Name: Zaitinov Edige

.

---

1) [10] Return of Alice! Alice recalls from lecture that processes and threads are two different ways to achieve concurrency in a system. Since threads can share data, Alice has made the connection that threads can "communicate" through this data section. On the other hand, Alice remembers that processes do not share resources well, which makes her wonder: *How can processes communicate with one another?* As her trusted (and smart) friend,, she comes to you to ask. Name two methods by which processes communicate with each other, and state an advantage of each method, and **briefly** explain an implementation challenge.

a) Method 1: Shared Memory

Advantage:
Really low overhead communication

Implementation Challenge:
Synchronization problems, memory protection

b) Method 2: Message Passing

Advantage:
no conflicts when communicating, so no need to protect memory

Implementation Challenge:
mailbox sharing issue, message buffering capacity

2) [8] Later that night, Alice decides to work more on understanding OS concepts. However, because it was a long day for Alice, she falls asleep at her desk, where she starts dreaming about multithreading models between user threads and kernel threads. In her dream, Alice goes to her computer and sees that she has started a study guide with a few statements on multithreading models. However, she doesn't know whether (dream) Bob has modified any of the statements. Mark an **X** in the appropriate box to indicate whether the statement is True or False.

	True	False
In the Many-to-One multithreading model, many kernel threads are mapped to a single user thread.		+
The One-to-One multithreading model suffers from scalability issues.	+	
The Many-to-Many multithreading model always uses equal numbers of kernel and user threads.		+
In the Many-to-One multithreading model, a system call from a user thread will block an entire task.	+	

3) [15] Upon waking up to the sound of chirping birds (and her alarm right before class would start), Alice flips the page of her notebook, and then enters the zoom link for the class, where scheduling algorithms are being discussed. However, Alice hasn't eaten anything yet, and can't take her mind off of food. Coincidentally, the lecture is discussing scheduling algorithms and starvation. Alice realizes that she is too distracted at the moment, and decides to make a table listing some scheduling algorithms, and whether they could result in starvation. Assuming that a process burst time can be infinite, mark an **X** in the appropriate column for whether the scheduling algorithm can result in starvation, and briefly explain.

	Yes	No	Explanation
First-Come First-Served (FCFS)	+		If one of the processes runs infinitely then the processes comes later than that will starve
Shortest Job First (SJF) (Non-preemptive)	+		If shorter jobs keep coming
Shortest Remaining Time First (SRTF) (Preemptive)	+		Same as above
Round Robin (RR)		+	Similar to timesharing, each process will get a time quantum
Priority (Preemptive)	+		Low priority processes may never execute

4) [32] Later, Alice needs to analyze some GANTT charts for some of the scheduling algorithms from class. She complains that GANTT charts are too tedious to draw on a computer, so she decides to print the homework and manually draw GANTT charts using a pen. However, after Alice finishes, Bob (being the prankster he is) decides to take the page and make a paper airplane, which (inconveniently for Alice) flies out the window into a puddle of water. Not wanting to waste more paper, Alice decides to redraw the GANTT charts on the last page.

On the bright side, Alice already organized the following information about the processes:

Process ID	Arrival Time	Burst Time	Priority
P1	0	70	3
P2	30	20	1
P3	10	50	2
P4	50	20	4

Additionally, she is given the following guidelines:

- When priority is being used, a smaller priority number means higher execution priority .
- For tie-breaking cases, the process with the earlier arrival time should execute.
- There is no context switching overhead

Help Alice by drawing GANTT charts for each of the following scheduling algorithms, and attach it to the end of this file as the last page using pdf editors or other online tools. Then, complete the following table with the average waiting time and turnaround time of each algorithm. [Note: the GANTT charts will be used to determine partial credit if the values in the table are incorrect]

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)	71.25	111.25
Priority (Preemptive)	45	85

5) [12] Learning from the last time she didn't eat breakfast, Alice decides to eat some bread before the next lecture. Also coincidentally, Alice (while munching on her bread) learns about the Bakery Algorithm, which is a solution to the critical section problem. Inspired by the Bakery Algorithm, Alice tries to design an algorithm of her own to provide a 2-process solution to the critical section problem. After she finishes, Bob comes along and sneakily erases the boolean values of the flags on line 1 and 7 of her algorithm, hoping that she wouldn't notice.

Shared variables: flag[0], flag[1]
Initially flag[0] = flag[1] = false;

Process P0:

```
0: while (true) {
1:        flag[0] = ____;
2:        while (flag[1]) {
3:            flag[0] = false;
4:            while (flag[1]) {
5:                no-op;
6:            }
7:            flag[0] = ____;
8:        }
9:        critical section
10:       flag[0] = false;
11:       remainder section
12: }
```

Process P1:

```
0: while (true) {
1:        flag[1] = ____;
2:        while (flag[0]) {
3:            flag[1] = false;
4:            while (flag[0]) {
5:                no-op;
6:            }
7:            flag[1] = ____;
8:        }
9:        critical section
10:       flag[1] = false;
11:       remainder section
12: }
```

a) A few hours later Alice notices the blanked out areas in her algorithm, but has lost the inspiration she had earlier in the day. Therefore, she comes to you (again) asking whether you can mark an **X** in the appropriate box for the value of the boolean flags on line 1 and 7 so that the algorithm satisfies mutual exclusion.

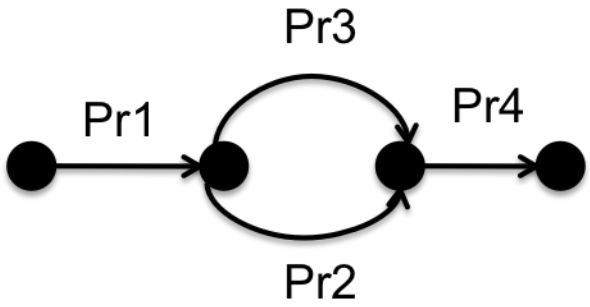
	true	false
Line 1	+	
Line 7	+	

b) You stare at the code and ask Alice about the other two requirements to the critical section problem: Progress and Bounded Waiting. Mark an **X** in the appropriate box for whether the requirement is satisfied or not.

	Satisfied?	Not satisfied?
Progress		+
Bounded waiting		+

6) Alice, wanting to get a bit more clarification on semaphores, decides to refer to a textbook. She flips to a page, and reads the following:

In an operating system, processes can run concurrently. Sometimes we need to impose a specific order in execution of a set of processes. We represent the execution order for a set of processes using a process execution diagram. Consider the following process execution diagram. The diagram indicates that **Pr1** must terminate before **Pr2**, **Pr3** and **Pr4** start execution. It also indicates that **Pr4** should start after **Pr2** and **Pr3** terminate and **Pr2** and **Pr3** can run concurrently.



The process execution diagram can also be represented using **Serial** and **Parallel** notation. The above execution diagram is represented as **Serial(P1, Parallel(P2, P3) , P4)**.

We can use semaphores in order to enforce the execution order. Semaphores have two operations as explained below.

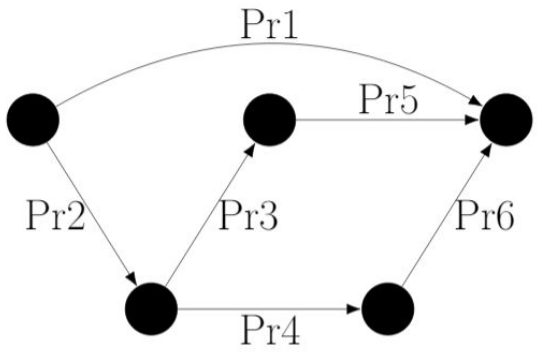
- **P** (or wait) is used to acquire a resource. It waits for semaphore to become positive, then decrements it by 1.
- **V** (or signal) is used to release a resource. It increments the semaphore by 1, waking up a blocked process, if any.

We can use the following semaphores to enforce the execution order above:

s1=0; s2=0; s3=0;
Pr1: body; V(s1); V(s1);
Pr2: P(s1); body; V(s2);
Pr3: P(s1); body; V(s3);
Pr4: P(s2); P(s3); body;

This assumes that the semaphores **s1**, **s2**, and **s3** are created with an initial value of **0** before processes **Pr1**, **Pr2**, **Pr3**, and **Pr4** execute.

Although Alice read the previous page several times, she still does not quite understand what the textbook is explaining. Frustrated, she takes a short break and decides to do some stargazing through her window. After some time, Alice spots the Cassiopeia constellation and excitedly draws out the placement of the stars on her homework paper. However, she forgets where the edges should go, so she decides to simply make some up, as shown below.



a) [5] Alice then realizes that it's long past midnight and hasn't gotten any additional work done. After getting back to her desk, Alice decides to use her drawing as a process execution diagram. She wishes to ask you a few questions about process execution diagrams over text. Write the representation of the process execution diagram using **Serial** and **Parallel** notation below.

Parallel (Pr1, Serial(Pr2, Parallel(Serial(Pr3, Pr5), Serial(Pr4, Pr6))

b) [18] Although your phone receives Alice's message, you are already asleep in bed (and thus ignore Alice's message). In the morning, you are greeted with a process execution diagram encoded as above. Having had a very nice sleep, you decide to help Alice with her question: How should the semaphores **s1,s2,s3,s4** be used to enforce the execution for the given process execution diagram? Assume that the semaphores **s1,s2,s3,s4,** are all initialized to 0 (as below).

s1=0; s2=0; s3=0; s4=0;

Pr1: body;

Pr2: body; V(s2); V(s2);

Pr3: P(s2); body; V(s3);

Pr4: P(s2); body; V(s4);

Pr5: P(s3); body;

Pr6: P(s3); body;

[Upload your GANTT Charts here]