Name: _______________________________________ Student ID# _____________________

Assignment #4 (60 pts): Concurrent UNIX Processes and Shared Memory

Due Date: WED 13 APR 2016

The goal of this assignment is to become familiar with concurrent processing in Unix/Linux using shared memory.

You will write a program that uses multiple processes to simulate a swim mill to show the behavior of a fish swimming upstream. The swim mill is represented as a one-dimensional array such that each element indicates the distance from the shore. The fish will occupy one and only one of these elements. The presence of a fish is indicated by changing the integer in the specified element. For example, we can represent the fish by the character F. By default, we will have the fish swimming right in the middle of the stream.

Somewhere upstream, a pellet is dropped into the stream. Our fish sees this pellet traveling towards it and moves sideways to enable its capture. If the pellet and the fish are in the same position at any point, the fish is said to have eaten the pellet. Of course, it is possible for our fish to miss the pellet if the fish cannot move close to it in time.

Different types of processes needed for this project are as follows:

- A set of `pellet` processes: Each pellet process drops a pellet at a random distance from the fish. We will call this process `pellet`. The `pellet` will start moving towards the fish with the flow of the river. For simplicity, we will assume that the `pellet` moves in a straight line downstream.

- A process called `fish`: the `fish` sees the pellet as it is dropped and moves one unit left or right to start aligning itself with the `pellet`. As the `pellet` moves towards the `fish` and the `fish` is at the same distance from the shore as the `pellet`, it stops changing its sideways position and devours the `pellet` when they are coincident. After eating the `pellet`, or if the `pellet` is missed, the `fish` returns to swimming midstream.

- A `coordinator` process: It is responsible for creating the `fish` process, `the` pellet processes, and coordinating the termination of pellets. We could have more than one `pellet` at any time. Note that the position of `pellet` and `fish` are maintained by their respective processes.

The `coordinator` process also sets a timer at the start of computation to 30 seconds. If computation has not finished by this time, coordinator kills all the children and grandchildren, and then exits. Make sure that you print appropriate message(s).

In addition, `coordinator` should print a message when an interrupt signal (^C) is received. Make sure that all the children/grandchildren are killed by `coordinator` when this happens, and all the shared memory is deallocated. The grandchildren kill themselves upon receiving interrupt signal but print a message on `stderr` to indicate that they are dying because of an interrupt, along with the identification information. Make sure that the processes handle multiple interrupts correctly. As a precaution, add this feature only after your program is well debugged.

Implementation

The code for `pellet`, `fish`, and `coordinator` processes should be compiled separately and the executables be called `pellet`, `fish`, and `swim_mill`, respectively. The program should be executed by calling: `swim_mill`

Each `pellet` process prints its *process id*, its *position* and whether it was eaten or missed, before exiting. Each time the coordinator gets a result from a `pellet`, it prints the pid of the `pellet`.

The coordinator will set up the two-dimensional array in shared memory, and write the result into a file after each child is finished. The fish will be restricted to move in the last row of the 2D array. The `swim_mill` process will create a pellet at random intervals, with the pellet being dropped into the swim mill at a random distance from the fish. The process pellet will increment its position and will terminate after the pellet has reached the last row, whether it is eaten or not.

The pellet can be represented by `0x80` and can be moved to location (x, `y)` by the statement `L[x][y] |= 0x80`. This will allow the pellet and the fish to be able to coincide. The process of eating is performed by getting rid of the most significant bit to return the fish to original state.

The process `fish` will scan the entire array and will focus on a pellet if it detects one, to arrange itself in its line of drift. Meanwhile, if another pellet is found that happens to be closer, and eatable, the fish will go for it.

Other points to remember: You are required to use `fork`, `exec` (or one of its variants), `wait`, and `exit` to manage multiple processes. Use `shmctl` suite of calls for shared memory allocation. Also make sure that you do not have more than twenty processes in the system at any time. You can do this by keeping a counter in `swim_mill` that gets incremented by `fork` and decremented by `wait`.

Hints

You will need to set up shared memory in this project to allow the processes to communicate with each other. Please check the man pages for `shmget`, `shmctl`, `shmat`, and `shmdt` to work with shared memory.

You will also need to set up signal processing and to do that, you will check on the functions for signal and abort. If you abort a process, make sure that the parent cleans up any allocated shared memory before dying. In case you face problems, please use the shell commands `ipcs` to find out any shared memory allocated to you and free it by using `ipcrm`.

You may also want to break down this assignment into parts in order to make it more manageable. First, focus on creating a coordinator process that sets up and can read and write to a shared memory location. Second, focus on getting a fish process that reads/writes to shared memory and can update its own location. Third, work on the pellet and its creation of random locations (pellet drops) and have it decrement its location through the shared memory. Finally, work on getting all three processes to coordinate and work in sync.

Timeline

While I am not strictly requiring a checkpoint for your project, you should have the coordinator process able to read and write to the shared memory within two weeks. The fish and pellet processes within another two weeks, and finally the coordination of all three within a week or two.

Finished Product

Demo your program for the instructor and submit a copy of your source code files through Beachboard Dropbox.