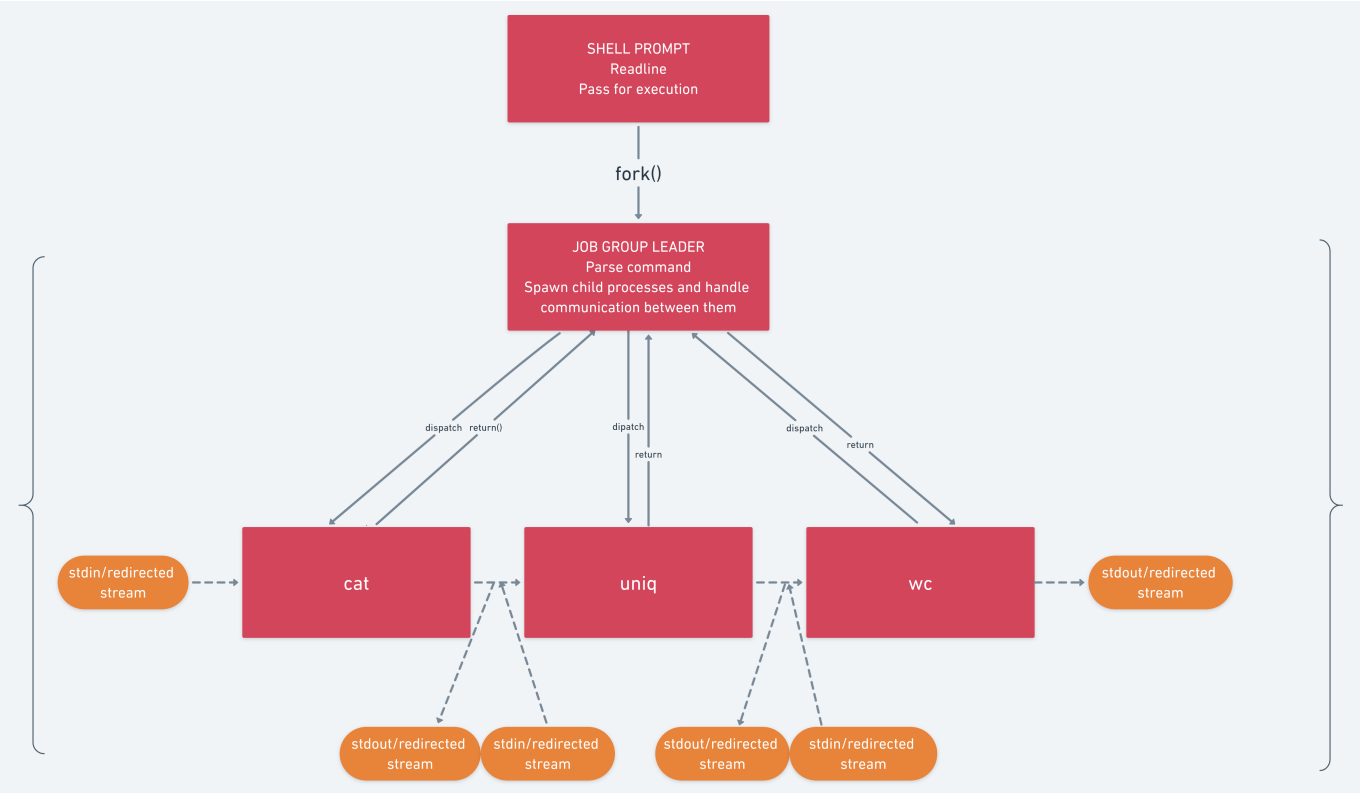
Shell

Submitted to: Dr. Hari Babu
Course: Network Programming IS F462
Assignment 1

Design

This exercise create a bash like shell primitive support for chaining processes via pipes. Functionality for daemonizing a process, redirection and limited job control is also provided.

The following figure gives an illustration of the design we incorporated.



Shell Functionality

The main shell process is a prompting process, which shows the prompt and waits for user to enter the command.

On successfully recieving the command, it passes the command to a newly spawned the child process in a seperate process group. This child process is the leader of the process group, and all of the subprocesses associated with the command will be present in this group for job control. This process acts as the leader for the subprocesses. The prompting parent process checks if & is present at the end or not, and if not, gives the terminal control to this group (to run in the fg). Parent then waits for the leader to finish, or stop executing.

The leader now parses the command and sees the relevant message passing techniques used between different subprocesses. It also sees the redirection operators. For each of the subprocess, it maintains a relevant structure denoting its input fd, and ouput fd.

This leader now starts the sequential execution of the processes by exec() and fork() and waits for each of them to finish. It sets up appropriate fds before executing.

Daemonize Functionality

The daemonize command helps in daemonizing the process by making sure that the child process which will be responsible for exec of the will be immune to signals such as SIGHUP, be devoid of any file descriptors, have its STDIN, STDOUT, STDERR set to /dev/null, and umask set to 0. Also, it will be in its own private session, and won't be a session leader, therefore won't be able to reclaim any terminal device.

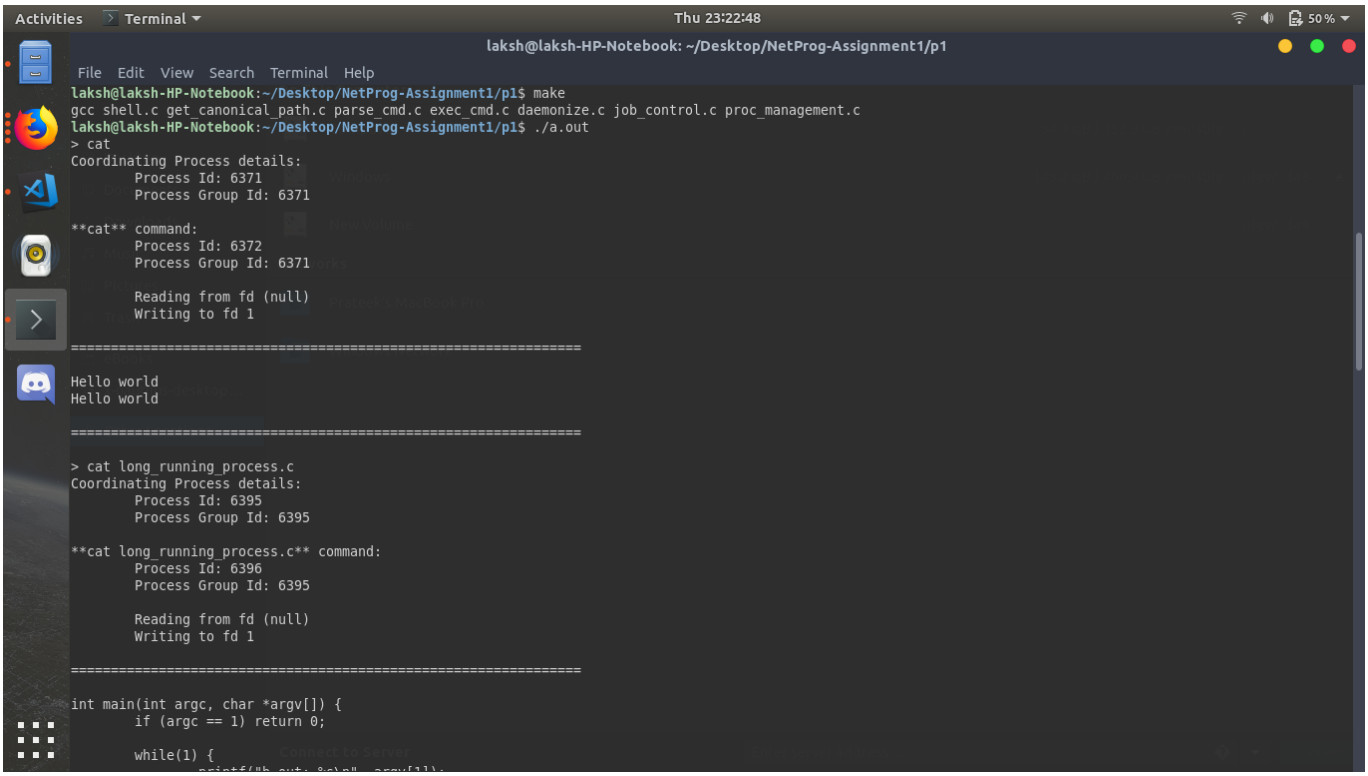
Job control Functionality

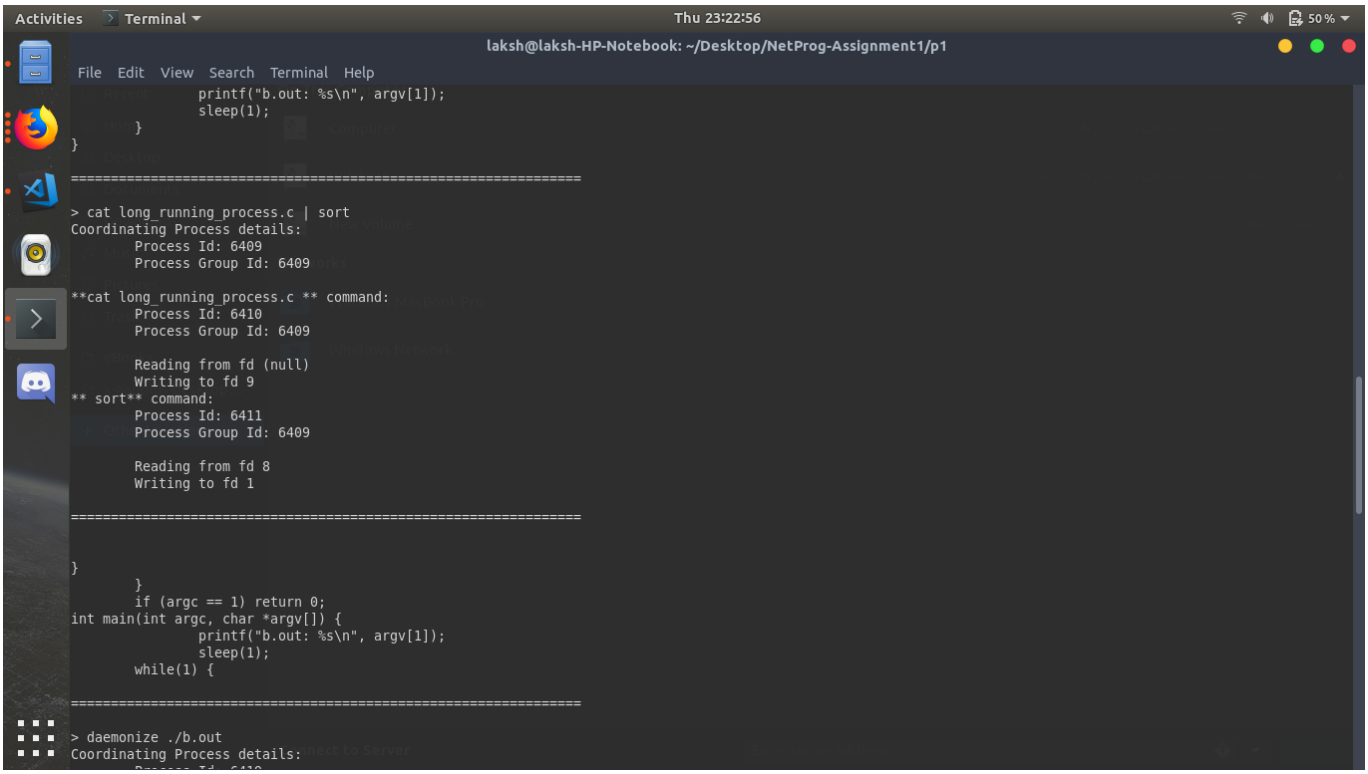
Each job's leader is the process group leader as well. The shell maintains a list of jobs/commands, and responds to the change of state of each of the child. Due to this, it keeps the track of current foreground controlling group, and background groups. Also it maintains which all jobs are in running state and stopped state. `jobs` is used to display the current list of jobs `bg` is used to run a background job in background (without giving it access to the controlling terminal) `fg` is used to run a background job in foreground (by setting it as the terminal controlling process)

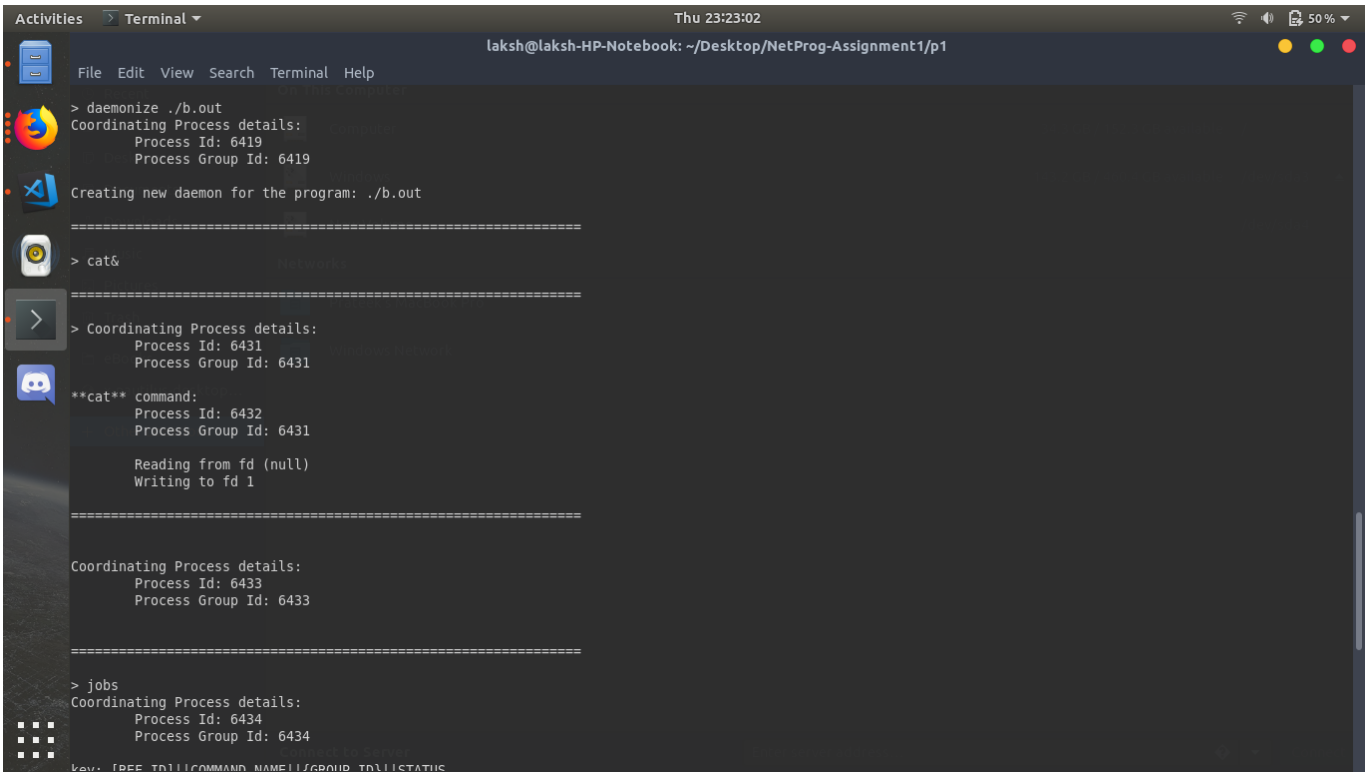
Usage

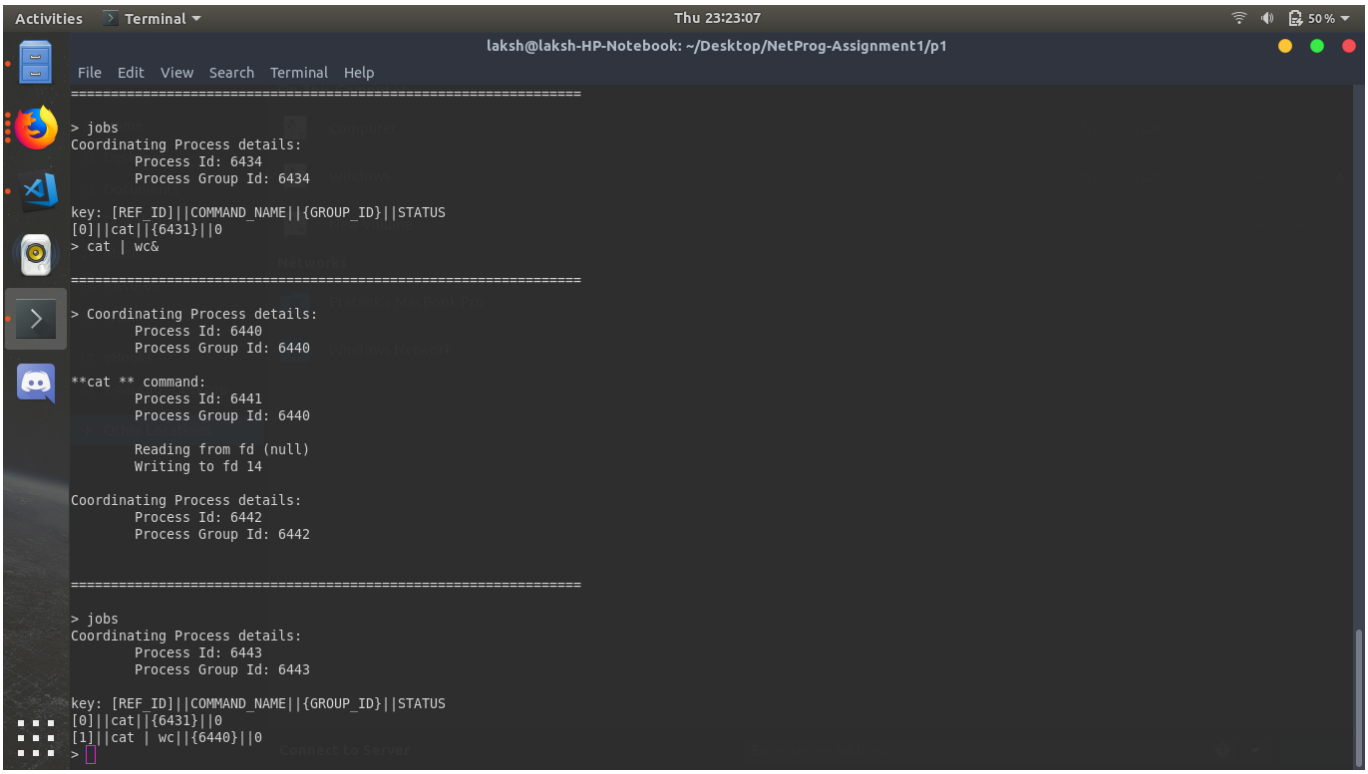
```
make
./a.out
```

Screenshots




Activities
Terminal
Thu 23:22:56
50 %
laksh@laksh-HP-Notebook: ~/Desktop/NetProg-Assignment1/p1
File Edit View Search Terminal Help
printf("b.out: %s\n", argv[1]);
sleep(1);
}
}
==========
> cat long_running_process.c | sort
Coordinating Process details:
Process Id: 6409
Process Group Id: 6409
**cat long_running_process.c ** command:
Process Id: 6410
Process Group Id: 6409
Reading from fd (null)
Writing to fd 9
** sort** command:
Process Id: 6411
Process Group Id: 6409
Reading from fd 8
Writing to fd 1
==========
}
}
if (argc == 1) return 0;
int main(int argc, char *argv[]) {
printf("b.out: %s\n", argv[1]);
sleep(1);
while(1) {
==========
> daemonize ./b.out
Coordinating Process details:


Activities
Terminal
Thu 23:23:02
50 %
laksh@laksh-HP-Notebook: ~/Desktop/NetProg-Assignment1/p1
File Edit View Search Terminal Help
> daemonize ./b.out
Coordinating Process details:
Process Id: 6419
Process Group Id: 6419
Creating new daemon for the program: ./b.out
==========
> cat&
==========
> Coordinating Process details:
Process Id: 6431
Process Group Id: 6431
cat command:
Process Id: 6432
Process Group Id: 6431
Reading from fd (null)
Writing to fd 1
==========
Coordinating Process details:
Process Id: 6433
Process Group Id: 6433
==========
> jobs
Coordinating Process details:
Process Id: 6434
Process Group Id: 6434



Assumptions

- No interactive commands allowed (eg ssh, cat, etc.)
- Shell properties (like environment variables) cannot be changed except the current directory using command `cd`
- No command should have a dot(`.`) in it except when using to identify the name of a node (i.e. `n1.<command>`). For example, relative paths cannot be give for any command because of `./`.
- `./client` should be opened in the home directory of the user logged in n1, else the server will continue to execute in the directory in which the shell is opened.