2024 Spring Semester

Operating System Mid-term Exam (30pts)

Without specific description, all problems consider a system having a single processor with a single core in Von Neumann architecture.

1. Please answer whether below statements are True or False. **If your answer is incorrect, the answer is counted as -1** so please think carefully. (1pt each; 6pts all)

a. If a single-core single-CPU system is given, then an OS cannot run more than two threads simultaneously but supports concurrent running of them.

-> True.

b. Kernel structures of operating systems are invariant to keep the compatibility.

-> False. For example, Mach used the microkernel structure which only adopts IPC, memory management, and CPU scheduling in the kernel, differing to other OSes.

c. If a thread in a multi-threaded process runs fork() system call, then OS creates only a copy of the thread without the other threads in the process.

-> False. Fork() system call duplicates the entire process of the thread called the fork(), therefore the other threads are also copied.

d. The busy waiting in a semaphore queue can effectively increase the efficiency of the entire system by running the process' code while waiting for the semaphore.

-> False. The busy waiting only runs the loop code inside the semaphore, which is irrelevant to the process' code, therefore no progresses occur while the waiting.

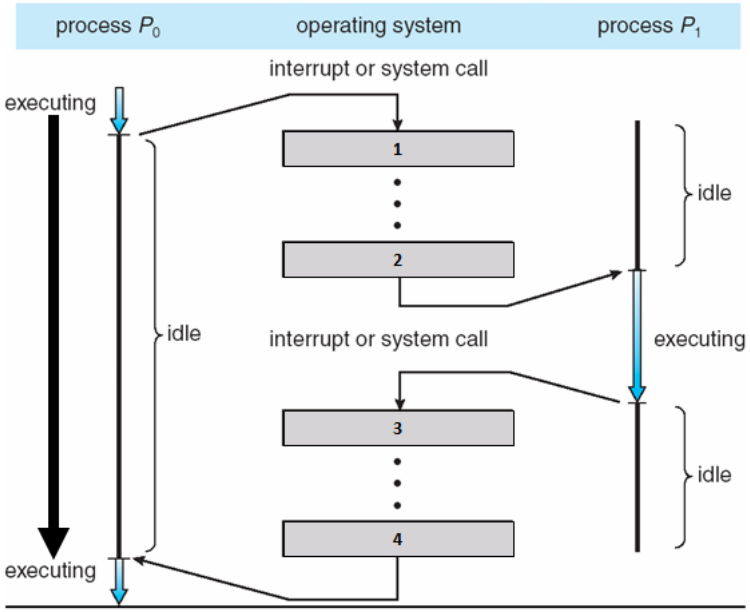
e. If the scheduler of an OS is preemptive, then the OS does not need to take care of the shared data accesses.

-> False. The preemption of the OS scheduler generates the race condition issue which requires an approach to control the access to the shared data.

f. One of the CPU scheduler's objective is balancing the CPU and I/O bursts for efficiently handling the processes.

-> True.

2. Below is the diagram describing the context switch between two processes. (8pts all)



a. Please fill in the boxes of 1-4. (2pts)

-> 1: save state into PCB0

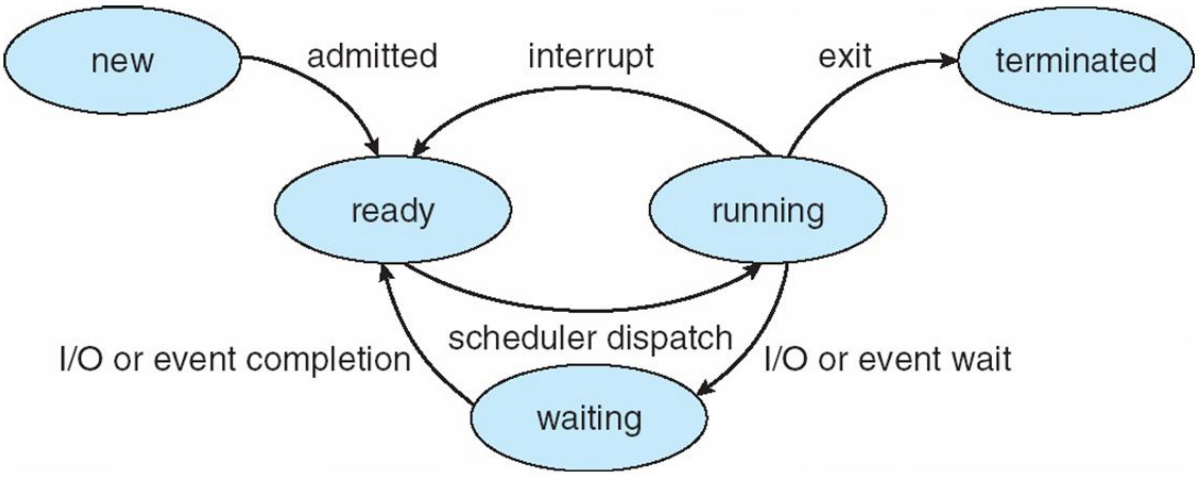
2: reload state from PCB1

3: save state into PCB1

4: reload state from PCB0

Writing only PCB0/1 is not counted as correct, even in partial scores.

b. Please draw the process state diagram. Do not forget to write down the transition events between the states (3pts)



For the correct state names -> 1pt

For the correct transition lines -> 1pt

For the correct transition labels -> 1pt

c. In the diagram of 2-a, let's assume that P_0 was interrupted and P_1 requested a system call. Please describe the state transitions of both processes during the context switch. (3pts)

-> P_0 started at the running state, and changed its state to the ready state by the interrupt. Then P_0 was rescheduled after P_1's system call, so again set as the running state by the scheduler dispatch.

Meanwhile, P_1 started at the ready state and changed to the running state by the scheduler dispatching. At the last execution of the diagram, P_1 was waiting for the system call responses from kernel, so its state became the waiting state.

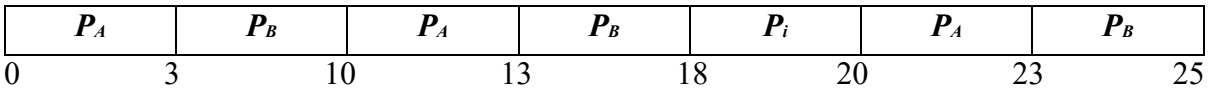
P_0: running -> ready -> running

P_1: ready –> running -> waiting

3. Assume that you implemented a hard real-time OS having **a Rate-monotonic scheduling system** for a robotic arm control computer. Currently, the system has two processes (**A, B**), where P_A reads the robot's sensor data and P_B sends the collected data to a server, and one idle thread (***P_i***). In this system, **shorter period means higher priority** and the idle thread is never scheduled until no processes exist in the ready queue. (8pts all)

ID	**Period**	**Process Ticks**
A	***10***	***3***
B	***20***	***12***
P_i	**-**	**-**

a. Draw a Gannt chart for the above scenario from 0 to 25 ticks. (1pt)



b. If P_A and P_B have to be executed in order, then answer whether the current system can run real-time or not considering the CPU resource. (2pt)

-> Due to the unclear explanation, writing one of the below answers is considered as correct. Of course, relevant explanation or description should be followed. A simple answer of 'yes' or 'no' is counted as wrong.

(1) P_A and P_B require 0.3 and 0.6 of CPU resources, so 0.9 of the system resource is required in running the system. Therefore, the system has sufficient CPU resources to run the processes.

(2) If we force the P_A->P_B execution order, and if we regard P_B marshals the prior P_A data and sends it within 12 ticks, then the current implementation in 3-a can send the data regularly in

every 20 ticks while the CPU utilization is 0.9, which is less than 1.

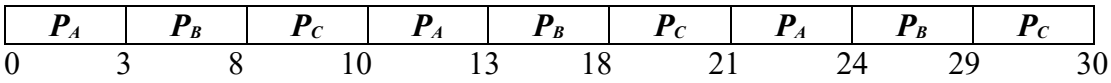
(3) Despite the current CPU utilization is 0.9, which is lower than 1, the current scheduling algorithm of the Rate-monotonic scheduling does not guarantee the deadline of those pairs, which requires 15 ticks for complete transmission but P_A preempts it.

(4) The current CPU utilization is 0.9, which is lower than 1, seems sufficient to run the processes concurrently. To make the two processes run in order, which requires 15 ticks for its completion, ruins the deadline of the third period of P_A. Therefore, this configuration cannot be run as hard real-time in this system. (if and only if 3-a is answered with the preemptive algorithm)

c. Now assume that you modified the processes as: P_A reads the robot's sensor data, P_B marshals the data, and P_C sends the collected data for 50 ticks. Below table shows the updated process information**.**

ID	Period	Process Ticks
A	***10***	***3***
B	***10***	***5***
C	***25***	***5***
P_i	-	-

Please draw a Gannt chart with the Earliest Deadline First Scheduling algorithm from 0 to 30 ticks. Here**, the earlier the deadline, the higher its priority** while priorities are $P_A > P_B > P_C$ when the deadlines are tied. (3pts)



4. Process Synchronization (8pts all)

a. Explain the three requirements to solve the critical section problem. (2pt)

-> Mutual Exclusion, Progress and Bounded Waiting.

b. Among the above, which requirement results in the starvation if it is not satisfied in the critical section problem? (1pts)

-> Bounded waiting.

c. Please explain two solutions to solve the Dining-Philosophers problem, except the solution limiting the number of philosophers. (2pts)

-> 1) Allowing a philosopher to pick up the chopsticks only if both are available.

2) Asymmetric solution: an odd-numbered philosopher picks up the left chopstick first and then the right chopstick, while an even-numbered philosopher does in reverse order.

d. Below example shows a scenario using **three binary semaphores S1, S2, and S3**, to run the code in the order of ReadDataStream -> AlienVoice -> PlayAudio -> ReadDataStream -> … periodically. Assume that T1, T2, T3 are threads in the same process and OS dispatches the threads fully random. Also, **please specify the initialization of each semaphore**. (3pts)

<table>
<tr><td colspan="3">// Shared Memory
float data[buffersize/2];
DeviceHandle hDev = DriverOpen();
int buffersize = 44100 * 0.01 * 2;
boolean exitCode = false;

semaphore S1, S2, S3;</td></tr>
<tr><td>T1() {
…
While (!exitCode) {

ReadDataStream(hDev, data, buffersize / 2);

}
}</td><td>T2() {
…
While (!exitCode) {

AlienVoice(data);

}
}</td><td>T3() {
…
While (!exitCode) {

PlayAudio(data);

}
}</td></tr>
</table>

-> At this time, the score was strictly graded as: running – 3, not running – 0.

<table>
<tr><td colspan="3">// Shared Memory
float data[buffersize/2];
DeviceHandle hDev = DriverOpen();
int buffersize = 44100 * 0.01 * 2;
boolean exitCode = false;

semaphore S1, S2; // Initialized as 0
semaphore S3; // Initialized as 1</td></tr>
<tr><td>T1() {
…
While (!exitCode) {

S3.wait();
ReadDataStream(hDev, data, buffersize / 2);
S1.signal();
}
}</td><td>T2() {
…
While (!exitCode) {
S1.wait();
AlienVoice(data);
S2.signal();
}
}</td><td>T3() {
…
While (!exitCode) {
S2.wait();
PlayAudio(data);
S3.signal();
}
}</td></tr>
</table>