same rate M (M >0). Thus, the earlier the process enters the ready quese(T0), the higher its priority will be (P(t) = M * (t - T0)).

Once the process with the highest priority (first-come compared to other process in the ready queue) is running, its priority increases at a higher rate (N> M >0) than the priorities of those processes in the ready queue. So, no other process will preempt it and it will run to its completion. (3 points)

LCFS (Last-come, first-serve). (2 points)
All the processes in the ready queue have the same initial priority 0. Their priority decreases at the same rate M (M <0). Thus, the earlier the process enters the ready quese(T0), the lower its priority will be (P(t) = M * (N - T0)).

Once the process with the highest priority (last-come compared to other process in the ready queue) is running, no other process in the ready queue will preempt it since its priority decreases at a lower rate (M < N <0) than the priorities of those processes in the ready queue. But it will be preempted by any new coming process because the new process's priority is 0 which is the highest possible priority. (3 points)

b. (20 points) Consider the following single-thread processes, arrival times, CPU time requirements, and priorities:

Process	Arrival Time	Burst Time	Priority
P_1	0	70	3
P_2	30	20	1
P_3	10	50	2
P_4	50	20	4

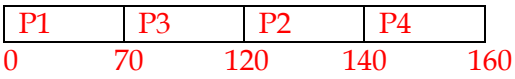
- We consider 5 algorithms: FCFS, SJF, SRTF, RR, Priority.
- The time quantum of the Round-Robin (RR) is 5.
- When priority is being used, a smaller priority number means higher execution priority
- There is no context switching overhead
- Whenever there is a tie among processors (same arrival time, same remaining time, etc), they are inserted into the ready queue in the ascending order of process id.
- The Priority in the last column **only** applies to the Priority (Preemptive) algorithm.
- *[Note: the GANTT charts will be used to determine partial credit if the values in the table are incorrect]*

For each scheduling algorithm, compute the average waiting time and average turnaround time. Please elaborate your answers with gantt charts.

Write your answers here

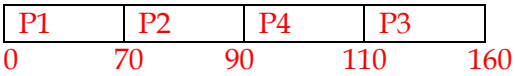
Scheduling Algorithm	Average waiting time	Average turnaround time
First Come First Served (FCFS)	60	100
Shortest Job First (SJF) (Non-preemptive)	45	85
Shortest Remaining Time First (Preemptive)	32.5	72.5
Round Robin (RR) (Time Quantum = 5)	70	110
Priority (Preemptive)	45	85

- **FCFS** (4 points)



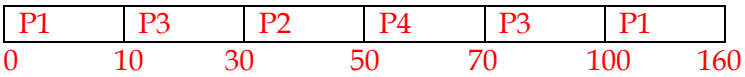
Avg waiting: {0 + (120-30) + (70-10) + (140-50)}/4 = 240/4=60
Avg turnaround: {70 + (140-30) + (120-10) + (160-50)}/4 = 400/4 = 100

- **SJF** (4 points)



Avg waiting: {0 + (70-30) + (110-10) + (90-50)}/4 = 180 /4 = 45
Avg turnaround: {70 + (90-30) + (160-10) + (110-50)}/4 = 340/4=85

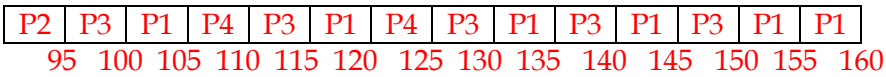
- **SRTF** (4 points)



Avg waiting: {(100-10) + 0 + (70-30) + 0} = 130/4=32.5

Avg turnaround: {160 + (50-30) + (100-10) + (70-50)} = 290/4=72.5

- **Round Robin (TQ=5)** (4 points)



Avg waiting

Method 1

P1: (20-15) + (30-25) + (45-35) + (60-50) + (80-65) + (100-85) + (115-105) + (130-120) + (140-135) + (150-145) = 90

P2: (35-30) + (50-40) + (70-55) + (90-75) = 45

P3: (15-10) + (25-20) + (40-30) + (55-45) + (75-60)+ (95-80) + (110-100) + (125-115) + (135-130) + (145-140) = 90

P4: (65-50) + (85-70) + (105-90) + (120-110) = 55

(90 + 45 + 90 + 55)/4 = 70

Method 2

Waiting time = finish time - arrival time - burst time P1: 160 - 0 - 70 = 90 P2: 95 - 30 - 20 = 45 P3: 150 - 10 - 50 = 90 P4: 125 - 50 - 20 = 55

Avg turnaround: {(160-0) + (95-30) + (150-10) + (125-50)}/4 = 440/4 = 110

- **Priority (preemtive)** (4 points)

Avg waiting: {(80-10) + 0 + (50-30) + (140-50)}/4 = 180/4=45

Avg turnaround: {140 + (50-30) + (80-10) + (160-50)}/4 = 340/4 = 85

13) **(10 points) Thread.**

Consider the following program (main.c) using the pthread library in C:

```
#include<stdlib.h>
#include<pthread.h>
#include<stdio.h>
void* f1(){
    printf("Nice day!\n");
    return NULL;
}
int main(){
    pthread_t th1;
    pthread_create(&th1,NULL,f1,NULL);
    return 0;
}
```

a. (2 point) What is the command to compile the above program using the gcc compiler to an executable file named as main? Remember, you need to link the pthread library

Answer: gcc main.c -o main -lpthread

b. (3 points) What is the output of the above code? please explain

Answer: no output, because when the code of the new thread has not been executed, the main thread has already executed return 0; After the main thread ends, other threads will end.

c. (5 points) How to fix this problem? You need to add a line of code in the blank. Please explain why this works. What is the output of the code after filling in the blanks?

```
#include<stdlib.h>
#include<pthread.h>
#include<stdio.h>
void* f1(){
    printf("Nice day!\n");
    return NULL;
}
int main(){
    pthread_t th1;
    pthread_create(&th1,NULL,f1,NULL);

    pthread_join(th1,NULL);        (2 points)        .

    return 0;
}
```

Answer: Output: Nice day! (1 point)
The pthread_join function can make the thread execute before executing the next code (2 points)

14) **(15 points) Fork and Pipe**

Consider the following C program (main.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int pd[2];
    pipe(pd);
    pid_t pid;
    pid = fork();

    if (pid == 0)
    {
        close(pd[0]);
        write(pd[1], "awesome!!", 8);
        close(pd[1]);
        exit(0);
    }

    close(pd[1]);
    char buf[10] = {0};
    read(pd[0], buf, 7);
    printf("COMP3511 is %s\n", buf);

    return 0;
}
```

a. (2 points) What is the output of the above code?
(8 points) Please explain the running logic of the above code in detail. [Possible keywords: parent process, child process, write side, read side, pd[0], pd[1]] **You can also draw several graphs to explain your answer.*

Answer: COMP3511 is awesome
Step1: The parent process creates the pipe
Step2: The parent process creates a child process
Step3: The pd[0] of the parent process and the child process point to one end of the pipe at the same time, and pd[1] points to the other end at the same time.
Step4: pd[0] of the child process disconnects from the pipe and writes data to pd[1].
Step5: The pd[1] of the parent process disconnects from the pipe and reads the data of pd[0]. At this time, pd[1] of the child process is connected to the write end of the pipe, and pd[0] of the parent process is connected to the read end of the pipe. This enables half-duplex communication between processes.

b. (3 points) What disadvantages does the above code show ordinary pipe?
(2 points) What is the possible solution?
Answer: Two processes and one pipe can only achieve half-duplex communication. Using two pipes can solve this problem one by one.