Exam 1 Makeup: Fork, Exec, etc.

So in this exam we want to build a system to benchmark interprocess communication over pipes. We have several senders and receivers and wish to measure the time it takes to communicate a standard set of messages from the sender to the receiver. The basic idea is to test each sender-receiver pair in parallel and if the test successfully completed, report the time required.

NOTE We have ordered the steps in what we think is the most convenient implementation order. If you have difficulty with one step though, some of them can be completed out of order so you can try the later parts too.

Table of Contents

Makefile (10 points)

So we're going to use several extra executables for this problem: fast_receiver, fast_sender, slow_receiver, slow_sender, unreliable_receiver, and unreliable_sender. Each of these includes common.h and uses functions in common.c. makescript will build all of these extra executables except slow_sender. You can compile slow_sender them by hand like this:

```
gcc -o slow_sender slow_sender.c common.c
```

But that is inefficient. Make a makefile that builds slow_sender using the 2-step process (i.e. compile each .c file to a .o individually, then link the .o files into an executable). For max credit, your system should only rebuild what is necessary based on when files are updated.

Feel free to further enhance your Makefile to build the other five executables (perhaps by calling makescript) and para_benchmark (the file you'll be editing for this assignment), but only slow_sender will be looked at in terms of grading the Makefile.

Running in parallel (20 points)

You can compile the given code like this (again, feel free to add this to your Makefile if you want):

```
gcc para_benchmark.c common.c -o para_benchmark
```

You can run the executable like this:

```
./para_benchmark
```

And the output you see should look like this:

```
prompt> ./para_benchmark
I would like to test ./fast_receiver with ./fast_sender
Done testing ./fast_sender with ./fast_receiver, 1.000103 sec required
I would like to test ./fast_receiver with ./slow_sender
Done testing ./slow_sender with ./fast_receiver, 1.000138 sec required
I would like to test ./slow_receiver with ./fast_sender
Done testing ./fast_sender with ./slow_receiver, 1.000140 sec required
I would like to test ./slow_receiver with ./slow_sender
Done testing ./slow_sender with ./slow_receiver, 1.000140 sec required
prompt>
```

You should notice a few problems:

1. The code is just saying it would like to run test a sender and receiver, but not actually running them.
2. The code is not reporting the time required for each combination.
3. The code is not running in parallel. It performs each test one at a time.

To start with, use fork to solve problem #3. Your output should look like this:

```
prompt> ./para_benchmark
I would like to test ./fast_receiver with ./fast_sender
I would like to test ./fast_receiver with ./slow_sender
I would like to test ./slow_receiver with ./fast_sender
I would like to test ./slow_receiver with ./slow_sender
prompt> Done testing ./fast_sender with ./fast_receiver, 1.000124 sec required
Done testing ./slow_sender with ./fast_receiver, 1.000125 sec required
Done testing ./fast_sender with ./slow_receiver, 1.000106 sec required
Done testing ./slow_sender with ./slow_receiver, 1.000089 sec required
```

In this example, the child processes are the ones printing "done testing ...." messages.

Note that the order of the sender-receiver pairs could well be different (they are happening in parallel after all). The key is that all the "I would like to test ...." messages print before any of the "done testing ...." messages.

NOTE: The parent does not need to wait for the children to complete before exiting. Eventually, you'll need to do this, but it's not required for this part. If you don't wait though, you might see the prompt print before all the "Done testing ...." messages (as it does in my example). That's fine. If you did make the parent wait though, that's fine too.

Exec the sender (20 points)

Use exec to make your forked children actually invoke the appropriate sender. The senders require a command line argument. For this part, simply pass in -1. You do not need to print out the "Done testing ...." message with the timing data. We'll add that in the next part. You also don't need to preserve the sleep – it's built in to the senders.

In addition, have the parent wait until all of the children are complete and print "Testing complete."

Your output should look like this:

```
prompt> ./para_benchmark
testing ./slow_sender
waiting for all tests to complete
testing ./fast_sender
testing ./slow_sender
testing ./fast_sender
starting slow_sender
starting fast_sender
starting slow_sender
starting fast_sender
slow_sender: fake mode (slow)
slow_sender: fake mode (slow)
fast_sender: fake mode
fast_sender: fake mode
Testing complete
prompt>
```

NOTE: This part and the next are closely related and it may help to think about them together.

Collect benchmark data (15 points)

Measure the time it takes for each sender to execute and restore the "Done testing . . . ." message. Do this by introducing an additional fork that starts the timer, forks and execs the child sender process, waits for it to complete, and then prints timing information.

Your output should look like this:

```
prompt> ./para_benchmark
testing ./fast_sender
waiting for all tests to complete
waiting for ./fast_sender to finish
testing ./slow_sender
waiting for ./slow_sender to finish
testing ./fast_sender
waiting for ./fast_sender to finish
starting fast_sender
testing ./slow_sender
waiting for ./slow_sender to finish
starting slow_sender
starting fast_sender
starting slow_sender
fast_sender: fake mode
Done testing ./fast_sender with ./fast_receiver, 1.000860 sec required
fast_sender: fake mode
Done testing ./fast_sender with ./slow_receiver, 1.000907 sec required
slow_sender: fake mode (slow)
Done testing ./slow_sender with ./fast_receiver, 2.000973 sec required
slow_sender: fake mode (slow)
Done testing ./slow_sender with ./slow_receiver, 2.000999 sec required
Testing complete
prompt>
```

Note that we've made the slower processes run slower than the fast ones, even in fake mode, so you should see output consistent with above.

NOTE: This part and the previous one are closely related and it may help to think about them together.

Exec the receiver (10 points)

Use exec to make your forked children also invoke the appropriate receiver. The receivers will eventually need a command line argument, but for now do not pass a command line argument. Make sure you measure the time it takes for each sender-receiver pair to run and print the results in the "Done testing ...." message.

Your output should look like this:

```
prompt> ./para_benchmark
testing ./fast_sender with ./fast_receiver
waiting for all tests to complete
waiting for ./fast_sender and ./fast_receiver to finish
testing ./slow_sender with ./fast_receiver
testing ./slow_sender with ./slow_receiver
waiting for ./slow_sender and ./slow_receiver to finish
starting fast_sender
waiting for ./slow_sender and ./fast_receiver to finish
testing ./fast_sender with ./slow_receiver
waiting for ./fast_sender and ./slow_receiver to finish
starting slow_receiver
starting fast_receiver
starting slow_sender
starting fast_receiver
starting slow_receiver
starting slow_sender
starting fast_sender
fast_sender: fake mode
fast_receiver: fake mode
Done testing ./fast_sender with ./fast_receiver, 1.001415 sec required
fast_receiver: fake mode
fast_sender: fake mode
slow_sender: fake mode (slow)
slow_sender: fake mode (slow)
Done testing ./slow_sender with ./fast_receiver, 2.002118 sec required
slow_receiver: fake mode (slow)
Done testing ./slow_sender with ./slow_receiver, 3.001036 sec required
slow_receiver: fake mode (slow)
Done testing ./fast_sender with ./slow_receiver, 3.001414 sec required
Testing complete
prompt>
```

Shared memory and dealing with failures (15 points)

In this step of the process we want determine the fastest overall send receive pair. Once all the parallel tests have finished the parent should print "The fastest overall runtime was XXXXXXX seconds".

Use shared memory to allow the children to communicate the benchmark results to the parent. Create an array of doubles with mmap() and modify the children to store the time required for each benchmark test in the array.

When a sender or receiver fails it's important that it's result not be used to calculate the final best time. Check the exit status of the sender or receiver in each pair and if either are unsuccessful enter -1 into the array. To check unsuccessful tests you can include unreliable_sender and unreliable_receiver in your testing pairs by changing nrcvr and nsndrs to 3.

After all the children have completed, the parent should be able to iterate through the array and find the smallest best time (obviously avoiding any -1s).

Here's what my output looks like:

```
prompt> ./para_benchmark
testing ./fast_sender with ./fast_receiver
waiting for ./fast_sender and ./fast_receiver to finish
testing ./slow_sender with ./fast_receiver
waiting for all tests to complete
waiting for ./slow_sender and ./fast_receiver to finish
testing ./unreliable_sender with ./fast_receiver
waiting for ./unreliable_sender and ./fast_receiver to finish
testing ./fast_sender with ./unreliable_receiver
starting fast_receiver
waiting for ./fast_sender and ./unreliable_receiver to finish
testing ./fast_sender with ./slow_receiver
starting fast_receiver
waiting for ./fast_sender and ./slow_receiver to finish
testing ./slow_sender with ./unreliable_receiver
starting fast_sender
waiting for ./slow_sender and ./unreliable_receiver to finish
testing ./unreliable_sender with ./unreliable_receiver
starting fast_receiver
testing ./unreliable_sender with ./slow_receiver
waiting for ./unreliable_sender and ./unreliable_receiver to finish
waiting for ./unreliable_sender and ./slow_receiver to finish
starting fast_sender
starting unreliable_receiver
testing ./slow_sender with ./slow_receiver
waiting for ./slow_sender and ./slow_receiver to finish
starting unreliable_sender
starting slow_sender
starting unreliable_receiver
starting slow_receiver
starting unreliable_receiver
starting unreliable_sender
starting unreliable_sender
starting fast_sender
starting slow_receiver
starting slow_receiver
starting slow_sender
starting slow_sender
fast_receiver: fake mode
fast_receiver: fake mode
fast_sender: fake mode
fast_receiver: fake mode
Done testing ./fast_sender with ./fast_receiver, 1.002493 sec required
fast_sender: fake mode
unreliable_receiver: fake mode
Done testing ./fast_sender with ./unreliable_receiver, -1.000000 sec required
unreliable_sender: fake mode
Done testing ./unreliable_sender with ./fast_receiver, -1.000000 sec required
unreliable_sender: fake mode
unreliable_sender: fake mode
unreliable_receiver: fake mode
Done testing ./unreliable_sender with ./unreliable_receiver, -1.000000 sec required
unreliable_receiver: fake mode
fast_sender: fake mode
slow_sender: fake mode (slow)
Done testing ./slow_sender with ./fast_receiver, 2.003382 sec required
slow_sender: fake mode (slow)
Done testing ./slow_sender with ./unreliable_receiver, -1.000000 sec required
slow_sender: fake mode (slow)
slow_receiver: fake mode (slow)
Done testing ./unreliable_sender with ./slow_receiver, -1.000000 sec required
slow_receiver: fake mode (slow)
Done testing ./fast_sender with ./slow_receiver, 3.003315 sec required
slow_receiver: fake mode (slow)
Done testing ./slow_sender with ./slow_receiver, 3.002664 sec required
The fastest overall runtime was 1.002493 seconds
Testing complete
prompt>
```

Create a pipe and pass the file descriptor to the sender and receiver (10 points)

Remember that when process are forked one of the few things that the child shares with the parent are open files. We will take advantage of this using pipes to allow interprocess communication. Create a pipe and pass the appropriate file descriptor to the sender and receiver on the command line instead of -1.

To send the pipe file descriptors to the execed processes we must pass them as strings. Here's how to convert:

```
char data[100];
snprintf(data, 100, "%d", fds[0]);
```

Note that your process must close unused fd ends before execing. Also Be sure to close both ends of the pipe in the parent or you can get a bug where the receives never finish.

Expected output:

```
prompt> ./para_benchmark
testing ./fast_sender with ./fast_receiver
waiting for ./fast_sender and ./fast_receiver to finish
testing ./slow_sender with ./fast_receiver
waiting for all tests to complete
waiting for ./slow_sender and ./fast_receiver to finish
testing ./fast_sender with ./unreliable_receiver
testing ./unreliable_sender with ./fast_receiver
waiting for ./fast_sender and ./unreliable_receiver to finish
waiting for ./unreliable_sender and ./fast_receiver to finish
testing ./slow_sender with ./unreliable_receiver
waiting for ./slow_sender and ./unreliable_receiver to finish
testing ./unreliable_sender with ./unreliable_receiver
testing ./unreliable_sender with ./slow_receiver
waiting for ./unreliable_sender and ./unreliable_receiver to finish
starting fast_receiver
starting fast_receiver
testing ./slow_sender with ./slow_receiver
waiting for ./slow_sender and ./slow_receiver to finish
starting slow_sender
starting fast_receiver
starting fast_sender
waiting for ./unreliable_sender and ./slow_receiver to finish
starting unreliable_sender
unreliable_sender random failure
starting fast_sender
starting unreliable_receiver
Done testing ./fast_sender with ./fast_receiver, 0.003422 sec required
unreliable_receiver random failure
Done testing ./fast_sender with ./unreliable_receiver, -1.000000 sec required
Done testing ./unreliable_sender with ./fast_receiver, -1.000000 sec required
```

```
starting unreliable_receiver
testing ./fast_sender with ./slow_receiver
waiting for ./fast_sender and ./slow_receiver to finish
starting unreliable_sender
unreliable_sender random failure
starting fast_sender
starting slow_receiver
starting unreliable_receiver
starting slow_sender
starting slow_receiver
starting slow_receiver
starting unreliable_sender
starting slow_sender
unreliable_sender random failure
unreliable_receiver random failure
Done testing ./unreliable_sender with ./unreliable_receiver, -1.000000 sec required
Done testing ./unreliable_sender with ./slow_receiver, -1.000000 sec required
Done testing ./fast_sender with ./slow_receiver, 13.793236 sec required
Done testing ./slow_sender with ./slow_receiver, 14.165205 sec required
Done testing ./slow_sender with ./unreliable_receiver, 15.380452 sec required
Done testing ./slow_sender with ./fast_receiver, 15.799632 sec required
The fastest overall runtime was 0.003422 seconds
Testing complete
prompt>
```

NOTE: The senders and receivers don't print when transmitting. However, if you want to see what they're doing you can set VERB to 1 in common.h.

Submitting your solution

Submit your Makefile and para_benchmark.c to the dropbox on Moodle.