IC221: Systems Programming
12-Week Practicum

2 Apr 2015

Instructions:

- This is a 1 hour 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 instant message, stack overflow, or etc.
- There are four practicum problems.
- All problems are graded pass/fail. No partial credit.
- 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.
- 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`

Name: ______________________________

Alpha: ______________________________

1. This problem refers to the starter code in the `makefile` directory. Your task is to complete the makefile such that the program `echoshell` properly compiles by just typing `make.`

   Note that the `echoshell` requires the `readline` library, so be sure to include the library in your last step of compilation.

   ```
   clang -lreadline  ________________  -o echoshell
   ```

   where underscores are replaced with the right object files. You can determine the dependencies for the compilation by observing the header files includes in the sources files.

2. This problem refers to the starter code in the `open` directory. Your task is to complete the `read_account.c` program which given an account file specified on the command line, will parse and print out the account information. For example:

   ```
   aviv@saddleback: open $ ./read_account aviv.acc
   Adam Aviv        1101012010304204
   --------------------------------
   bal:$1020399.20  cred:$500.19    deb:$30276.87

   aviv@saddleback: open $ ./read_account pepin.acc
   Joni Pepin       11010120528374119
   --------------------------------
   bal:$412877.41   cred:$609.31    deb:$79.01

   aviv@saddleback: open $ ./read_account aviv.acc pepin.acc
   Adam Aviv        1101012010304204
   --------------------------------
   bal:$1020399.20  cred:$500.19    deb:$30276.87

   Joni Pepin       11010120528374119
   --------------------------------
   bal:$412877.41   cred:$609.31    deb:$79.01
   ```

   The accounts is stored in the structure `account_t` and was written to the files using the following `write()` statement.

   ```
   write(fd,&acc,sizeof(account_t));
   ```

   Your must open the files and read the account information back to structure.

3. This problem refers to the starter code in the `pipes` directory. Your task is to complete the `reversing.c` program which will reverse input on `stdin` line-by-line. For example:

```
aviv@saddleback: pipes $ cat HelloWorld.txt
Hello
World
aviv@saddleback: pipes $ cat HelloWorld.txt | ./reversing
World
Hello
aviv@saddleback: pipes $ cat GoNavyBeatArmy.txt
Go
Navy
Beat
Army
aviv@saddleback: pipes $ cat GoNavyBeatArmy.txt | ./reversing
Army
Beat
Navy
Go
```

This is accomplished with pipes to a forked child process executing `tac`. The pipes used are described below:

```
int parent_to_child[2];   //pipe from the parent to child
int child_to_parent[2];   //pipe from child to parent

pipe(parent_to_child);
pipe(child_to_parent);
```

Data from the parent's standard input will be written to the pipe to the child (which executes `tac`), and the output of the child will be written to the pipe to the parent (which is the output of `tac`). Finally, the parent will print out anything read from the child back to `stdout`.

Your primary task is to properly set up the pipes using calls to `close()` and `dup2()`. Don't forget to widow!

4. This problem refers to the starter code in the `signals` directory. Your task is to properly signal `signal-me` program and decrypt its output by completing the `get-secret` program. First the `signal-me` program will respond with a secret, encrypted message if it receives the right signal. First, you must determine which signal will result in the secreted, encrypted message.

Next, once you've determine the right signal, an encrypted message in raw bytes will be printed to the terminal. You must decrypt that message by flipping all the bits. To flip the bits of a byte, we use the XOR function, for example:

```
unsigned char c;

//...

c = c ^ 0xff ; //flip bits!
```

will flips the bits of `c` from 1's to 0's and 0's to 1's. `get-secret` will read one byte at time from `stdin`, flip the bits, and write the decrypted byte to `stdout` to reveal the secret message.

To receive credit, save the secret message in a file called `secret` like so when `signal-me` is properly signaled:

```
./signal-me | ./get-secret > secret
```