Recitation 6
02-29-16
Today

• Project 2
• I/O Redirection
• Pipes
Project 2

- Three types of processes:
  - Server process
  - Server Shell process
  - User shell process
Server process

- Function: `main()`
- Forks server shell process
- Reads commands from server shell and executes them (main cmd : \add user)
- Non-blocking read to read from pipes
- Ref: Section 4.1.1
..[contd.]

• \texttt{add user} – server forks an xterm with user shell running inside it
• Server reads commands from user shell as well
• Processes command and writes back to appropriate pipes
• Ref: Section 4.1.2
Server Shell process

• Created by Server on startup
• Acts as administrator's interface; starts a prompt
• New users can be added using this shell
• Forks child to read messages from server; prints out those messages
• Ref: Section 4.2
User shell process

- Created by Server on `\add user` command
- Same C program as server shell, only running inside new xterm window
- So works exactly like the server shell
- Ref: Section 4.2
Suggestions

- Follow suggested steps; Section 7.2
- Execl: use execl to send pipe fds to the exec'ed shell program as arguments -
  ```c
  execl(XTERM_PATH, XTERM, "+hold", "-e", path, fd1, fd2, username,...);
  ```
- Use `fflush(stdout)` if your shell program seems to be buffering printfs
I/O Redirection

• By default, programs read from standard input (what you type into the terminal) and write to standard output (print to the terminal).
• You can override this using I/O redirection.
I/O Redirection

- Redirection methods:
  - `>` redirects output to a file.
  - `<` redirects input, taking it from a file
- Examples:
  - `ls -l > ls_out.txt`
  - `echo Be very very very quiet. I’m hunting rabbits. > temp.txt`
  - `sed s/r/w/g < temp.txt > fudd.txt`
Pipes

• Pipes can be used to chain together multiple programs.
• They automatically feed output of one program into the input of another without needing a temporary file.
• Example:
  - `echo abcdef | sed s/abc/xyz/g`
Pipes

• Pipes can be combined with < and >

• Previous example:
  - `echo Be very very quiet. I’m hunting rabbits. > temp.txt`
  - `sed s/r/w/g < temp.txt > fudd.txt`

• The same can be accomplished with:
  - `echo Be very very quiet. I’m hunting rabbits. | sed s/r/w/g > fudd.txt`
Pipes

- Processes can create a pipe internally:
  - `void pipe(int fd[2])`
- Takes an array of two `ints`, creates a pipe, and places the input and output file descriptors in the array.
  - `fd[0]` is the “read” end of the pipe,
  - `fd[1]` is the “write” end.
Pipes

• Pipes can also be used for interprocess communication.
• After a `fork()`, parent and child both have access to any previously created pipes. Parent can write data to pipe for the child to read, and vice versa.
• Try it out in the pipes exercise. (`copy.c`)
Exercises

• copy.c: practice with pipes.
  – The parent should read in the file and write it to the pipe.
  – The child should read from the pipe and write it to a file.

• Note: Please make sure you complete the leftover/all program as HW. This is important for the Project.
Questions?