Operating Systems

Fall 2020

First and Last Name JaVonne Harrison

Exam #1

Read all instructions and questions before starting the quiz. All work must be completed independently.

Submission Instructions

There are two portions to this exam: a written portion, and a coding portion. The written portion is this document completed by you. When completed, you will upload this document to Blackboard as your submission for the written portion. Included in this document is a form for you to enter the GitLab repository for the coding portion. **No late or resubmissions will be accepted**, once this document has been submitted on Blackboard the attempt is complete.

Completion Instructions

Begin by entering your name in the form at the top of this page. If you do not see the form, download a new copy of this document from Blackboard. Enter your answers in the forms provided, **no information outside of the provided forms will receive credit**. There are multiple selection questions. Each multiple selection question may have more than one correct answer, check **all** correct answers. (Answers are checked by clicking upon them).

GitLab Repository

Enter the git URL (not https) of the GitLab repository used for the coding portion in the form below. The repository should be named *cosc439-lastname-exam-1*, where *lastname* is your last name. Add ctessler@towson.edu as a developer to the GitLab project.

git@gitlab.com:jharri86/cosc439-harrison-exam-1.git

Academic Integrity

By submitting this document on Blackboard, the student certifies the work is entirely their own and subject to Towson University's Academic Integrity Policy (link).

Points Available: 70

Chapter 1

Question 1 (6 points)

What is a bootstrap program, and where is it stored?

A bootstrap program is a small piece of code located in the ROM or EEPROM (firmware). Its job is to initialize all parts of the system, run diagnostics to determine the state of the machine, and to locate the operating system, load it into the kernel, and start its execution.

Question 2 (2 points)

What is the purpose of GRUB?

GRUB is a common bootstrap loader that allows selection of the kernel from multiple disks.

Question 3 (1 points)

How is GRUB related to the bootstrap program?

GRUB is an opensource bootstrap program.

Question 4 (1 point)

A(n) _____ is the unit of work in a system.

☒ **process**
☐ **operating system**
☐ **timer**
☐ **mode bit**

Chapter 2

Question 1 (6 points)

Describe three general methods used to pass parameters to the operating system during system calls

One way is to pass individual parameters to the registers. However, when there are more parameters than registers, parameters can be stored in a block or table, and the address of the block or table can be passed as a parameter. The third way is to push parameters into the stack, where the operating system can pop the parameters off of the stack.

Question 2 (2 Points)

_____ provide(s) an interface to the services provided by an operating system.

☐ Shared memory
☒ System calls
☐ Simulators
☐ Communication

Question 3 (2 Points)

_____ allow operating system services to be loaded dynamically.

☐ Virtual machines
☒ Modules
☐ File systems
☐ Graphical user interfaces

Chapter 3

Question 1 (2 Points)

A process control block ____.

- ☒ **includes information on the process's state**
- ☐ **stores the address of the next instruction to be processed by a different process**
- ☐ **determines which process is to be executed next**
- ☐ **is an example of a process queue**

Question 2 (4 Points)

Explain the concept of a context switch.

A process A will be executing, and an interrupt or system call will trigger the kernel to save the context of the old process in process A's PCB and loads the saved context of a different PCB, that of process B, which is scheduled to run. Process B will run until an interrupt or system call, where the same process could repeat, but with process B and another process C, respectively.

Question 3 (2 Points)

Explain the difference between an I/O-bound process and a CPU-bound process.

An I/O-bound process is one that uses more time to do I/O operations than computations, while CPU-bound processes spend more time doing computations, and do I/O operations less frequently.

Question 3 (2 Points)

Which type of tasks benefit more from multi-core (rather than single core) systems: CPU or I/O bound tasks, and why.
CPU-bound tasks benefit more from multi-core systems, because if there are computations that need to be done that take a long time, CPU-bound processes can be run concurrently on multiple cores within the CPU, and the system will not be stuck on one computation.

Chapter 4

Question 1 (2 Points)

Of the following, which do *not* use an existing thread — rather than creating a new one — to complete a task.

☐ **lightweight process**
☒ **thread pool**
☐ **scheduler activation**
☐ **asynchronous procedure call**

Question 2 (4 Points)

Distinguish between data and task parallelism.

Data parallelism is when subsets of data are shared across the different cores and performing different parts of the same operation. Task parallelism is the concept that threads are distributed among the cores, and that each thread is performing a unique operation.

Question 3 (2 Points)

Consider the following code segment for an initial process beginning at line 1:

```
1  for (int i = 0; i < 2; i++) {
2        pid_t pid = fork();
3        work(i);
4  }
```

Assuming no run-time errors, how many calls to `work()` are made in the immediate children of the initial process? An immediate child of process *p* is `fork()`ed from *p*, i.e. it is not forked from a child of *p*.

3

Question 4 (2 Points)

In Question 3's code segment, what are the values of `i` for each invocation of work in the child and grand-children of the initial process?

The first child of p invokes work(0) and work(1) (i = 0 and i = 1). The second child of p invoked work(1) (i = 1).

Chapter 5

The following definitions of `wait()` and `signal()` appear on pages 215 and 216 of the book.

```
1   wait (semaphore *S) {
2         S->value--;
3         if (S->value < 0) {
4               add this process to S->list;
5               block();
6         }
7   }
8   signal (semaphore *S) {
9         S->value++;
10        if (S->value <= 0) {
11              remove a process P from S->list;
12              wakeup(P);
13        }
14  }
```

Question 1 (4 Points)

Given book's implementation of `wait()` and `signal()`, two threads attempting to use a semaphore S, may still encounter a race condition. Provide the line number(s) of the functions that create the race condition(s).

2 and 9

Question 2 (2 Points)

Provide an example of inter-leaved instructions that illustrate the race condition. An example of the expected format is present in the form below for threads i and j

S = 0

i : line 2 S->value = -1
j : line 9 S->value = 1
i : line 2 S->value = 0
j : line 9 S->value = 1

Question 3 (2 Points)

How can the race condition be resolved *without* the use of a mutex?

You can resolve this race condition by making sure that both the wait() and signal() functions run atomically.

Question 4 (2 Points)

Can the following code be modified to use a mutex lock rather than a semaphore? Why or why not? (`sem_post()` is the POSIX form of `signal()`)

```
1   sem_t sem;
```

```
2   sem_init(&sem, 0, 1);
3   sem_wait(&sem);
4         // critical section
5   sem_post(&sem);
6   sem_destroy(&sem);
```

Since a binary semaphore is being implemented, I assume that only one variable is being protected, so yes, a mutex lock can be used in this situation because binary semaphores behave very similarly to mutex locks.

Deadlock

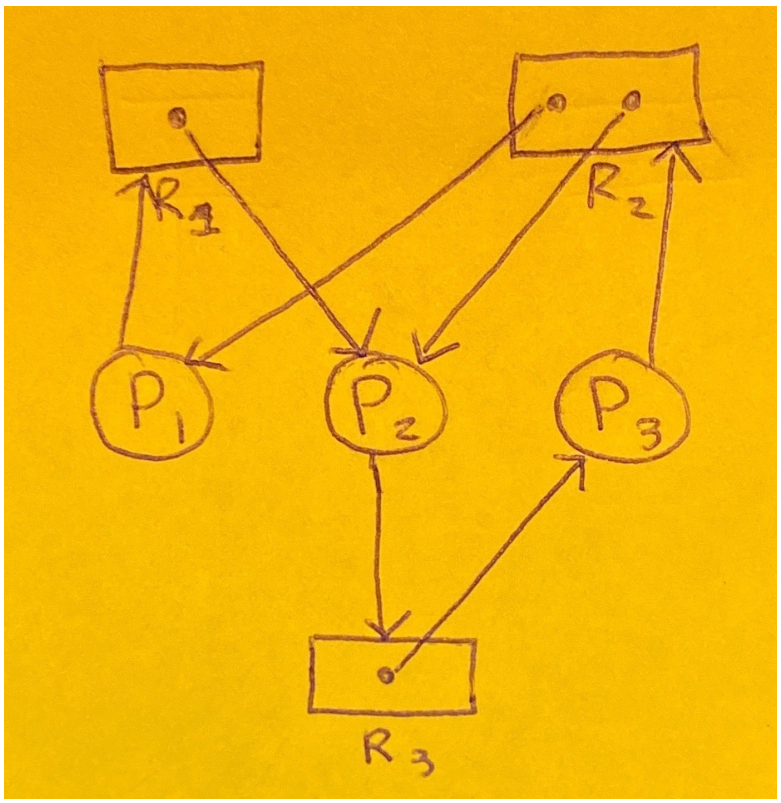
A system has three processes P_1, P_2, and P_3, and three resources, R_1, R_2 and R_3, respectively. R_1 and R_3 have one instance and R_2 has two instances.

Consider the following scenario:

- **P1 is allocated an instance of R2, and requests an instance of R1**
- **P2 is allocated an instance of R1 and R2, and requests an instance of R3**
- **P3 is allocated an instance of R3, and requests an instance of R2**

Question 1 (4 Points)

Draw the resource-allocation graph.



Question 2 (2 Points)

Is there a cycle in the graph? If yes name it.

Yes, there are two cycles. The first cycle is (P_1, R_1), (R_1, P_2), (P_2, R_3), (R_3, P_3), (P_3, R_2), (R_2, P_1). The second cycle is (P_2, R_3), (R_3, P_3), (P_3, R_2), (R_2, P_2).

Question 3 (4 Points)

Is the system in deadlock? If yes, explain why. If not, give a possible sequence of executions after which every process completes.

The system is in deadlock because all of the resources are unavailable, and the processes are unable to proceed without the proper resources, so none of the processes can proceed.

Stack Calling Convention (10 Points)

In Lab 3, the little operating system utilized the stack to invoke the `sum_of_three()` function written in C, with the return value placed in the `eax` register. Begin by recreating (copying) Lab 3 in a subdirectory of your GitLab repository named calling, after adding the source, the full contents of the repository should be as follows

```
cosc439-lastname-exam-1.git/
+-- calling/
    +-- bochsrc.txt
    +-- iso/
    |   +-- boot/
    |       +-- grub/
    |           +-- grub.cfg
    +-- kmain.c
    +-- loader.s
    +-- link.ld
    +-- makefile
```

Part 1 (4 Points)

Create a function named `product()` (in C) that computes the product of two integers and places the result in the `eax` register. Invoke the product function in assembly where the first argument is the index in the alphabet of the first letter of your name, and the second argument is index of the first letter of your last name (e.g. Robert Smith, R = 18, S = 19).

Part 2 (4 Points)

Invoke `product()` in the assembly using the result of the previous call to product as both the first and second argument. For Robert Smith product is invoked with 18 and 19, resulting in 342. Product is then invoked with 342 and 342 resulting in 116964.

Part 3 (2 Points)

Provide the expected values of the call to `product()` and the return values of both calls in the form below

Click or tap here to enter text.

Provide a screenshot of the final value of `eax` from the log in the form below.

Push the completed project to your GitLab repository, do not include any object or executable files, only the sources listed above. make run must build and start the littleos on the bochs emulator.

Extra Credit (4 points)

The following question and answer, it is incorrect. How is it incorrect and why? (be explicit)

Question: *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.*

Click or tap here to enter text.