CSE 451 – Midterm 1

Name: _____

1. [2 points]
   Imagine that a new CPU were built that contained multiple, complete sets of registers – each set contains a PC plus all the other registers available to user programs. The new CPU also contains a single new register that contains the index of the register currently in use. Assigning a value to that index register would be a privileged instruction.
   How might this CPU feature be useful to an operating system? What advantage would it have relative to a system that ran on a standard processor (that has only a single register set)?

   The OS would not have to save/restore sets of register to/from memory on each context switch. This might result in much faster context switch times than on a more conventional processor.

2. [5 points]
   I want to build an OS that supports multiprogramming. What facilities must my OS provide? What additional ones are likely to be useful, if not 100% necessary?

   I added a point for each idea corresponding to one of the following, independently of how it was worded (and up to a maximum of 5 points).
   - trap handling / timers
   - dual-mode / OS protection
   - process management/creation
   - memory management (allocation/protection)
   - scheduling/dispatching
   - IPC
   - virtual memory

   I didn't count anything having to do with IO/file systems one way or the other. I see that this is a reasonable thing to have answered, but it has nothing to do with multiprogramming.

3. [2 points]
   What does it mean for a process "to block"? Give an example of a situation where it might do so.

   The process is neither running nor on the run/ready queue. An example is a file read where the data must be fetched from disk.

4. [7 points]
The following code is intended to be compiled and then placed in a library. Is it thread-safe? That is, if a multi-threaded application contains unsynchronized calls to this routine, is the sequence of results returned guaranteed to correspond to a sequence that could be generated by some set of calls issued sequentially (one at a time)?
If so, informally but convincingly argue that it is. If not, modify it to make it thread-

```
unsigned int total;
unsigned int grandTotal = -1;

unsigned int MyLibRoutine (unsigned char a,
unsigned char b)
{
     unsigned int   product;
     unsigned int   left;
     unsigned int   right;

     product = a * b;

     left = a;
     right = b;

     total = 0;
     while (left > 0) {
          if (left & 1) {
               total += right;
          }
          left>>=1;
          right<<=1;
     }

     grandTotal = grandTotal + (total - product);
     return grandTotal;
}
```

safe. (The **details** of syntax won't matter.)

Three possible answers:

1. Create a new local unsigned int retVal and replace the return statement with "retVal = grandTotal; return retVal;", PLUS create a global mutex, lock it just before "total = 0;" and unlock it just before the (new) return statement.

2. Move the declaration of total to inside MyLibRoutine (i.e., make it local), then do the same as in #1 except put the lock/unlock only around the two statements affecting grandTotal.

3. Replace all the code and declaration inside MyLibRoutine with “return –1;”

5. [4 points]
Give one important of advantage of user-level threads over kernel threads? Of kernel threads over user-level threads? The two should not be opposites sides of a single coin – they should be independent ideas.

ULT’s leave thread scheduling decisions to application code, so thread management can be customized to individual applications. Additionally, thread creation and synchronization operations/variables are faster because you don’t have to do a syscall.

Kernel threads can make effective use of multiple processors on a multiprocessor. More importantly, if a single kernel thread of an application blocks, the other threads can continue, something not possible when a single kernel thread is used to multiplex multiple ULTs.

6. [3 points]
Why must a system call involve a mechanism like a trap? That is, why can’t a system be built where calls to the operating system are just regular procedure calls, exactly like calling another procedure in your own application?

The trap does two things: (1) switches the processor to privileged mode, and (2) forces a jump to a specific address contained in a (protected) table. Both are needed to have safe dual-mode operation.

7. [2 points]
Give an example of:
(a) the name an application might use to identify a particular process in a call to the Linux kernel

2108

(b) the name an application might use to identify a particular open file to the Linux kernel

6

8. [3 points]
What is the difference between a binary semaphore and a lock?

I took two answers here:

(A) None.
(B) A semaphore implies the existence of a thread queue (i.e., blocking), while a lock doesn’t imply anything either way (i.e., it’s an implementation decision).

(A) is the preferable answer.

9. [3 points]
The text presents "monitors" as a synchronization primitive. Monitors are a language-level facility, meaning they're recognized and enforced by the compiler. Along with inherent mutual exclusion of method (procedure) execution on a single object, monitors provide condition variables. The operations on condition variables are **wait(condition_var)** and **signal(condition_var)**.

The pthreads (and other) thread library also provides a synchronization like a condition variables, with operations **wait** and **signal**. The wait call has the following signature

```
int  pthread_cond_wait(pthread_cond_t  *cond,
                       pthread_mutex_t *mutex);
```

Why does the pthread library implementation of condition variables require a mutex as an argument to wait?

The key is that the thread wants to suspend itself while holding a lock. The lock has to be released while it's suspended. Releasing the lock and suspending must happen atomically (together, "uninterruptably") – if they're two separate operations, neither order works (if you suspend first, you can't unlock, and if you unlock there's a race in getting to the suspend).

10. [5 points]
For each of the following inter-process communication (IPC) mechanisms, give an example of a situation in which it would be an appropriate choice **and** the other choices would not be appropriate. (So, for instance, an answer like "to send either a 0 or a 1 from some process A to some process B" would not be an answer to any of these.)

The question asked for an example of each, but I'm going to give the idea of each because there are so many specific examples.

Signals
Reception of the communication is *asynchronous* from the point of view of the receiver – there is no "read" operation coded into the receiver that causes the communication to be received.

Pipes (specifically, anonymous pipes, which are the ones we've talked about in class and you've used in assignments)

Transparently created (no modification of source required) communication between two concurrently executing processes.

I also took “streaming” or “synchronized” communication, although sockets provide this as well.

Files

Communication between processes that may run at different times.

I also took “many receivers” and “persistence” and “random access.”

Sockets

Communication between (concurrently executing) processes on distinct machines.

Shared Memory

Fine-grained interaction through program variables (plus “low cost”).

11. [4 points]
In the current assignment (Project 2), you’re asked to do a pair-wise comparison of all files in a file system subtree by computing MD5 hashes of each file’s contents and comparing the hashes. This question is basically “Why? Why are MD5 hashes useful here?”

In particular, it seems reasonable to assume that most files will be very different from each other. So, if you compared the contents of two files for equality you’d have a decision after examining a small number of bytes – they’d probably differ after at most a few bytes, and you’d never have to read either file in its entirety. In contrast, to compute the MD5 hash you do have to read all of every file.

So, the question is, are MD5 hashes useful for doing this pair-wise file comparison? (And, as usual, justify your answer either way.)

Well, there are two things going on here: Is a hash useful?, and Is MD5 a good hash?

The answer to the first is “yes,” because two files can be equal only if there hashes are equal (for any reasonable hash scheme). So, you can hash one and compare it to all others by testing if any other produced the same hash. If not, the common case, then no. If a match is found, then you may have to investigate further.

Is MD5 useful? Well, actually I’m not sure. It REQUIRES you to read the full contents of every file. That’s bad. On the other hand, if you get a match on the hash values, you can assume you’re done – they’re equal. It’s hard to know if it would be faster than using some cheaper hash – for example, concatenate the first, last, and middle bytes of the file with the file length. That’s 7 bytes, and is likely to work

pretty well, and doesn't need the full file. ("Work pretty well" means you probably won't get too many matches on that hash when the files aren't equal.)

12. [2 points]
What is the advantage of the "microkernel" approach to structuring an operating system implementation (relative to the "monolithic kernel" approach)? (The "layered implementation" talked about in the book is an example of a monolithic kernel.)

I took "smaller, so easier to write/debug/maintain" and "pushes policy up to the user-level, where it's easier to modify." I didn't take "faster", because that's an anti-answer – monolithic kernels are justified by their speed.

13. [4 points]
Here is a **modified** version of the mutual exclusion code given in the text as "Algorithm 3" (a correct, two-process solution). Is **this modified version** correct? Briefly justify your answer.

```
flag[i] = true;
turn = i;
while (flag[j] && turn == j);

<critical section>

flag[i] = false;
```

This doesn't work. The simplest case is
- one process shows up and enters the critical section (because turn==j is false)
- while the first process is in the critical section, another shows up and executes this code
- it also enters (because it sets turn to i just before testing that it equals j)
- so, no mutual exclusion