IC221: Systems Programming
12-Week Practicum

April 4, 2014

Instructions:

- This is a 1 hour and 50 minute practicum.
- The practicum is open notes, open book, and open internet. Previous lab submissions count as notes. You may not, however, communicate with anyone directly, e.g., via intstant message.
- Complete all the programs as described in this document and in the source code provided.
- To retrieve the source, run `~aviv/bin/ic221-up`, then `cd ~/ic221/practicum/12-week`
- All code submissions will occur via the `~aviv/bin/ic221-submit` with the option `practicum/12-week`. You can submit multiple times, only your final submission will be graded. **DO NOT FORGET TO SUBMIT.**
- **WRITE YOUR NAME AND ALPHA IN THE README FILE** where you can also include any notes or comments.
- You must also turn in this document at the completion of the examination. Feel free to include any notes or comments in writing within this document.
- You can **test your submission** by running the test script `test.sh` from the `~/ic221/practicum/12-week` directory. The test script will report pass or fail for each grading unit.

Name: _____

Section: _____

Alpha: _____

Question	Points	Score
1	5	
2	10	
3	10	
Total:	25	

1. [**5 points**] Change into the `ic221/practicum/12-week/prob1` directory, where you will find the following files:

   - `mathlib.h` : Header file for mathlib
   - `mathlib.c` : Source file for mathlib
   - `rpn.c` : Source code for the RPN calculator
   - `Makefile` : A makefile

   Your goal is to complete the Makefile to properly compile the RPN calculator program, `rpn`, from the source `rpn.c`

   For reference: an RPN calculator, which stands for *Reverse Polish Notation* is a post-fix calculator. As opposed to in-fix calculators where the operator appears between the operands, like the + operator between the operands in 1 + 1, in post-fix calculators, the operator appears *after* the operands, like 1 1 +. Here is a sample run of the calculator you should expect after compilation:

```
#> ./rpn
(Ctrl-D to EXIT))
RPN> 1 1 +
0: 2
(Ctrl-D to EXIT))
RPN> 1 2 *
0: 2
(Ctrl-D to EXIT))
RPN> 3 !
0: 6
(Ctrl-D to EXIT))
RPN> 6 2 /
0: 3
(Ctrl-D to EXIT))
RPN> 4 5 -
0: -1
```

   Your `Makefile` must compile all source files to object files first (ending in `.o`) and then assemble object files into a single executable, `rpn`. Additionally, you must have a `clean` target that will remove any object files or executables, but not source or header files. The following targets must apear:

   - `rpn` : target for assembling the executable from object files
   - `rpn.o` : target for compiling the `rpn.c` file to an object file
   - `mathlib.o` : target for compiling the `mathlib.c` file to an object file
   - `clean` : target for cleaning the directory of object files and executables.

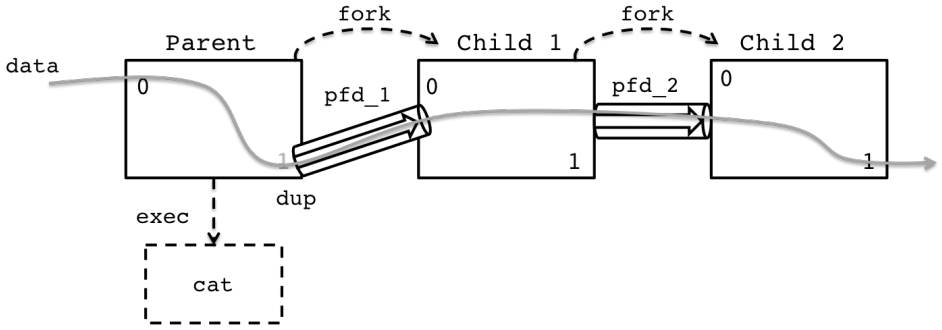
   Note that the `rpn` program also requires the `readline` library which must be indicated during final assembly of the `rpn` executable with the flag `-lreadline`.

   For this problem, you **may not** use the `make` constructs `CC` or `CFLAGS` nor may you rely on the default compilation settings. All compilation steps must be explicitly provided for each target.

2. **[10 points]** Change into the `ic221/practicum/12-week/prob2` directory, where you will find the following files:

   - `grandchild.c` : Source file for `grandchild` program, which you will complete
   - `Makefile` : A Makefile to compile `grandchild.c`
   - `*.txt` : test files

   Your goal is to complete `grandchild.c` so that the pipeline properly functions. As an overview, refer to the following diagram:



   The program will fork twice: first the Parent will fork Child 1 who will then fork Child 2. The parent of Child 2 is Child 1, so Child 2 is the *grandchild* of the Parent process.

   There are also two pipes, `pfd_1` and `pfd_2`. The write end of `pfd_1` should be duplicated onto the standard output of the Parent, while the remaining ends of both pipes are used without duplication.

   The Parent will eventually `exec` the `cat` program which will make use of the duplicated pipe, `pfd_1`.

   The expected result is that data will flow through the Parent's standard input and written to `pfd_1` (via `cat`). Data is then read from `pfd_1` and written to `pfd_2` in Child 1. Finally, data is read from `pfd_2` and written to standard output in Child 2.

   You must add all the calls to `pipe()`, `dup()` and `close()` to complete the program, and your program must properly handle EOF (end-of-file). *HINT:* Don't forget to widow *all* unused end of the pipes.

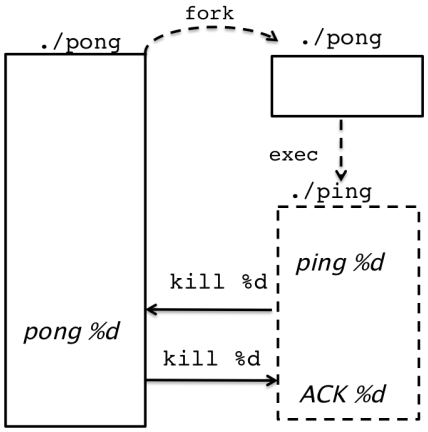
   Here is some sample output:

```
#> ./grandchildren < GoNavy.txt
Beat Army!
#> ./grandchildren < BeatArmy.txt
Go Navy!
#> cat GoNavy.txt | ./grandchildren
Beat Army!
#> ./grandchildren
Hello
Hello
^D
#>
```

3. **[10 points]** Change into the `ic221/practicum/12-week/prob3` directory, where you will find the following files:

   - `ping.c` : Source file for signal pinger
   - `pong.c` : Source file for signal ponger, which you will complete
   - `Makefile` : Source file for mathlib

   Your goal is to complete the `pong.c` program such that it will properly interact with the `ping` program. The `ping` program will choose a random signal number, excluding `SIGKILL`, `SIGSTOP` and `SIGCHLD`, and send that signal to its parent and await the same signal to be sent in reply. You can view this interaction and control flow of `pong` and `ping` like so:



   When `ping` sends the signal, it prints "ping [signum]" where "[signum]" is replaced by the signal number. The `pong` program should print "pong [signum]" upon receiving the signal before sending it back to `ping`. Finally, when `ping` receives the signal, it prints "ACK [signum]".

   Here are some sample runs of `pong` to compare against, note that a random signal number is chosen for each run, so your output may differ slightly from mine.

```
#> ./pong
ping 31
pong 31
ACK 31
#> ./pong
ping 6
pong 6
ACK 6
#> ./pong
ping 13
pong 13
ACK 13
#> ./pong
ping 13
pong 13
ACK 13
```

   DO NOT run the `ping` program directly. It will most likely terminate your shell. You should only run the `pong` program.