01/09/2022

Kamal Gurung

CSCI 340-33452 (Winter)

Prof. Dr. Simina Fluture

**Assignment 2**

**Homework # 2 Unix  
I WILL GRADE THIS HOMEWORK**

**Go over the homework requirements. In a separate file (use a text editor) cut and paste the commands that you typed (including the prompt) and the complete or partial output of these commands. The output of the commands should not contain more than six lines.**

1. **Standard Directories and Files**

**Directory**: contains the names of files and/or sub-directories. Standard directories contain some special files.

**Root Directory (/)**

The root directory is the top of the file system. It is the master cabinet that contains all folders and files.

1. **Get a listing of your root directory. (use, cd and ls –l)**

Answer:

[guka4106@venus /]$ ls -l

total 3716

drwxr-xr-x 2 root root 6 Oct 17 2021 a

lrwxrwxrwx 1 root root 7 Jun 22 2021 bin -> usr/bin

dr-xr-xr-x. 6 root root 4096 Sep 7 08:13 boot

drwxr-xr-x 21 root root 3400 Aug 17 11:13 dev

drwxr-xr-x. 179 root root 12288 Jan 9 00:33 etc

drwxr-xr-x. 24 root root 4096 Jan 1 18:10 home

lrwxrwxrwx 1 root root 7 Jun 22 2021 lib -> usr/lib

**/bin**

The binary directory: contains executable files and most Unix commands.

2. **Go to /bin directory. (use cd /bin)**

Answer:

[guka4106@venus /]$ cd /bin

[guka4106@venus /bin]$

1. **List its contents.**

Answer:

[guka4106@venus /bin]$ ls

'['

a2x

a2x.py

ab

abrt-action-analyze-backtrace

abrt-action-analyze-c

abrt-action-analyze-ccpp-local

abrt-action-analyze-core

abrt-action-analyze-oops

abrt-action-analyze-python

abrt-action-analyze-vmcore

1. **List 6 commands that you recognize.**

Answer:

man, alpine, ls, passwd, ssh, chsh

**/dev**

Device directory.

5. **Get a listing of the device directory. Do you recognize any device?**

Answer:

[guka4106@venus /bin]$ cd /dev

[guka4106@venus /dev]$ ls

autofs log sdb1 tty20 tty46 uinput

block loop-control sdb2 tty21 tty47 urandom

bsg lp0 sdb3 tty22 tty48 usbmon0

bus lp1 sdb4 tty23 tty49 usbmon1

char lp2 sdb5 tty24 tty5 usbmon2

cl lp3 sg0 tty25 tty50 vcs

console mapper sg1 tty26 tty51 vcs1

core mcelog sg2 tty27 tty52 vcs2

**/etc**

Contains commands and files for system administration. Usually a user is not allowed to change these files.

6**. Go to /etc directory.**

Answer:

[guka4106@venus /dev]$ cd /etc

[guka4106@venus /etc]$

7. **Do a long listing; Mention a few files that you have already heard about.**

Answer:

[guka4106@venus /etc]$ ls -l

total 3152

drwxr-xr-x. 3 root root 101 Jun 16 2021 abrt

drwxr-xr-x 3 root root 28 Oct 15 2021 accountsservice

-rw-r--r--. 1 root root 18 Jan 16 2020 adjtime

-rw-r--r--. 1 root root 1529 Nov 29 22:06 aliases

-rw-r--r--. 1 root root 12288 Feb 11 2021 aliases.db

drwxr-xr-x. 3 root root 65 Jul 11 10:27 alsa

8. **What is the most used permission? What does it mean? (read about permissions in Unix Handout)**

Answer:

drwxr-xr-x. 3 root root 101 Jun 16 2021 abrt

d: directory

r: read

w: write

x: execute

-: no access

The first letter "d" is the directory entry the next three letters "rwx" provide permission to owner of file to read and write and execute. The next three letters "r-x" gives permission to group members to read and execute but not write. The last three letters "r-x" provide permission to others to read and execute but not write.

9. **Using cat, check the passwd file or similar; look for yourself in the file.**

Answer:

[guka4106@venus /etc]$ cat passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

halt:x:7:0:halt:/sbin:/sbin/halt

**/lib**

Contains a collection of related files for a given language in a single file called an archive.

**/tmp**

Contains temporary files.

**/etc/passwd**

Contains one line for every user on the system and describes that user.

1. **Determine the absolute pathname for your home directory**

10. Type:

echo $HOME

Answer:

[guka4106@venus /]$ echo $HOME

/home/sp23/340/guka4106

11. Type:

Pwd

Answer:

[guka4106@venus /]$ pwd

/

**C. Shell(s) and Shell Environment variables**

1. **Check your default shell using: echo $SHELL**

Answer:

[guka4106@venus /]$ echo $SHELL

/bin/tcsh

1. **Use the chsh command and find a list of available shells.**

Answer:

[guka4106@venus /]$ chsh -l

/bin/sh

/bin/bash

/usr/bin/sh

/usr/bin/bash

/usr/bin/tmux

/bin/tmux

1. **Change the current shell to a tcsh .**

Answer:

[guka4106@venus /]$ chsh -s /bin/tcsh

Changing shell for guka4106.

chsh: Shell not changed.

1. **Check your new shell. The change will not be listed until the next login.**

Answer:

Last login: Mon Jan 9 18:00:20 2023 from 24.102.79.183

[guka4106@venus ~]$ echo $SHELL

/bin/tcsh

[guka4106@venus ~]$

1. **Use the ps (process status – gives a lists of running processes). What do you observe?**

Answer:

[guka4106@venus ~]$ ps

PID TTY TIME CMD

80063 pts/41 00:00:00 tcsh

80372 pts/41 00:00:00 ps

It shows the screenshots of current processes which has two different ‘pid’80063 and ‘pid’80372. Likewise, it displays the time also.

**Shell Environment variables**

**Bourne, Korn shell C shell**

CDPATH cdpath alias names for directories accessed with cd

ENV path along which Unix looks to find config. files

PS1 prompt shell prompt that appears in the command line

PWD cwd name of current directory

HOME home the name of the user’s home directory when the user logs

TERM type of console terminal being used

1. **Processes**

Check the Unix Handout and go over the section about **Processes -section 17.**

The action of each shell, the mechanism of how it executes commands and programs, how it handles the command and program I/O and how it is programmed, are affected by the settings of certain environment variables.

1. **Learn about the ps command using man (type man ps)**

Answer:

[guka4106@venus ~]$ man ps

PS(1) User Commands PS(1)

**NAME**

ps - report a snapshot of the current processes.

**SYNOPSIS**

**ps** [options]

**DESCRIPTION**

**ps** displays information about a selection of the active processes. If

you want a repetitive update of the selection and the displayed

information, use top(1) instead.

1. **Give a list of possible states together with their significance. Identify your login shell.**

Answer:

PID TTY TIME CMD

6943 tty1 01:53:05 Xorg

8113 tty1 00:00:00 dbus-run-sessio

8117 tty1 00:00:00 dbus-daemon

8124 tty1 00:00:42 gnome-session-b

8301 tty1 00:00:00 at-spi-bus-laun

8319 tty1 00:00:00 dbus-daemon

1. Type ps –l and explain the significance of: F, S, UID, PID, PPID, C, PRI, NI, ADDR, SZ, WCHAN, TTY, TIME, CMD fields.

Answer:

[guka4106@venus /]$ ps -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

0 S 2032 80063 80056 0 80 0 - 3767 - pts/41 00:00:00 tcsh

0 R 2032 96132 80063 0 80 0 - 11377 - pts/41 00:00:00 ps

F - flags associated with the process S - process status code (S = interruptable sleep (waiting for an event to complete) ,

R = running on runnable (on run queue) ) UID - effective user ID PID - process ID PPID - ID of parent process

C - processor utlization. (interger value of percent usage over the lifetime of the process)

PRI - priority of process (highest number means lower priority)

NI - nice value. (ranges from 19 to -20) ADDR - memory address of the process SZ - Virtual memory usage

WCHAN - address of the kernel function where the process is sleeping ( '-' if process is running or '\*' if process is multithreaded and ps is not displaying threads.)

TTY - terminal associated with the process

TIME - cumulated CPU time in [DD-]hh:mm:ss format CMD - executable name All of the variables provide specific information about the process.

1. **Use the top command to monitor the CPU activity in real time. It displays the status of the first 15 of the most CPU-intensive task on the system as well as the CPU activity. To stop the execution of top enter .**

Answer:

[guka4106@venus /]$ top

top - 19:24:41 up 145 days, 9:11, 10 users, load average: 2.02, 2.03, 2.00

Tasks: **1117** total, **3** running, **1113** sleeping, **1** stopped, **0** zombie

%Cpu(s): **1.3** us, **1.8** sy, **0.0** ni, **96.9** id, **0.0** wa, **0.0** hi, **0.0** si, **0.0** s

MiB Mem : **515176.3** total, **191081.1** free, **13195.8** used, **310899.4** buff/cache

MiB Swap: **4096.0** total, **4067.1** free, **28.9** used. **495082.4** avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

**916954 gadi8984 20 0 8640 1164 1052 R 99.7 0.0 45229:00 out.exe**

**1472830 shmu7349 20 0 35816 8024 5220 R 99.3 0.0 71957:43 vim**

6520 root 20 0 573044 271640 8008 S 3.6 0.1 2820:18 accoun+

8568 gdm 20 0 13.1g 7.8g 117116 S 3.6 1.6 1039:02 gnome-+

**97570 guka4106 20 0 55444 5444 3736 R 0.7 0.0 0:00.36 top**

6525 root 20 0 83612 9736 6632 S 0.3 0.0 8:44.73 system+

6943 gdm 20 0 2553056 74480 58016 S 0.3 0.0 113:05.89 Xorg

94753 root 20 0 136764 10204 8908 S 0.3 0.0 0:00.03 sshd

2083605 root 20 0 110336 18216 6280 S 0.3 0.0 17:42.78 pmdapr+

2083608 root 20 0 101824 10212 6580 S 0.3 0.0 15:59.01 pmdali+

3774459 apache 20 0 2575080 31428 10056 S 0.3 0.0 0:11.15 httpd

1 root 20 0 240756 13464 8364 S 0.0 0.0 33:47.62 systemd

2 root 20 0 0 0 0 S 0.0 0.0 0:22.36 kthrea+

3 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 rcu\_gp

4 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 rcu\_pa+

6 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworke+

10 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 mm\_per+

1. **Give the total number of tasks, number of running processes, sleeping processes, stopped processes and zombies.**

Answer:

Tasks: **1123** total, **3** running, **1119** sleeping, **1** stopped, **0** zombie

1. **Identify the shell process. Use the “regular” kill command to terminate the shell.**

Answer:

[guka4106@venus ~]$ ps

PID TTY TIME CMD

80063 pts/41 00:00:00 tcsh

108184 pts/41 00:00:00 ps

The current pid is 80063 in the current shell.

[guka4106@venus ~]$ kill 80063

[guka4106@venus ~]$

1. **Use the “sure kill” command to terminate the shell. Explain.**

Answer:

[guka4106@venus ~]$ kill -9 80063

Connection to mars.cs.qc.cuny.edu closed.

‘Regular kill’ means that it is not fully terminated.

‘sure kill’ is the process that terminates the process.

‘kill -9 80063 terminates shell. It also closed the connection with mars.cs.qc.cuny.edu

**PART E 1.**

1. Use Internet sources and give an overview of the command that is used in **Windows for creating a process.**

Answer:

The CreateProcess function works by creating new process. This function runs independently of the process created. If CreateProcess is done sucessfully, PROCESS\_INFORMATION is returned which contains handles and identifiers to be used in the new process and its thread. CreateProcess leads to creation of a process having a one thread. It is important to state the name of a program file which is executable in order to facilitate CreateProcess call. Ten parameters are involved in the CreateProcess which plays a vital role in supporting functions.

1. In a Unix environment, execute parent.c, child.c and orphan.c as follows: Note: upload first the 3 files in your venus/mars home directory.

child and parent:

- compile the child and parent:

gcc parent.c –o parent

gcc child.c –o child

- run the parent in the current directory (the parent after the fork will call the child) (5 points) Don’t worry about warning messages.

./parent

orphan:

- compile and run the orphan: gcc orphan.c –o orphan ./orphan

Answer:

[guka4106@venus ~]$ gcc parent.c -o parent

**parent.c:** In function ‘**main**’:

**parent.c:5:6:** **warning:** implicit declaration of function ‘**fork**’ [**-Wimplicit-function-declaration**]

if (**fork**() ==0){ /\*This is the child process\*/

[guka4106@venus ~]$ gcc child.c -o child

**child.c:** In function ‘**main**’:

**child.c:4:2:** **warning:** implicit declaration of function ‘**printf**’ [**-Wimplicit-function-declaration**]

**printf**("Process[%d]: child in execution ...\n",getpid());

**^~~~~~**

[guka4106@venus ~]$ ./parent

Process[123785]: Parent in execution ...

Process[123786]: child in execution ...

Process[123786]: child terminating ...

Process[123785]: Parent detects terminating child

Process[123785]: Parent terminating ...

[guka4106@venus ~]$

[guka4106@venus ~]$ gcc orphan.c -o orphan

**orphan.c:4:1:** **warning:** return type defaults to ‘**int**’ [**-Wimplicit-int**]

**main**()

**^~~~**

[guka4106@venus ~]$ ./orphan

I'm the original process with PID 123807 and PPID 121343.

I'm the parent process with PID 123807 and PPID 121343.

my child's PID 123808

PID 123807 terminates.

[guka4106@venus ~]$ I'm the child process with PID 123808 and PPID 1.

PID 123808 terminates.

Observe and understand the programs’ execution output. (15 points)

**Extensively comment the output of the programs by relating the theory discussed in class, the meaning of the covered commands and the program listings.**

Answer:

./parent executing process

In mars, while running the parent.c file as ./parent. The child process is created as the parent process uses fork(). when child.c is compiled as gcc child.c -o child in mars account, it is saved as child. the created child process share everything from parents but in execve("child", NULL, NULL) call, the execve() function runs the “child” code that has different code from parent thatv runs a different code in “child”.

In if fork != 0), the parent process runs and terminates first and the parent sleeps for 5 seconds and waits for a child. the child sleeps for 1 seconds and it prints child terminating. When the child terminates it sends signal to parent using wait() command as child has terminated. The parents detects the termination signal from child, the parent prints its output and it finishes its execution.

./orphan executing process

Likewise, in mars “orphan” is created after orphan.c is compiled as gcc orphan.c -o orphan.

When ./orphan is run it prints out pid(123807) , ppid(121343) from using getpid() and getppid().

In the next step, fork() creates a new child process.

Likewise, If (pid!=0), it is parent process that has pid (123807) and ppid(121343). Now, child pid (123807) is printed.

If pid == 0, it is a child process, then the child sleeps for 5 seconds the parent process finishes its execution and finishes before the child process. When a parent finishes execution and terminates.

As the child has sleeping time of 5 seconds, it prints pid (123807) and it's parent's ppid (1). The ppid is 1 since the original parent pid is 123807 which was terminated. The child is adopted by init process that has new parent with pid 1 and the child also terminates.

In case of if fork() <0, then fork fails.

1. **Write a very simple program that will show the possibility of having zombie processes.**

**Write a program named zombie.c (5 points)**

Answer:

In the beginning, the fork() creates a process that is saved in pid.

If the pid is less than zero, it prints fork failed and exits.

If pid equals 0, it is child process. Printf prints “I am the child with pid …… and my parent has ppid ……..”. the child sleeps for 1 second and printf prints “exitting child”

Likewise, if pid is greater than 0, it is parent process and it prints “I am the parent and my id is ……”. The parent sleeps for 25 seconds. child runs and finishes it’s execution before parent. The parent has no wait() command so, we can use the top command for the zombie before the parent process terminates.

In case, if the parent process terminates and child is still running, it is a zombie process. But it is adopted by init process.

Table

Description automatically generated

**You need to submit your homework solution on Blackboard YourLastname\_H2 Your submission should contain three files each of them must have your fullname part of the name:**

**1. doc/txt file that cover parts A to E**

**2. fullname\_zombie.c**

**3. snapshot of top command that shows the zombie process**