Lab 1

Cygwin Installation and Linux Tutorial

Description

This is a setup lab the objective is to setup your laptop for Linux and familiarize yourself with some important Linux commands. In this course we'll be using the Linux (UNIX) operating system (OS) as our reference design. In this lab you'll install the C-compiler for Linux on your windows laptop and run a few Linux commands to familiarize yourself with the OS.

You must also be enrolled in a Lab Group of maximum 2 students to be able to submit your solution for this lab. This lab will be a trial lab, which means that you automatically get the mark if you successfully submit the lab to Bright Space.

If you're using a MAC your already setup, since MAC is a Linux-like OS.

Lab Setup (Windows)

You have two options.

1. First Option: Windows 10 provides a Subsystem for linux, here is a link from microsoft on how to set it up for windows 10: https://docs.microsoft.com/en-us/windows/wsl/install-win10

2. Second Option: You can setup Cygwin. Go to http://preshing.com/20141108/how-to-install-the-latest-gcc-on-windows/ and follow the instructions to install Cygwin
    a. Install Required Cygwin Packages as shown in step2 in the steps above
    b. Make sure you can open a Cygwin terminal (also available in Cygwin.bat in install directory)
    c. Check that you can use gcc (the c++ compiler) in Linux. Type which gcc in the Cygwin terminal, you should get something like:

```
$ which gcc
/usr/bin/gcc
```

3. Third Option: Use Linux server provided by the faculty. Let men know if you need more info.

Part A – Linux Tutorial

Go to https://ryanstutorials.net/linuxtutorial/ . Familiarize yourself with Linux by browsing the following tutorials:

1. https://ryanstutorials.net/linuxtutorial/commandline.php
2. https://ryanstutorials.net/linuxtutorial/navigation.php
3. https://ryanstutorials.net/linuxtutorial/aboutfiles.php

4. https://ryanstutorials.net/linuxtutorial/manual.php
5. https://ryanstutorials.net/linuxtutorial/filemanipulation.php
6. https://ryanstutorials.net/linuxtutorial/permissions.php
7. https://ryanstutorials.net/linuxtutorial/filters.php
8. https://ryanstutorials.net/linuxtutorial/grep.php
9. https://ryanstutorials.net/linuxtutorial/piping.php
10. https://ryanstutorials.net/linuxtutorial/processes.php

Part B – Compile & Run a C program

Objective

Familiarize yourself with the directory structure of your Cygwin installation.

1. Try the command ls / , you should get a response like:

```
$ ls /
bin  cygdrive  Cygwin.bat  Cygwin.ico  Cygwin-Terminal.ico  dev
etc  home  lib  proc  sbin  setup-x86_64.exe  tmp  usr  var
```

2. Try the command in detailed mode: ls –l / , you should get a response like:

```
$ ls -l /
total 1245
drwxr-xr-x+  1 Ayman Ayman               0 Feb  2 07:56 bin
dr-xr-xr-x   1 Ayman Ayman               0 Apr 27 21:23 cygdrive
-rwxr-xr-x   1 Ayman Administrators     59 Feb  1 22:57 Cygwin.bat
-rw-r--r--   1 Ayman Administrators 157097 Feb  1 22:57 Cygwin.ico
-rw-r--r--   1 Ayman Administrators  53342 Feb  1 22:57 Cygwin-
Terminal.ico
drwxr-xr-x+  1 Ayman Ayman               0 Feb  1 22:57 dev
drwxr-xr-x+  1 Ayman Ayman               0 Feb  2 07:56 etc
drwxrwxrwt+  1 Ayman Ayman               0 Feb  1 22:58 home
drwxr-xr-x+  1 Ayman Ayman               0 Feb  2 07:56 lib
dr-xr-xr-x  11 Ayman Ayman               0 Apr 27 21:23 proc
drwxr-xr-x+  1 Ayman Ayman               0 Feb  1 22:56 sbin
-rwxr-xr-x   1 Ayman Ayman          938003 Feb  1 22:53 setup-
x86_64.exe
drwxrwxrwt+  1 Ayman Ayman               0 Apr 27 21:19 tmp
drwxr-xr-x+  1 Ayman Ayman               0 Feb  2 07:56 usr
drwxr-xr-x+  1 Ayman Ayman               0 Feb  1 22:56 var
```

3. Check out windows explorer and browse your Cygwin root directory, you should see the same directory structure as shown above. Your home directory is under /home/<name>
4. Edit a file in text editor and type the following
   #include <stdio.h>

```
int main()
{
  printf ("Hello World! \n");
  return 0;
}
```

5. Save the file in new directory under your home directory, e.g. C_programs/hellowWorld.c
6. Compile the program using gcc hellowWorld.c –o hellowWorld.exe
7. Run the executable:

```
$ ./helloWorld.exe
Hello World!
```

8. Type in this c-program:

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

int main()
{
  pid_t pid;

  printf ("fork \n");
  /* fork a child process */
  pid = fork();
    if (pid < 0) {
      /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
    }
    else if (pid == 0)
    {
      /* child process */
      execlp("/bin/ls","ls",NULL);
    }
    else {
      /* parent process */
      /* parent will wait for the child to complete */
      wait(NULL);
      printf("Child Complete");
    }
  return 0;
}
```

9. Save it as fork.c
10. Compile using gcc fork.c –o fork.exe
11. Run using ./fork.exe
12. Modify the child process by making it loop 5 times, each time it run the ls command and sleeps for 1 sec.
13. Show your progress to the TA
14. Submit your source file to Bright Space (you must be enrolled to a lab group)

Part C

Objective

Study the behaviour of process execution by monitoring the process structures via the Linux "**/proc**" directory and get some experience with command line interface (shell) in Linux, learn some system commands/programs.

Direction

The Linux OS provides a convenient way to explore the system properties, as well as properties and states of the currently running processes. Many of the OS structures are visible as files in the pseudo-file system under the directory /proc

1. Enter command **ls /proc** (any text in this font represents commands to be executed). The content of the /proc directory is listed. You will see many number entries (representing directories containing information about processes, for example directory 23406 contains information about process with PID 23406), as well as some more meaningful entries (for example 'version' and 'cpuinfo').
2. Enter command **cat /proc/version**. This should display the content of the file '/proc/version' that contains information about the OS version.

3. Enter command **cat /proc/cpuinfo** and have a look. You might want to explore the content of other files as well.
4. Enter command **ps**. This will list the processes you have launched. At least two processes will be listed: 'ps' (the one you just launched and which is producing this and a shell process (probably 'bash', but could be different, depending on your settings). Write down the PID of the shell process.
5. Enter command **cat /proc/XXXX/stat**, where XXXX is the PID of your shell process from the previous step. You will see a bunch of numbers containing lots of information about this process. You can learn the meaning of the numbers by entering man proc command (which would show you the manual for proc) and scrolling down. You can see most of this information in human readable form in the file '/proc/XXXX/status'. Try **cat /proc/XXXX/status**
6. We want to explore the state changes of processes, but there is not much interesting activity by the shell process. In order to see more interesting behaviour, we will launch and observe other, specially created programs. Extract the provided file 'Lab1_code.zip' into your working directory on the Linux system. Compile the create a 'bin' directory and compile the 3 provided application fiels: **calcloop.c cploop.c procmon.c** using the following command:
   gcc calcloop.c –o clacloop
   gcc cploop.c –o cploop
   gcc procmon.c –o procmon
   a. **procmon** This is a program which periodically (every second) reads the file '/proc/PID/stat' and extracts and prints the process state and the number of jiffies (1/100 of a second) this process spent in kernel and user mode, respectively. This program is launched with one parameter – the PID of the process it has to monitor. procmon terminates when it cannot open the /proc/PID/stat file – usually because the process PID has terminated.
   b. The program calcloop does the following: First, it goes into a loop (10 iterations) that sleeps for 3 seconds, and then starts another loop that increments a variable 400,000,000 times. The calculation takes approximately 2 secs of real time (this may vary according to load on the system).
   c. The program cploop does the following: Creates a file that is 500,000 bytes long (fromfile). Goes into a loop (10 iterations) that sleeps for 3 seconds, and then copies the fromfile to the tofile. The copy operation takes approximately 2 secs of real time (this may vary according to load on the system). The program uses two system calls to copy bytes one at a time.
7. Try to run calcloop and get its pid and use it as argument for procmon using the following piped command: **./bin/calcloop & echo $! | xargs ./bin/procmon** this piped command will run the calcloop command in the background and then will echo its pid and use it as an argument for procmon. Observe the procmon output about the calcpool process status.
8. Similarly try **./bin/cploop & echo $! | xargs ./bin/procmon**
9. Finally run both concurrently using:
   ./bin/calcloop & echo $! | xargs ./bin/procmon & **./bin/cploop & echo $! | xargs ./bin/procmon** &