

FINAL EXAM

- This test contains 8 questions with a total of 70 points
 - 1-4 5 points each
 - 5-7 10 points each
 - 8 20 points
- You have 1 hour and 50 minutes to complete this exam.
- You may **not** use your text, or any other reference material.
- You may not use any electronics devices during the exam.
- Do not turn this page or turn over the exam until instructed to do so.

Hints

- Don't spend too much time on short answer questions. If you don't know the answer right away, move on and come back.
- There may be some questions that will require more time to think and organize your thoughts. If you find yourself stuck on one of these questions, save it until you have answered all the questions you can answer quickly. If you spend too much time on one question you may get frustrated and that will impair your ability to answer the other questions.
- Don't turn your test in early. If you have time left over, use it to review your answers. Students who turn tests in early often make trivial mistakes that they would catch if they reviewed their answers.
- Students that get the highest grades usually go over their answers several times.
- The space below a question **does not** indicate how long of an answer I am expecting. Sometimes I want to start the next question on a new page.
- Double check that you understand the question before you answer it. If you don't understand the question, ask me.

- 1) (5 points) A file descriptor (fd) is an ordinary integer. How is that integer used by the operating system when you pass it to a function like read()?

```
ssize_t read(int fd, void *buf, size_t count);
```

For each process the operating system keeps a table of all the open files. The file descriptor is an index into that table. The table contains a pointer to the internal data structure necessary to perform operation.

- 2) (5 points) When sending binary files across a socket there is no character that can be used to indicate the end of the file (the file could contain any pattern of bits). Describe a mechanism that can be used to definitively identify the end of a file (there are two common methods, describe only one of them).

The size of the file can be sent first. Then the receiver can receive bytes until it has received the correct number of bytes.

An alternative method is to create a new connection for each file and close the connection after sending the last byte. The receiver will know it has received all the bytes when the connection closes.

- 3) (5 points) Explain how the shell uses the environment variable PATH.

PATH is a string that contains a set of directories separated by colons.

When the user types a command at the shell prompt, if the command is the full path of a file (starts with a /) the shell executes the file (assuming it exists and that it is executable). Otherwise, the shell looks for the given command in each directory in the PATH.

- 4) (5 points) ctime(t) converts calendar time into a string of the form: "Wed Jun 30 21:49:08 1993\n"

```
char *ctime(const time_t *calendar_time);
```

This function is not thread safe. What must it do that makes it not thread safe? How could the function be rewritten to make it thread safe.

Since we know the function is not thread safe it must use the same buffer each time it is called. Thus if two threads call the function the buffer will become corrupt.

We could make the function thread safe by passing in a buffer into which it would put the string form of the given time. If the buffer was passed into the function then when multiple threads call the function they would not interfere with each other.

- 5) (10 points) Outline how UNIX pipes, fork(), and exec() are used to implement pipes in the UNIX shell. For example, how `$ ls | grep | sort` would be handled by the shell.

The shell creates a pipe for each neighboring command to communicate. In the above example, one pipe for ls and grep to communicate and one pipe for grep and sort to communicate. Then it forks a process to run each command. Each child process connects its input and output to the appropriate pipe and then calls exec() on the target command.

- 6) (10 points) Describe the concept of a threadpool. What are the advantages of a threadpool?

Threadpool is a group of reusable threads. At program start up a group of threads is created and each is blocked. When a thread is needed by the program one of the existing threads is assigned the task at hand. When the thread is done it is added back to the pool of threads. Alternatively threads can be created during runtime when needed and then added to the threadpool when finished the assigned task. There is overhead involved in creating and destroying threads. The main advantage is that it is much faster to assign an existing thread to a task than to create a new thread.

- 7) (10 points) Assume the following test and set function is atomic. Use it to solve the critical section problem.

```
bool test_and_set(bool &value)    // & means that value is passed by
reference
{
    bool temp = value;
    value = true;
    return temp;
}
```

```
bool locked = false; // global variable shared by all threads
```

```
pi:
```

```
    while (test_and_set(locked) == true)
        ; // do nothing
```

```
    critical section
```

```
    locked = false;
```

8) (20 points) Demonstrate how pthreads, mutex locks, and conditional variables can be used to synchronize access to a shared bounded buffer. Specifically, write a producer thread that generates integers and places them in a shared buffer. The producer should not block unless the buffer is full. Write a consumer thread that reads integers from the buffer. The consumer should not block unless the buffer is empty. Assume the buffer is of size N.

```
void *producer_code(void *)
{
    int next_write_slot = 0;
    for (int i = 0; i < 10; i++)
    {
        pthread_mutex_lock(&mutex);
        while (size == N)
            pthread_cond_wait(&full, &mutex);
        size++;
        buffer[next_write_slot] = produce_next_value();
        pthread_mutex_unlock(&mutex);
        pthread_cond_signal(&empty);
        next_write_slot = (next_write_slot + 1) % N;
    }
}

void *consumer_code(void *)
{
    int next_read_slot = 0;
    for (int i = 0; i < 10; i++)
    {
        pthread_mutex_lock(&mutex);
        while (size == 0)
            pthread_cond_wait(&empty, &mutex);
        size--;
        consume_next_value(buffer[next_read_slot]);
        pthread_cond_signal(&full);
        pthread_mutex_unlock(&mutex);
        next_read_slot = (next_read_slot + 1) % N;
    }
}
```