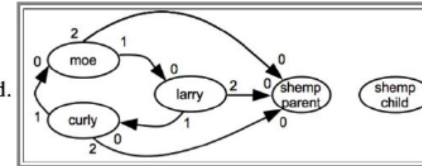
You are taking CS402, and your design team consists of you, Larry, Moe and Curly. The professor has made you all sign non-disclosure agreements, but for the life of you, you don't know why Larry, Moe or Curly would want to disclose their disastrous programming skills to anyone! Whatever, you have taken the design lead, and you have decided that the four of you will each write your own C programs:

- Moe's will be compiled into the executable **moe**;
- Larry's will be compiled into the executable **larry**, and
- Curly's will be compiled into the executable **curly**.
- None of the executables (not even yours) will have command line arguments, and none of them will be multi-threaded.



Your program (named **shemp.c**, of course), is going to be the master program. It is going to create processes so that when the system runs, it is going to look as drawin in the picture above. As you can see:

- The shemp program will fork a child. We'll call the two processes, "shemp parent" and "shemp child." They are going to share code.
- Moe's standard output goes to Larry's standard input.
- Larry's standard output goes to Curly's standard input.
- Curly's standard output goes to Moe's standard input.
- Moe, Larry and Curly's standard error will all go to the standard input of "shemp parent." Shemp parent is going to repeatedly read its standard input, process it a bit and write to standard output.
- Moe, Larry and Curly will all be children of "shemp child."

You may wonder why we're splitting **shemp** into a parent and a child? The reason is as follows. Larry, Moe and Curly can't be trusted. Their processes are supposed to talk to each other forever, but in reality, they may seg fault or go into infinite loops. The child is there to detect when any of them die. When that happens, it is going to kill the others and then exit. When the others are dead, the parent will be able to detect it. It will then print "NFS not responding, still trying" and go into an infinite loop. That way, when you demo your project, professor Birdwell will hopefully be suckered into thinking that we have network problems, and he won't realize that disaster has occurred.

Fun as it would be, I'm not making you write **shemp.c**. However, I'm sure you would write it flawlessly, using only the system calls **fork()**, **execlp()**, **close()**, **wait()**, **pipe()**, **dup2()** and **kill()**.

Now, answer the following questions:

- **Question 1**: How many times is "shemp-parent" going to call **fork()**?
- **Question 2**: How many times is "shemp-child" going to call **fork()**?
- **Question 3**: How many times is "shemp-parent" going to call **execlp()**?
- **Question 4**: How many times in total is **execlp()** called by any of the processes?
- **Question 5**: "Shemp-Child" is going to call **pipe()** three times. How many times is "shemp-parent" going to call **pipe()**?
- **Question 6**: How many times is the **larry** process going to call **dup2()**?
- **Question 7**: During the demo, disaster indeed occurs. **Larry** goes into an infinite loop and stops reading and writing. **Curly** doesn't bother reading from standard input, and instead goes into an infinite loop writing to standard output. And **moe** seg faults. How does "shemp child" detect that **moe** has died on a seg fault?
- **Question 8**: After detecting that **moe** has died, "shemp child" kills **larry** and tries to kill **curly**, but **curly** is already dead. Why?
- **Question 9**: How does "shemp-parent" detect that **larry**, **moe** and **curly** are dead?
- **Question 10**: When "shemp-parent" goes into its infinite loop, you want to make sure that there are no zombie processes. How many times must it call **wait()** to do that?

Clicker Questions for today

Let's review what's going to happen with the various processes:

Shemp Parent

- It will first call **pipe()** to set up a communication channel from stderr of **moe/larry/curly** to its stdin.
- It will then call **fork()** to create **shemp child**.
- It will **dup2()** the read end of the pipe onto its standard input.
- It will close both ends of the pipe.
- It will read from standard input and process the input.
- Eventually, when **moe/larry/curly** exit, it will read EOF from standard input, because there is no longer a write end of the pipe.
- It will call **wait()** so that **shemp child** doesn't become a zombie.
- It goes into its infinite loop.

Shemp Child

- It will call **pipe()** three times so that **moe/larry/curly** may communicate with each other.
- It will call **fork()** three times to create the **moe/larry/curly** processes.
- It will close all of the pipe file descriptors (there are 8 of these -- 4 pipes).
- It will call **wait()** to wait for the first of **moe/larry/curly** to exit.
- Once that happens, it will call **kill()** to kill the other two processes.
- It will **exit()**.

moe/larry/curly

- These will do the proper **dup2** calls so that they are talking with each other in the proper order, and so that stderr is going to the parent's pipe. Each process will do three **dup2()** calls -- one each for stdin, stdout and stderr.
- They will close all of the pipe file descriptors.
- They will call **execlp()** to execute **moe/larry/curly**.
- Once **moe/larry/curly** start running, they take over the process. The **execlp()** calls do not return.

Onto the questions

Given all of that information, most questions don't need explanation. Here are the answers:

- **Question 1:** One.
- **Question 2:** Three.
- **Question 3:** Zero.
- **Question 4:** Three.
- **Question 5:** One.
- **Question 6:** Three.
- **Question 7:** **wait()** returns with Moe's pid.
- **Question 8:** Since **moe** died, there is no read end for Curly's stdout. Therefore, Curly generates SIGPIPE and exits.
- **Question 9:** It reads EOF from stdin.
- **Question 10:** Once.

Clicker Questions for today

Let's review what's going to happen with the various processes:

Shemp Parent

- It will first call **pipe()** to set up a communication channel from stderr of **moe/larry/curly** to its stdin.
- It will then call **fork()** to create **shemp child**.
- It will **dup2()** the read end of the pipe onto its standard input.
- It will close both ends of the pipe.
- It will read from standard input and process the input.
- Eventually, when **moe/larry/curly** exit, it will read EOF from standard input, because there is no longer a write end of the pipe.
- It will call **wait()** so that **shemp child** doesn't become a zombie.
- It goes into its infinite loop.

Shemp Child

- It will call **pipe()** three times so that **moe/larry/curly** may communicate with each other.
- It will call **fork()** three times to create the **moe/larry/curly** processes.
- It will close all of the pipe file descriptors (there are 8 of these -- 4 pipes).
- It will call **wait()** to wait for the first of **moe/larry/curly** to exit.
- Once that happens, it will call **kill()** to kill the other two processes.
- It will **exit()**.

moe/larry/curly

- These will do the proper **dup2** calls so that they are talking with each other in the proper order, and so that stderr is going to the parent's pipe. Each process will do three **dup2()** calls -- one each for stdin, stdout and stderr.
- They will close all of the pipe file descriptors.
- They will call **execlp()** to execute **moe/larry/curly**.
- Once **moe/larry/curly** start running, they take over the process. The **execlp()** calls do not return.

Onto the questions

Given all of that information, most questions don't need explanation. Here are the answers:

- **Question 1:** One.
- **Question 2:** Three.
- **Question 3:** Zero.
- **Question 4:** Three.
- **Question 5:** One.
- **Question 6:** Three.
- **Question 7:** **wait()** returns with Moe's pid.
- **Question 8:** Since **moe** died, there is no read end for Curly's stdout. Therefore, Curly generates SIGPIPE and exits.
- **Question 9:** It reads EOF from stdin.
- **Question 10:** Once.

Clicker Questions for today

Let's review what's going to happen with the various processes:

Shemp Parent

- It will first call **pipe()** to set up a communication channel from stderr of **moe/larry/curly** to its stdin.
- It will then call **fork()** to create **shemp child**.
- It will **dup2()** the read end of the pipe onto its standard input.
- It will close both ends of the pipe.
- It will read from standard input and process the input.
- Eventually, when **moe/larry/curly** exit, it will read EOF from standard input, because there is no longer a write end of the pipe.
- It will call **wait()** so that **shemp child** doesn't become a zombie.
- It goes into its infinite loop.

Shemp Child

- It will call **pipe()** three times so that **moe/larry/curly** may communicate with each other.
- It will call **fork()** three times to create the **moe/larry/curly** processes.
- It will close all of the pipe file descriptors (there are 8 of these -- 4 pipes).
- It will call **wait()** to wait for the first of **moe/larry/curly** to exit.
- Once that happens, it will call **kill()** to kill the other two processes.
- It will **exit()**.

moe/larry/curly

- These will do the proper **dup2** calls so that they are talking with each other in the proper order, and so that stderr is going to the parent's pipe. Each process will do three **dup2()** calls -- one each for stdin, stdout and stderr.
- They will close all of the pipe file descriptors.
- They will call **execlp()** to execute **moe/larry/curly**.
- Once **moe/larry/curly** start running, they take over the process. The **execlp()** calls do not return.

Onto the questions

Given all of that information, most questions don't need explanation. Here are the answers:

- **Question 1:** One.
- **Question 2:** Three.
- **Question 3:** Zero.
- **Question 4:** Three.
- **Question 5:** One.
- **Question 6:** Three.
- **Question 7:** **wait()** returns with Moe's pid.
- **Question 8:** Since **moe** died, there is no read end for Curly's stdout. Therefore, Curly generates SIGPIPE and exits.
- **Question 9:** It reads EOF from stdin.
- **Question 10:** Once.