CSC 412 – Operating Systems

Lab Session 09, Spring 2022

Monday, April 18th and Wednesday, April 20th, 2022

What This Lab Is About

In this lab we are going to develop communication between a parent process and multiple child processes that use dedicated communication threads to listen on a pipe.

The Handout

The handout for this lab consists of the code for two-way communication between a parent process and its child process, using unnamed pipes.

Task 1: Build and run the handout code

The current version of the program asks the user for a "command," which it sends to the child process. The child process in return sends a "handshake" message back to the parent process.

The parent process can handle three kinds of commands:

- `D <value>` to send a value to the child process child (for example, `D 68` to send the `int` value 68 to the child process). For this command, the child process returns an `OK` handshake message.
- `S` to request from the child process the sum of all the values that it has received so far;
- `END` to tell the child process to terminate.

Task 2: Support Communication with Multiple Child Processes

I have commented out Line 22 of the handout, which defines and initializes a constant named `NUM_PROCESSES`. This should give you an idea of where we are headed.

Set up pipes to communicate with the required number of child processes, then create them. Modify the output of the parent and child processes so that they mention the index of the child with which communication is going on.

The parent process will now receive three kinds of commands:

- `D <child id> <value>` to send a value to a specific child (for example, `D 3 68` to send the `int` value 68 to Child 3). For this command, the child process returns the usual `OK` handshake message.

- `S <child id>` to request from a specific child process the sum of all the values that it has received so far (for example, `S 4` to request the sum of all values received by Child 4);
- `END` to tell all child processes to terminate.

As usual in all these labs, don't waste time on data validation (is the child index valid? I). Obviously, the child processes will need to store the values that they receive in a `vector<int>` or `list<int>`.

Task 3: Dedicated Communication Thread

One major weakness of our communication system is that child processes are blocked, waiting for new data from the parent. Of course, since we are only computing a sum, they really haven't got much to do. Just keep in mind that our sum task is just a "toy" task. In a real system (which we haven't got the time to implement) the child process may have lots of things to do when it gets new data.

For this purpose, we are going to create in the child processes a thread dedicated to listening on the pipe. When new data arrives, it adds the data to the list stored. If a request for the summ arrives, it sends the value computed. This poses naturally a race condition problem, which we are going to solve by adding three variables to the child process:

- An `int` variable storing the sum computed so far;
- A `bool` variable indicating whether the sum is "up to date" (that is, including the last values sent);
- A `mutex` lock controlling access to the list of value, sum, and the "up to date" flag.

The communication thread should wait for the sum to be up to date before sending it to the parent process. The main thread of the child process is going to sleep for a while, check the "up to date" flag, update the sum if needed, then go back to sleep.