Recitation 5
10-13-14
Today

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

• Three types of processes:
  • Router
  • Controller
  • URI_Rendering
Router

- Function: \textit{main()}
- Creates Controller
- Triggers webpage rendering in tab when controller asks
- Non-blocking read to read from pipes
- Ref: Section 4.1
Controller

- Function: `run_controller()`
- Controls tab creation and `uri_rendering`
- Wrapper Lib functions involved:
  - `show_browser()`
  - `new_tab_created_cb()`
  - `uri_entered_cb()`
URL-Rendering

• Function: `run_url_browser()`
• Read and interpret messages from Router
• Wrapper Lib functions:
  – `render_web_page_in_tab()`
  – `process_all_gtk_events()`
  – `process_single_gtk_events()`
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. Will not be graded, but imp. practice for Project.
Questions?