0.1 Practice Exercises

1. [2 points] why is it important for a scheduler to distinguish I/O-bound processes from CPU-bound processes?

Because the CPU-bound processes do not require OS services or IO operations, it means that CPU burst perform computation on the CPU, such as instructions executing or memory accessing. In the view of maximum the use of CPU, OS needs to do more CPU-bound processes.

IO-bound processes will be blocked until request complete, such as access a file on hard disk or from the internet. For those I/O burst, the CPU utilisation generally is not high.

As a result, if a scheduler can identify I/O-bound and CPU-bound processes, the scheduler can perform these different types of processes according to the utilisation of CPU. For example, if the current CPU utilisation is not enough, the scheduler can active those CPU-bound processes. Otherwise, the CPU can resume those I/O-bound processes.

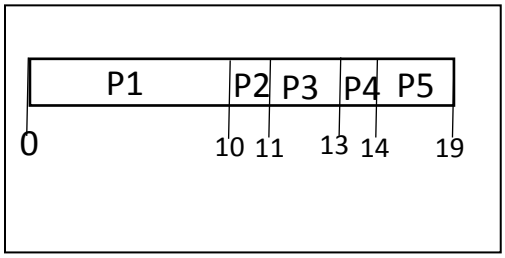
2. [8 points] consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

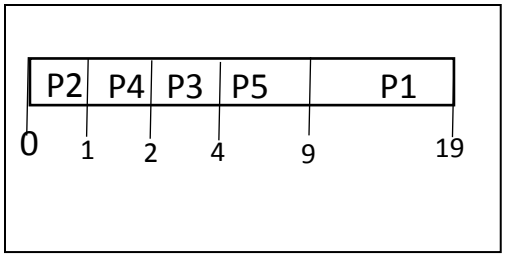
Assume the processes arrive in the following order: P1, P2, P3, P4, and P5.

a) [1.5 points] Draw three Gantt charts to illustrate the execution of these processes using FCFS, SJF, and RR (with quantum= 1)

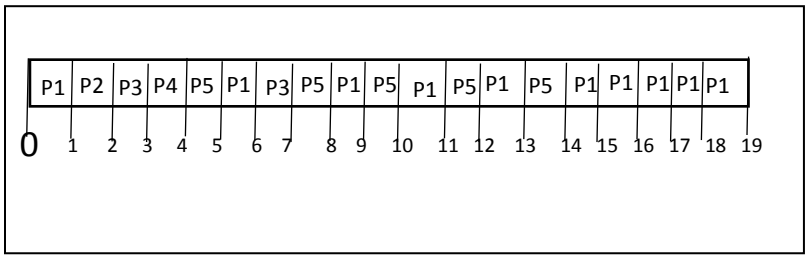
The Gantt chart for FCFS is:



The Gantt chart for SJF is:



The Gantt chart for RR is:



b) [3 points] What is the turnaround time of each process for each of the scheduling algorithms?

FCFS

Turnaround time: P1=10 P2=11 P3=13 P4=14 P5=19

SJF

Turnaround time: P1=19 P2=1 P3=4 P4=2 P5=9

RR

Turnaround time: P1=19 P2=2 P3=3 P4=4 P5=14

c) [3 points] What is the waiting time of each process for each of the scheduling algorithms?

FCFS

Waiting time: P1=0 P2=10 P3=11 P4=13 P5=14

SJF

Waiting time: P1=9 P2=0 P3=2 P4=1 P5=4

RR

Waiting time: P1=18 P2=1 P3=2 P4=3 P5=13

d) [0.5 points] which of the algorithms results in the minimum average waiting time (over all processes)?

FCFS = (0+10+11+13+14)/5 = 9.6

SJF= (9+0+2+1+4)/5 =3.2

RR= (18+1+2+3+13)/5 =7.4

As a result, the SJF has the minimum average waiting time.

3. [2 points] Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system?

It depends on the kernel thread model. In many-to-One model, many user threads mapped to a kernel thread, so no support for parallel execution on multiprocessors. In this situation, the performance will not be better.

But in One-to-One Model and Many-to-Many Model, process is not blocked when one thread makes a system call, so it's possible for multiple threads run in parallel on multiprocessors. In this situation, the performance will be improved.

4. [6 points] In addition to general-purpose server software, a number of classes of applications benefit directly from the ability to scale throughput with the number of threads and/cores. Give at least 3 classes of applications with examples.

The first example is the games. In order to get the best performance, all modern games are multithreaded in one way or another. We can classify the tasks in a game into render thread, audio thread, IO thread, network thread, tasks thread and etc.

The render thread focus on the display so it will own the display device and do anything about rendering. In PC, for example, the display driver support multithreading, this way can improve the performance significantly. In some mobile device, such as IOS or Android, there is some limitation about display driver, so it is more important the game APPs implement a multithread model.

Generally, the audio thread need to handle and own the audio device because typically playing background audio or others is not the most frequent events. Another reason why needs a separate audio thread is because the end user do not want to listen other sound during gaming, if there is a separate thread to handle the audio device, it will be better to control and remove the sound from other APPs.

IO and network tasks generally are done by separate threads as well. For example, some version Android do not support asynchronous IO, but there will be a couple of IO in a game, so it is a must for a game to separate the IO and network request and handling the responses.

The second example is office software such as Microsoft Office programs. In this type of application, there are some build-in features which need a multithread technology. For example, in a word process application, during the end user typing the words, a separate thread is running for grammar check. Another thread will save the current document automatically. In the excel, the situation is the same.

The third good example is explorer applications such as Microsoft Edge, Chrome, Firefox and etc. when we take a look for a single web page's source code, we can find a couple of HTML elements, CSS and Javascript. When we focus on the HTML, we can find a couple of things can be multithreaded. For example, generally, there are a couple of images in one web page, different images maybe locate in different servers, and the explorer can download the images from <img> tags at the same. Another example is about video playing, due to the internet access limitation, most of the explorer can play one video during downloading rather than play it until download completed.

5. [4 points] consider you have the following program:

```
1. boolean blocked [];
2. int turn;
3. void P(int id)
4. {
5.      while (true){
6.              blocked[id] = true;
7.              while (turn!=id){
8.                      while (blocked[1-id]) ;//do nothing
9.                      turn = id;
10.             }
11.             //critical section
12.             blocked[id] = false;
13.             //remainder section
14.     }
15. }
16.
17. void main(){
18. blocked[0] = false;
19. blocked[1] = false;
20. turn = 0;
beginProcess(P(0), P(1));
21.
22. }
```

Assume beginProcess(P(0), P(1)) does the following: suspend the execution of the main program; initiate concurrent execution of P(0), P(1); when P(0) and P(1) terminate, resume the main program.

This program represents a software solution to the mutual exclusion problem for two processes proposed in 1966 by Hymann, H. in a paper submitted to the Communications of the ACM. Find a counterexample that demonstrates that this solution is incorrect. Notice how it is easy to get fooled on this one!

After review the above source code, in basic idea for mutual exclusion program is to set the 'turn' as a locker. Only one process gets this locker, this process can enter the line 11: critical section.

The normal situation is:

P(0) start

Line 6, blocked[0] =true

Line 7, turn is now 0, turn==id

Line 11 enter critical section

P(1) start

Line 6, blocked[1] =true

Line 7, turn is now 0,

Line 8, check blocked[0] is true, begin to wait

P(0) start

Line 12, leave crtical section

P(1) start

Line 8, find blocked[0] is false,

Line 9, get the locker

Line11 enter cirtal section

Line12 leave critical section

But we can easily find a counterexample in order demonstrate this solution is incorrect.

P(1) start

Line 6, blocked[1] = true

Line 7, current turn is 0

Line 8, blocked[0] is false

Line 9 turn set to 1, at this same time, OS suspend P(1), so P(1) no time to write the value of 1 to turn in the memory.

P(0) start

Line 6, bocked[0] =true

Line 7, since the P(1) not write 1 to memory, at this time, P(0) get the value turn is also 0

Line 11, enter critical section, at this same time, OS suspend P(0)

P(1) start

Line 9, set the turn to 1, then line 7 find trun==id

Then line 11, enter critical section.

At this situation, it means in the next schedule, both of p(0) and P(1) are in the critical section.

6. [8 points] consider a sharable resource with the following characteristics:

1. As long as there are fewer than 3 processes using the resource, new processes can start using it right away.

2. Once there are 3 processes using the resource, all three must leave before any new process can begin using it.

Counters are needed to keep track of how many processes are waiting and active, and these counters are themselves shared resources that must be protected with mutual exclusion.

```
//share variables: semaphores, counters, and state information
Semaphore mutex = new Semaphore (1);
Semaphore block = new Semaphore (0);
int active = 0, waiting = 0;
boolean mustWait = false;
```

The following program offers a solution that appears to do everything right.

```
/* entry section */
mutex.acquire(); // enter the critical section
if (mustWait) { // if there are (or were) 3, then
    ++waiting; // we must wait, but also
    mutex.release(); // we must leave the critical section first
    block.acquire(); // wait for all current users to depart
```

```
}else{
        ++active; // update active count
}
mustWait = active == 3; // record if the count reached 3
mutex.release(); // leave mutual exclusion
/* critical section */
mutex.acquire(); //enter the critical section
--active; //update the active count
if (active == 0){ //check if this is the last one to leave
        int n;
        if (waiting <3) n = waiting;
        else n = 3; //see how many processes to unblock
        waiting -= n; //deduct this number from waiting count
        active = n; //set active to this number
        while (n>0){ //now unblock the processes
                block.release(); //one at a time
                --n;
        }
        mustWait= active ==3; //record if the count is 3
}
mutex.release(); //Leave the critical section
/* remainder section */
```

a. [4 points] Explain how this program works and why it is correct.

The first process comes:

Firstly, enter the entry section successfully because mustWait is false by default. After leave entry section (mutex.release()), the active value is 1, mustWait is false as well.

Then, enter the critical section. At this same, the second process comes.

The second process comes:

Firstly, enter the entry section successfully as process one. After leaving entry section(mutext.release()), the active value is 2, mustWait is false as well.

The third process is the same with number 2.

Considering the fourth process comes:

Firstly, number 4 start the mutext.acquires(), lock the entry section. Then mustWait is true. Waiting++ is 1, then leave the entry section and block itself.

If the 5th process comes, the process is the same with number 4.

Then, let us analysis the how to enter critical section.

The first process, start to enter the critical section. The first thing is lock the critical section by using mutex.qcquire(). Then - - active. The current active value is 3, so after - - active, leave the critical section.

The second process is similar with the first one.

When the third process is coming, the active is 0, then try to add 3 waiting task into active queue. If there are more than 3 waiting process, just selection 3. Otherwise put all waiting process into active queue. At the same time, unblock the waiting process.

From the analysis result, this program works and is correct.

b. [2 points] this solution does not completely prevent newly arriving processes from cutting in line but it does make it less likely. Give an example of cutting in line.

Considering there are 3 process, and then number 4th 5th 6th 7th comes.

Due to some reasons, before the 5th process try to block itself in the entry section (block.acquire()), it means this process run at mutex.release(), but not block itself, it is suspended by OS for a long time, so this process will not be rescheduled during 4th 6th 7th is resume, it's not frequent but likely.

As a result, the 7th process can block itself successfully, it means that 7th has a cutting in line.

c. [2 points] This program is an example of a general design pattern that is a uniform way to implement solutions to many concurrency problems using semaphores. It has been referred to as the I"ll Do It For You pattern. Describe the pattern.

Considering there are 5 processes. The first three can enter critical section. The 4th and 5th is in waiting list and is blocked. The advantage of 'I'll Do It For You' is that it's the first three processes to resume the blocked 4th 5th processes rather them the 4th and 5th themselves.

For example, when the 3rd process leave the critical section, the variable active will become 0. So, the 3rd process start to unblock the rest waiting processed.

Compare other design patterns, I'll Do It For You provide a better performance because the blocked processes do not need to check if they can enter the critical section. What the blocked processes need to do is waiting, if the resource become available, the leaving process will unblock the processes in the

waiting list. This is why this pattern is named I (the leaving process) will do it (unblock) for you (for those processes in the waiting list).

0.2 Programming Exercises

There are three java source code files:

SudokuWorker is a java class to implement the Runnable interface in order to support multi-threading.

SudokuCheck ia a java class which is designed to check if the input is a valid Sudoku puzzle using multi-threading solutions.

SudokuDemo is a demo class.

In order to test my program, I use three test cases:

sudoku1.txt

sudoku2.txt

sudoku3.txt

The first two files are downloaded from brightspace. After checking, the first is a valid Sudoku puzzle, but the second one is an invalid. In order to do more tests, I prepared a third file which also is not a valid puzzle.

The scree shot of running result of SudokuDemo is as below:

```
The sudoku1.txt is valid Sudoku puzzle!

The sudoku2.txt is invalid Sudoku puzzle! The detailed check result is:[1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0]

The sudoku3.txt is invalid Sudoku puzzle! The detailed check result is:[0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0]
```

Self-evaluation Please answer the following questions:

1. [1 points] Were you able to complete this assignment? What grade are you expecting? Please justify.

Yes, I did this assignment by myself on the time, and I feel my solution has reached the expectation from the teacher.

2. [2 points] Describe 2-3 challenges you faced while completing this assignment. How did you tackle those challenges?

Firstly, I have to learn how to work of the source code of the exercise 5. What's the meaning of turn and blocked array? Why there are two while loops? How can one process enter the critical section? After having a clear understanding, I can answer this question.

Secondly, there is not so much introduction about 'I'll Do It For You' design patterns, even I search this in google research engine. My solution is to lean the source of, and then try to give some my understanding about this pattern.

3. [2 points] provide a break down for the activities/milestones for this assignment. Give an estimate of hours spent on each activity. Try to be honest!

Date	Activities	Hours	Outcome
10/10	Revie the lecture handouts	3	Understanding Thread/Process and others
12/10	0.1 Practice Exercises 1-3	2	Solutions
13/10	0.1 Practice Exercises 4	2	Solutions
14/10	0.1 Practice Exercises 5	2	Solutions
15/10	0.1 Practice Exercises 6	2	Solutions
16/10	0.2 Programming Exercise: Java Threads support	2	Source code
17/10	0.2 Programming Exercise: main method	1	Source code

Lingda Cai A00372181