

### FINAL EXAM

- This test contains 7 questions worth a total of 72 points & 1 extra credit question worth 8 points
  - 1-5           8 points each
  - 6             12 points each
  - 7             20 points
  - 8             8 extra credit points
- You have 170 minutes to complete this exam.
- You may **not** use your text, or any other reference material.
- 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. It is very likely you will not get any points if you don't understand the question. **If you don't understand the question, ask me.**

- 1) (8 points) Explain how two process (which are not parent/child) can use a message queue to communicate. Make sure you explain the concept of a key.

Step 1: need to make sure that both process are using the same queue. This can be achieved by both processes passing the same integer key to `msgget()`. The key can either be hard coded (a constant in the code) or `ftok()` can be used to generate a key from a given filename. The drawback of a hard coded key is that some other application may be using the same key by chance.

Step 2: Both processes send the key to the `msgget()` function which returns a message queue id.

Step 3: The processes can send messages to the message queue and receive messages from the message queue simply by passing the message queue id to the send (`msgsnd()`) and the receive (`msgrcv()`) functions.

The messages can be any size as long as the first part of the message is a long that contains the message type.

- 2) (8 points) How can a parent process terminate a child process so that the child process can finish any critical tasks (such as finish writing to a file)?

There are multiple ways to achieve this. One is to use signals:

1. The child process sets up a signal handler to handle a signal from the parent (probably a `SIGUSR1` signal).
2. When the parent wants the child to terminate it can send a signal to the child.
3. The child's signal handler can set an *exit now* flag that the rest of the program periodically checks (such as after finishing writing a file). When the child process “notices” that the *exit now* flag has been set it can call `exit()`.

- 3) (8 points) The which command takes an argument and indicates which (if any) of the directories listed in the PATH environment variable contain an executable file with the same name as the argument. Outline how one would implement which. Include the system calls that would have to be called. If you forget the exact name of the system calls, state what has to be done (for example, if you forgot that the `getenv(char *name)` function returns the value of an environment variable simply write “use a system call to get the value of the environment variable”). You don't have to write the code, just provide an outline of what has to happen to implement which.

The first step is to get the value of the PATH variable by calling the `getenv()` function. The PATH value contains a series of directories separated by colons. This string will have to be parsed into its components (this can be done from scratch or a function like `strtok()` can be used).

Each directory in the PATH must be read (this is accomplished using `opendir()` and `readdir()`). The filename of each file (sub directories should be skipped) in this directory must be compared to the given target. If it matches then `stat()` can be used to find out if the user has execute permission on the file. If so, the full path of the file is printed.

4) (8 points) Consider the code:

initialization:	thread1	thread2	thread 3
int y = 42;	y = y + 1;	y = y + 2;	y = y + 3;

Give all the possible values for y after running this code. Give at least one explanation for each value.

- y = 43      thread1 starts but is interrupted after getting the value of y but before setting the value. Then thread2 and thread3 run until completion. Finally thread 1 completes.
- y = 44      thread2 starts but is interrupted after getting the value of y but before setting the value. Then thread1 and thread3 run until completion. Finally thread 2 completes.
- y = 45      thread3 starts but is interrupted after getting the value of y but before setting the value. Then thread1 and thread2 run until completion. Finally thread 3 completes.
- y = 46      thread1 executes, then thread3 starts but is interrupted after getting the value of y but before setting the value. Then thread2 runs until completion. Finally thread 3 completes.
- y = 47      thread2 executes, then thread3 starts but is interrupted after getting the value of y but before setting the value. Then thread1 runs until completion. Finally thread 3 completes.
- y = 48      The three threads run one after the other (not interleaved) in any order.

5) (8 points) Describe how you implemented the inactive() function in your threadpool assignment. In order to simplify the question, assume that blocking is always true.

My program has an vector of available threads. If the vector contains all the threads, then all the threads are inactive and inactive() returns.

If the vector does not contain all the threads, then inactive()'s thread is blocked on a conditional variable:

```
// error checking has been removed so the algorithm is easier to understand
pthread_mutex_lock(&m_available_mutex);
while (m_available.size() != m_num_threads)
    pthread_cond_wait(&m_inactive_conditional, &m_available_mutex);
```

When a thread completes its function it adds itself to the vector of available threads. If the vector now contains all the threads, the m\_inactive\_conditional conditional variable is signaled (using a broadcast in case multiple threads called inactive()).

- 6) (12 points) The `system()` function takes a string and creates a new process that executes the string as if it were typed at the shell's command prompt. For example, `system("ls -l");` will execute the program `ls` passing `"-l"` as a command line argument. The command's output will be written to standard output and its exit status is returned by `system()`.

Write a pseudo-code implementation of the `system()` function without calling the actual `system()` function (you will need to call other system functions). Parsing the arguments is tedious, you don't have to provide the code, just explain what has to be done.

```
// in real programs the return status of system calls must be checked
int system(const char *command)
{
    // the single string command has to be broken up into separate strings and
    // put in an argv struct (null terminated array of C-style strings)
    char **argv = parse_command(command);

    if (fork() == 0)
    {
        if (execvp(argv[0], argv) == -1)
        {
            perror("execvp() failed");
            return -1;
        }
        assert(false);
    }
    else
    {
        int status;
        wait(&status);
        return WEXITSTATUS(status);
    }
}
```

7) (20 points) Write code to simulate a car ferry. The ferry travels between ports A and B. It leaves port A for port B at 1:00, it will leave port B for port A at 2:00. The ferry can hold **n** cars and leaves on time no matter how many cars are on it. Using pthreads and conditional variables, write code to implement the car threads and the ferry thread. Use integers to represent time (e.g. ferry goes from A to B at time 1, B to A at time 2, A to B at time 3, ...) You can assume there is a **non-blocking** conditional variable signal function that delays sending the signal.

```
pthread_cond_delayed_signal(&conditiona_variable, delay); // send signal after waiting delay
```

The answer will be posted at some future time (maybe soon, maybe not).

8) (8 points extra credit) How would you implement the delayed conditional variable signal in the last problem? Document your assumptions about how the function works.