IC221 Project 2: IPC Pipes

AY2022 Spring

Learning Objectives

Unroll a shell command pipeline

Set process group IDs

Inter-process communication using IPC pipes

Submission

Submit both unroll.c and pipe.c

Test Script: ./test.sh (you can comment out individual tests while working on different sections. Open up the test script and place comments at the bottom where appropriate.)

Pipeline Unrolling

Consider the following pipelined bash command:

sleep 10 | sleep 20 | sleep 30 | sleep 50

The entire pipeline runs as a single “job” in the shell, and each part of the pipeline is a *separate process running in parallel within a process group*.

In this section, you will implement “pipeline unrolling,” which is the process of creating a command shell pipeline based on the | symbol and forking each of the child processes into a process group. This “unrolling” step will only concern itself with the individual process forks, and not any inter-process communication (IPC).

If working in WSL, install the following dependency: sudo apt install libreadline-dev

**Task 1: Unroll** (40 points) unroll.c

Requirements:

* fork() each part of the pipeline as its own process with proper arguments.
* Ensure all processes of the pipeline end up in one *process group*, separate from unroll.
* Your program must be able to execute a set of sleep commands, at a minimum, but should be written generally enough to execute other commands.

Example Output. You can use the command line tool time to see how long a pipeline *should* take; notice that the pipeline, the first argument to unroll, is in quotes:

$ time ./unroll "sleep 1"

real 0m1.010s

user 0m0.003s

sys 0m0.005s

$ time ./unroll "sleep 1 | sleep 3"

real 0m3.011s

user 0m0.004s

sys 0m0.006s

$ time ./unroll "sleep 1 | BAD | sleep 2"

exec: No such file or directory

real 0m2.012s

user 0m0.004s

sys 0m0.008s

$ time ./unroll "head -c 5 /dev/zero"

real 0m0.013s

user 0m0.002s

sys 0m0.005s

$ ./unroll "sleep 200 | sleep 300 | sleep 400" &

[1] 17012

$ ps -o pid,pgid,args

PID PGID ARGS

17012 17012 ./unroll sleep 200

17016 17016 sleep 200

17017 17016 sleep 300

17018 17016 sleep 400

(...)

**PART 2: Pipe** (60 points) pipe.c

In this section, you will focus on the inter-process communication of the processes in a command shell pipeline by unrolling different length pipelines of commands in a program called pipe.

To complete this task, you will need to think about how to unroll a pipeline such that there exists an IPC pipe between each of the processes. This requires a sequence of calls to pipe() to and dup2() for duplicating the write/read end of the pipes to stdin/stdout of each of the processes appropriately.

The fork/join/wait portion of part 2, however, is just like it was with unroll, in part 1. The only difference is that now, the child processes have to have their respective outputs and inputs connected to each other.

Pipe Widowing. One key to properly setting up a pipe between processes is “widowing” the pipe. This is where one end of the pipe is closed (normally the end that is not in use by that process). Here are the key points:

- Pipes created before a fork are shared by the parent and child processes -- each gets its own copy.

- All instances of the pipe must be used or widowed, including the copy that exists in the parent. *If the pipe file descriptors aren’t either used then closed or widowed, the entire pipeline may hang*.

Unrolling Arbitrary Length Pipelines

Your program needs to be able to handle arbitrary length pipelines. There are three main cases for arbitrary length pipelines:

* The process is at the *start* of the pipeline: You should not duplicate the stdin (it receives regular input), but should duplicate stdout to the outgoing pipe.
* The process is in the *middle* of the pipeline: You should duplicate stdin to the incoming pipeline, and duplicate stdout to the outgoing pipeline.
* The process is at the *end* of the pipeline: You should duplicate stdin to the input pipeline, but not duplicate stdout (it produces regular output).

The following diagram illustrates this process. A given pipe will be referred to as cur\_pipe by the process to its left, but the same pipe will be called last\_pipe by the porocess to its right.

last\_pipe last\_pipe

proc\_0 proc\_1 proc\_2 proc\_n

.--------. .--------. .--------. .--------.

| | .---| | .---| | .--- ... | |

| stdin | .\_/ .\_\_| stdin | .\_/ .\_\_| stdin | .\_/ .\_\_ | stdin |

| |/ .\_/ | |/ .\_/ | |/ .\_/ | |

| stdout |\_/ | stdout |\_/ | stdout |\_/ | stdout |

'--------' '--------' '--------' '--------.

cur\_pipe cur\_pipe cur\_pipe

In other words, the labels cur\_pipe and last\_pipe are *relative to each process*. For proc\_1, the cur\_pipe is the same as proc\_2's last\_pipe. This is called “shifting the pipes.” In the parent, after forking a child, you'll need to advance the pipe indices before creating a new next pipe. For example, something like:

last\_pipe[0] = cur\_pipe[0];

last\_pipe[1] = cur\_pipe[1];

Complete pipe.c. Requirements:

* It must use the calls pipe() and dup2() to set up inter-process communication between the child processes.
* All processes must run in parallel.
* The first process in the pipeline must leave the stdin file descriptor unaltered, and the last process in the pipeline must leave the stdout file descriptor unaltered.

There are six example pipelines in the Makefile that can be used for testing. You can test each by entering make pipe<number> as in:

make pipe1 will compile and execute with the following pipeline:

cat sample-db.csv | cut -d , -f 8 | sort | uniq | wc -l