Sample Exam 1

1-Oct-19

3 October, 2019

_____ , _____
Last Name , First Name

Instructions:

- ***Show all work on the front of the test papers. No work shown may mean 0 points given!*** *If you need more room, make a clearly indicated note on the front of the page, "MORE ON BACK", and use the back. The back of the page will* ***not*** *be graded without an indication on the front.*
- ***Read*** *each question* ***carefully*** *and* ***follow the instructions****. Unless otherwise stated,* ***you must show your work and clearly illustrate your steps****.*
- *If you round a numerical answer, you must give at least* ***3 significant digits****.*
- *Put your name at the top of* ***each*** *test page and be sure your exam consists of the number of pages designated in the headers.*
- *The space provided does* ***NOT*** *necessarily represent the amount of writing necessary.*
- *You may* ***not*** *use any notes, homework, labs, or other books. Only* ***memoryless*** *calculator is allowed.*
- ***You may not use any notes, homework, labs, or other books.***

COMMENTS, FEEDBACK, or any special instructions for the professor:

Problem	Available	Points
1		
2		
3		
4		
5		
6		
7		
Total	100	

Important: *In completing this exam, I used a calculator with no communications capability, and no information of relevance to the course was stored in the calculator. I did not use any other electronic device or any other references. My work was solely my own.*

Your Calculator's Maker and Model#: _____

Signature: _____

You must sign this to receive credit for the exam.

Sample Exam1

3 October, 2019

1-Oct-19

_____ , _____
Last Name , First Name

Remember to show ALL work here and in EVERY problem on this exam.

Q1.

(a) How would you design a system to run an entire operating systems as an application on top of another operating systems?

ANS: The sue of virtual machine would help to run an entire operating systems as an application on top of another operating system. The host operating system creates an abstract virtual machine that is identical to the hardware the host operating system is running on. The guest operating system then runs in the virtual machine as if it were running on the real hardware.

(b) Describe the difference between symmetric and asymmetric multiprocessing. What are advantages of multiprocessor systems?

ANS: Symmetric multiprocessing treats all processors as equals, and I/O can be processed on any CPU. Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The master distributes tasks among the slaves, and I/O is usually done by the master only. Multiprocessors can save money by not duplicating power supplies, housings, and peripherals. They can execute programs more quickly and can have increased reliability. They are also more complex in both hardware and software than uniprocessor systems.

Sample Exam1

3 October, 2019

_____ , _____
Last Name , First Name

Q2.

(a) How should operating system support communication between applications? Explain your reasoning.

ANS: There an operating systems communicate between applications in three ways: producer-consumer-model, client-server-model, and file-system. In producer-consumer-model, the communication is one way where the producer writes and the consumer reads. The client-server-model allows two way communication between processes. The file system allows for communication that can be separated in time.

(b) List the four steps that are necessary to run a program on a completely dedicated machine.

ANS:

i. Reserve machine time.
ii. Manually load program into memory.
iii. Load starting address and begin execution.
iv. Monitor and control execution of program from console.

(c)

(i) What are the three main purposes of an operating system?

ANS:

- To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible.
- To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.
- As a control program it serves two major functions: (1) supervision of the execution of user programs to prevent errors and improper use of the computer, and (2) management of the operation and control of I/O devices.

Sample Exam1

1-Oct-19

3 October, 2019

_____ , _____
Last Name , First Name

Q3.

(a) What is the purpose of system calls? What are the three major activities of an operating system in regard to memory management?

ANS: System calls allow user-level processes to request services of the operating system. i. Keep track of which parts of memory are currently being used and by whom. ii. Decide which processes are to be loaded into memory when memory space becomes available. iii. Allocate and deallocate memory space as needed.

(b) Explain the fundamental differences between the UNIX **fork()** and Windows **CreateProcess()** functions.

ANS: Each function is used to create a child process. However, fork() has no parameters; CreateProcess() has ten. Furthermore, whereas the child process created with fork() inherits a copy of the address space of its parent, the CreateProcess() function requires specifying the address space of the child process.

Sample Exam1

3 October, 2019

1-Oct-19

_____ , _____
Last Name , First Name

Q4.

(i) Explain, what will happen if you run the following program on UNIX?

```
main() {
        while (fork() > 0)
    ;
}
```

ANS: Create an infinite number of child process

ii) Why are user-level threads packages generally cooperatively scheduled?

ANS: User-level thread packages are co-operatively scheduled because generally they form part of a single kernel-level thread. Most often this means that the process runs on its own user-level scheduler, separate from the kernel scheduler. This means it does not have access to the strict timing-based preemptive scheduling that kernel level threads enjoy, so must use cooperative scheduling.

Sample Exam1

3 October, 2019

1-Oct-19

_____ , _____
Last Name , First Name

Q5.

(a) How many processes are created if the following program is run?

```
main (int argc, char ** argv) {
        forkthem(5)
}
void forkthem (int n) {
        if (n > 0) {
                fork();
                forkthem(n-1);
        }
}
```

ANS: Six, one for main, and five children from forkthem(5).

Final answer: _____

(b) Show the output of the following program.

```
main () {
        int val = 5;
        int pid;

        if (pid = fork())
                wait(pid);
        else
                exit(val);
        val++;
        printf("%d\n", val);
        return val;
}
```

ANS: Program prints 6 twice.

Final answer: _____

Sample Exam1

3 October, 2019

1-Oct-19

_____ , _____
Last Name , First Name

Q6.

```
#define NTHREADS 10
thread_t threads[NTHREADS];
main() {
    for (i = 0; i < NTHREADS; i++)  thread_create(&threads[i], &go, i);
    for (i = 0; i < NTHREADS; i++) {
        exitValue = thread_join(threads[i]);
        printf("Thread %d returned with %ld\n", i, exitValue);
    }
    printf("Main thread done.\n");
    return 0;
}
void go (int n) {
    printf("Hello from thread %d\n", n);
    thread_exit(100 + n);
    // REACHED?
   }
```

For the above code fragments, answer the followings:

(a) What is the minimum and maximum number of items that the main thread enters the READY state on a uniprocessor?

ANS: The minimum number of times is TWO. The maximum number of times is (near) infinity.

(b) **(5 pts)** What is the minimum and maximum number of items that the main thread enters the WAITING state?

ANS: Minimum 0 and maximum 10 items

(c) **(5 pts)** Suppose that we delete the second for loop so that the main routine simply creates NTHREADS threads and then prints "Main thread done." What are the possible outputs of the program now?

ANS: "Main thread done." Could display at any point when the other threads are printing. Since, we are no longer waiting for the thread to exit.

Sample Exam1

3 October, 2019

1-Oct-19

_____ , _____
Last Name , First Name

Q7.

(a) Suppose that you mistakenly create an automatic (local) variable *v* in one thread *t1* and pass a pointer to *v* to another thread *t2*. Is it possible that a write by *t1* to some variable other than *v* will change the state of *v* as observed by *t2*? If so, explain how this can happen and give an example. If not, explain why not.

Solution: Yes. This can happen if t1 returns from the procedure that allocated v and then calls another procedure that allocates something else in the same place on the stack.

```
foo () {
        bar ();
        baz ();
    }
    bar () {
        Object v;
        sthread_t t2;
        sthread_init(\&t2, function, \&v);
    }
    baz() {
        int buffer[1000];
        int i;
        for (i=0; i<1000; i++) {
            buffer[i] = 42;
        }
    }
```

(b)

For the following implementations of the "H20" problem, say whether it either (i) works, (ii) doesn't work, or (iii) is dangerous -- that is, sometimes works and sometimes doesn't. If the implementation does not work or is dangerous, explain why and show how to fix it so it does work.

Here is the original problem description: You have just been hired by Mother Nature to help her out with the chemical reaction to form water, which she does not seem to be able to get right due to synchronization problems. The trick is to get two H atoms and one O atom all together at the same time. The atoms are threads. Each H atom invokes a procedure *hReady* when it is ready to react, and each O atom invokes a procedure *oReady* when it is ready. For this problem, you are to write the code for *hReady* and *oReady*. The procedures must delay until there are at least two H atoms and one O atom present, and then one of the threads must call the procedure *makeWater* (which just prints out a debug message that water was made). After the *makeWater* call, two instances of *hReady* and one instance of *oReady* should return. Your solution should avoid starvation and busy-waiting.

You may assume that the semaphore implementation enforces FIFO order for wakeups—the thread waiting longest in P() is always the next thread woken up by a call to V().

Sample Exam1

3 October, 2019

1-Oct-19

_____ , _____
Last Name , First Name

(i) Here is a proposed solution to the "H20" problem:

```
Semaphore h_wait = 0;
Semaphore o_wait = 0;
int count = 0;

hReady()
{
    count++;
    if(count %2 == 1) {
        P(h_wait);
    } else {
        V(o_wait);
        P(h_wait);
    }

    return;
}
```

```
oReady()
{
   P(o_wait);
   makeWater();
   V(h_wait);
   V(h_wait);

   return;
}
```

This solution is dangerous. Threads calling **hReady()** *access shared data without holding a lock! For example, you could have N threads increment* **count**, *and because they do so without a lock, the result could be 1 instead of N -- in other words, no water would be made regardless of how many* **H**'s *arrived.*

The solution is to put a lock acquire before the first line in **hReady**, *and release before the first* **P(h_wait)** *and after the second* **P(h_wait)**. *Some put the lock acquire after the increment, and that simply doesn't work!*

(ii) Another proposed solution to the "H20" problem:

```
Semaphore h_wait = 0;
Semaphore o_wait = 0;

hReady()
{
    V(o_wait)
    P(h_wait)

    return;
}
```

```
oReady()
{
   P(o_wait);
   P(o_wait);
   makeWater();
   V(h_wait);
   V(h_wait);

   return;
}
```

This is dangerous, since it may lead to starvation. If two **H**'s *arrive, then the value of the* **o_wait** *semaphore will be 2. If two* **O**'s *arrive, then they can each decrement* **o_wait**, *before either can decrement it twice. So no water is made, even though enough atoms have arrived.*

The fix is to put a lock acquire before the first line in **oReady**, *and a lock release after the two* **V(h_wait)**'s. *This way, only one oxygen looks for waiting* **H**'s *at a time -- if there aren't enough* **H**'s *for the first oxygen, there won't be enough for any of the later oxygens either.*

Sample Exam1

1-Oct-19

3 October, 2019

_____ , _____
Last Name , First Name

This Page Intentionally Left Blank