Process Management
Practice Exercises
Winter 2025

Question 1: Fork-Exec (A)

How many times does the following program print “C”?

First, answer without running the code. Then, you can run the code to check your answer.

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

void fork_test() {
  if (fork() != 0) {
    if (fork () != 0) {
      fork();
    } else {
      printf("A\n");
    }
  } else {
    if (fork() != 0) {
      printf("B\n");
    } else {
      fork();
    }
  }

  printf("C\n");
}

int main() {
  fork_test();
  return 0;
}
```

Question 2: Fork-Exec (B)

When a running process is duplicated by fork(), which of the following are shared between the parent and the child process? “Shared” here means that one process changing the data it sees results in changes that the other process can *also* see. (don’t overthink this!)

a. Stack
b. Heap
c. Code
d. Static data
e. Registers
f. I/O Buffers
g. Content on the hard disk

Question 3: Multi-process Communication

Conceptually, a browser is an infinite loop of receiving and handling events (and, usually, updating the screen, which is an I/O operation). When running on a machine with a single CPU core, does it make sense to use multiple processes (or threads) to implement the browser? If yes, why? If not, why not?

Question 4: Message Passing

Suppose we want to use message passing to allow a client and server to communicate. The server has a specialized, very fast implementation of a string “find” operation that the client would like to use. (Note that this is a case of RPC, but we are going to focus only on the messaging aspect.) The implementation of this operation has the following signature:

```
int position(char c, char *s)
```

and can only be called **in code on the server side**. Write pseudocode for the server and client implementing a communication protocol allowing the client to use the function. This should involve the client and server communicating a message in the form of a struct. Show the definition of the struct you are using. You can assume that both the client and the server have a pointer named `msg` to an already heap-allocated space for a message. (If your struct requires specialized

allocation, you can assume that the client and server both allocated space with the appropriate helper function. There are good solutions that use this, and good solutions that don't!)

Hint: Remember that all data in the message must be passed *by value*. Additionally, since you can only define one struct, the message struct needs space both for the client to send the request, *and* for the server to send its response.

Question 5: Scheduling (A)

Two processes, A and B, have the following sequential execution patterns:

A: CPU (4ms), I/O (2ms), CPU (4ms), I/O (2ms), CPU (4ms)
B: CPU (1ms), I/O (2ms), CPU (1ms), I/O (2ms), CPU (1ms)

The I/O operations for the two processes do not interfere with each other and are blocking (meaning the process cannot continue until the operation completes). Both processes are ready to run at time 0.

1. If the processes are run consecutively, one after the other, what is the elapsed time before they are **both** complete?
2. Sketch the execution pattern under a non-preemptive, FCFS scheduler and determine the elapsed time for completion. (Recall an FCFS scheduler simply schedules tasks in the order that they became ready.) In the event of a tie, you should give priority to task A. Furthermore, you can assume that the scheduler and context switching are both instantaneous.
3. Repeat (2), but for a preemptive scheduler that operates on time slices of 2ms. (That is, after 2ms, the scheduler must perform a context switch unless no other process is runnable.)
4. From your results of (2) and (3), what are the advantages and disadvantages of preemption for this scenario?

Question 6: Scheduling (B)

The following processes arrive for execution at the times indicated and have the indicated runtimes. Assume the scheduler is **not preemptive**.

Process	**Arrival time**	**Run time**

A	0.0	8.0
B	0.4	4.0
C	1.0	1.0

1. What is the average turnaround time for these processes under FCFS scheduling? (Note that the scheduler must make its first decision at time 0.)
2. What is the average turnaround time under SJF scheduling?
3. Compute the average turnaround time if the CPU is idle for the first 1 time unit, and then SJF scheduling is used.

For additional practice, consider the average response times as well.

Question 7: Semaphores

Suppose a semaphore is being used as a lock (initialized to 1, as seen in class) and that a bug in the implementation of wait() results in wait() not performing its operations atomically. The post() function is implemented correctly. In a short paragraph, explain how this bug can result in the violation of mutual exclusion.

Question 8: Dining Philosophers, Revisited

Recall the dining philosophers problem seen in class: 5 philosophers come to eat at a round table. In front of each philosopher there is a plate of noodles. Between each philosopher there is one chopstick. To be able to eat, philosophers need to hold *both* chopsticks on either side of their plate. The rules of the diner are as follows:

- There is an infinite food supply.
- Philosophers can only be in one of two states: eating or thinking.
- **Philosophers may not communicate with each other.**
- Philosophers only atomic operations are picking up a single chopstick, or releasing a single chopstick.
- Once a philosopher starts eating, they must release both of their chopsticks after a finite amount of time.

In lecture, we discussed conceptually several approaches to the dining philosophers problem. Pick one. Formalize this algorithm by implementing it in code. On an exam, you may use pseudocode, but for practice here you should write code that compiles and runs (recall that you must pass the `-pthread` option to gcc/clang when using pthreads).

1. Using the synchronization primitives seen in class for this problem (locks and semaphores) implement your chosen solution. Each of the five chopsticks should be represented by an individual lock (or equivalently by a semaphore initialized to 1) in addition to any other primitives that you use to synchronize the philosophers.
2. Explain why your chosen solution avoids deadlocks.