Group 4 Members: Isaac Supeene and Braeden Soetaert

Not a design change but we are using Ruby 1.9.3 for our code development and execution.
Run compile.sh to compile for the lab computers.

Overall:
We changed our designs to not do as much input validation and error handling as it should due to time constraints. We also do not properly handle and respond to signals due to time constraints but our programs should do this. For the shell this could be implemented in the application layer but for the other 2 exercises this would need to be changed in the code base itself.

Shell:
We changed our shell design in a couple ways. Our shell now relies heavily on Ruby's “Shell” class and hands many interactions off to this shell class after pruning the input somewhat. Our decision to use the Shell class was due to the fact that it was already there and did most of the things we wanted to do. Due to this we also changed our shell's name to be Shell421 so that name conflicts did not occur.

Another design change we made was to make our shell just pass commands from the user to the file system via the Shell class. We do not handle keyboard shortcuts or many commands that are not system commands. One example is exit which will not exit the shell. Because of this our shell in production would actually have an application layer over top of it that handles inputs appropriately. This design change separates the responsibility of a shell, which just needs to provide a user with access to system commands, from the inputs. In this way the shell could be reused with different application layers for a different feel. Also this allows us to make a shell that does not contain all the window dressing that modern shells have and this window dressing can be added and changed in the application layer without changing the underlying shell.

A bare bones application layer is implemented in shell_driver.rb where ctrl-D causes the shell to exit.

Timed Printout:
We changed the timed printout module name to be DelayedAction as it can now handle both a delayed print and a delayed block. Our current implementation also has a flaw in that it forks a new process for every delayed action that is requested. We tried to remove this from our code by changing to threads but because we are using a C library, this approach does not work. Ruby threads do not recognize the nanosleep inside our C library as a sleep and they block on the call to C library instead of sleeping. This is due to Ruby threads being “green threads” instead of actual threads.

File Tracker:
We changed our design from using inotify due to the complexity that would ensue from using inotify. Instead we now use a loop that checks for file changes every so often and if a file has been modified in the specified way then an action will be performed after the time delay. One flaw in this approach is that it makes the time delay before the action is taken not as accurate because an additional time delay may be incurred by waiting for the checking loop to run.

Also we changed the order of the arguments for the file tracker functions to be file list and then duration. This was done to allow duration to have a default value of 0 so that actions could be immediately executed by default.

Another change is that we are using imprecise timing in the file tracker before an action is run because

the plan was to use the timer printout but due to the errors we ran into with the threading of the timer input we are just using the general Ruby sleep function because we thought that an imprecise sleep would be better than spawning a new process for each file tracking.