CSC 360 Summer 2018 Tutorial#1

By: Deepak Kumar
Graduate Student, Computer Science
University of Victoria
dkumar@uvic.ca

Development Environment

Linux Server : linux.csc.uvic.ca

- Your code must run on this server
- Logging In
  - Unix/Linux: **$ ssh [netlink]@linux.csc.uvic.ca**
  - Windows: Putty

Programming: C (gcc version 4.8.4)

Reference programs are at directory **/home/zastre/csc360/a1**

File Transfer

- SCP/PSCP
  - To the server: **$ scp [-r] <local/path/file> netlink>@linux.csc.uvic.ca:remote/path**
  - From the server: **$ scp [-r] <netlink>@linux.csc.uvic.ca:remote/path/file local/save/path**
- FileZilla or WinSCP (Windows only)
- Git

Program

- Editing
  - VIM or Sublime Text
  - Do not use notepad, if required use notepad++
- Compiling
  - gcc [compiler flags] <file.c> -o [executable] [libraries]
  - Example: $ gcc -W -g hello.c -o hello.out
  - Can use Makefile for compilation
- Debugging Tools
  - gdb, valgrind/helgrind

Some Important Linux Commands

- **$ pwd:** display current working directory
- **$ ls:** list content of current directory
- **$ lsof -u <username>:** List all opened file of user
- **$ lsof -p <pid>:** List the files being used by process with id <pid>
- **$ lsof -c <program>:** all devices the program is accessing
- **$ <command> &**: backgrounds the process and return PID which you can use later if you want to kill the program

Some Important Linux Commands

- **$ ps**: List all running processes
- **$ pstree:** List and visualize processes in hierarchical structure
- **$ kill <pid>:** kill programs with <pid>
- **$ pkill <name>:** kill program having <name>
- **$ pgrep <name>:** List processes based on <name>
- **$ top:** display system resources and processes using most of the resources

Process Creation

- Fork() system call creates a child process
- It returns three values
  - ***Negative Value***: creation of a child process was unsuccessful.
  - ***Zero***: Returned to the newly created child process.
  - ***Positive value***: Returned to parent or caller. The value contains process ID of newly created child process.

Process Creation

```c
#include<stdio.h>
#include <sys/types.h>
#include<unistd.h>
void forkexample()
{

int main()
{
    // child process because return value zero
    if (fork()==0)
        printf("I am Child!\n");

    // parent process because return value non-zero.
    else
        printf("I am Parent!\n");

}
```

Output ?

Process Creation

```c
#include<stdio.h>
#include <sys/types.h>
#include<unistd.h>

int main()
{
    // child process because return value zero
    if (fork()==0)
        printf("I am Child!\n");

    // parent process because return value non-zero.
    else
        printf("I am Parent!\n");

}
```

Output

I am Child!

I am Parent!

OR

I am Parent!

I am child!

Stream Redirection and Piping

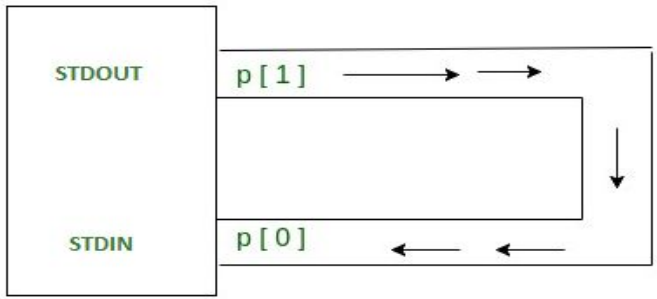
- Used for Interprocess communication
- Redirection
  - Output redirection using “>” eg. ls -al > file.txt
  - Input redirection using “<” eg. sort <file.txt
- Standard I/O Streams
  - Redirection with handles/file descrip
  - Eg. Command 2>error.txt
    Command >file 2>&1

Handle	Name	Description
0	stdin	Standard input
1	stdout	Standard output
2	stderr	Standard error

Stream Redirection and Piping

- Piping
  - Commands can be run together such that one command reads the output from another
  - Syntax : Command1 | Command2 | Command 3 |....|Command n
    Eg. ls -la | sort
  - Can used combined with redirection

- In C we use pipe() to create pipe
  - Prototype is int pipe(int fds[2])
  - F[0] is file descriptor for read end
  - F[1] is file descriptor for write end



Examples

- Redirect STDERR to STDOUT and to file
  - **$ <Command> $ 2>&1 > file.txt**
  - NOTE: this is helpful with some commands where logging information is sent to STDERR
- I don’t care about the output, make it go away
  - **$<Command> 2>&1 /dev/null**
- Compare the output of two programs
  - **$ diff <(cat file1.txt) <(cat file2.txt)**

Assignment 1 is posted and due on 6 June 11:55 pm

All the Best!