Nyeinchan Kyaw

CS-340 (Homework Assignment #2 UNIX)

03/06/2012

Standard Directories and Files

Root Directory(/)

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

```
   SSH Secure Shell 3.2.9 (Build 283)
   Last login: Thu Feb 16 19:50:31 2012 from bsc.qc.cuny.edu
   Welcome to Computer Science !
   [kyny1670@venus ~]$ ls -l
   total 8
   -rw------- 1 kyny1670 underg 1015 May 20  2011 dead.letter
   drwx------ 2 kyny1670 underg 4096 May 30  2011 mail
   [kyny1670@venus ~]$ cd mail
   [kyny1670@venus mail]$ ls -l
   total 96
  -rw-r--r-- 1 kyny1670 underg  1358 Mar  2  2011 8queenscross.cpp
  -rw-r--r-- 1 kyny1670 underg  1212 Apr  5  2011 EightQueenBruteOneD.cpp
  -rw-r--r-- 1 kyny1670 underg  1932 Apr  5  2011 EightQueenNXN.cpp
  -rw-r--r-- 1 kyny1670 underg  1224 Apr  5  2011 EightQueenOneDWithoutGoto.cpp
  -rw-r--r-- 1 kyny1670 underg   702 Feb 28  2011 EightQueenProblem1dimensionalGoto.cpp
  -rw-r--r-- 1 kyny1670 underg  2192 Feb 28  2011 EightQueenProblemDump.cpp
  -rw-r--r-- 1 kyny1670 underg  1117 Feb 28  2011 EightQueenProblemGotoBT.cpp
  -rw-r--r-- 1 kyny1670 underg  3078 May 19  2011 fancy.cpp
 -rw-r--r-- 1 kyny1670 underg   713 May 29  2011 IntegerationF.cpp
 -rw-r--r-- 1 kyny1670 underg  2011 May 20  2011 rat.cpp
 -rw------- 1 kyny1670 underg   508 Feb 28  2011 saved-messages
 -rw------- 1 kyny1670 underg  9960 Apr  5  2011 sent-mail
```

```
  -rw------- 1 kyny1670 underg 16928 Feb 28  2011 sent-mail-feb-2011

  -rw------- 1 kyny1670 underg  3896 Mar  2  2011 sent-mail-mar-2011

  -rw-r--r-- 1 kyny1670 underg  1912 May 20  2011 shortestPathTopDown.cpp

  -rw-r--r-- 1 kyny1670 underg  2015 May 20  2011 stableMarriage.cpp

  -rw-r--r-- 1 kyny1670 underg   608 May 20  2011 TowerOFHanoiR.cpp

  -rw-r--r-- 1 kyny1670 underg  1051 May 20  2011 TowerOfHanoiS.cpp
```

/bin

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

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

```
    [kyny1670@venus dev]$ cd /bin
```

3. List its contents.

```
    [kyny1670@venus bin]$ ls -l

    total 8872
```

4. List 6 commands that you recognize.

6 commands that I recognize are cat, ls, cp, mv and rm.

```
   -rwxr-xr-x 1 root root   25216 Jul 21  2011 cat

   -rwxr-xr-x 1 root root   91272 Jul 21  2011 ls

   -rwxr-xr-x 1 root root   70984 Jul 21  2011 cp

   -rwxr-xr-x 1 root root   80488 Jul 21  2011 mv

   -rwxr-xr-x 1 root root   47088 Jul 21  2011 rm
```

/dev

Device directory.

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

```
   [kyny1670@venus /]$ cd /dev

   [kyny1670@venus dev]$ ls -l

   total 0
```

Yes, I recognize these devices : CPU, DISK, AUDIO, RAM.

```
   drwxr-xr-x 4 root root         80 Nov 20 09:53 cpu

   drwxr-xr-x 6 root root        120 Nov 20 09:52 disk

   crw-rw---- 1 root audio  14,    4 Nov 20 09:52 audio
```

```
lrwxrwxrwx 1 root root          4 Nov 20 09:52 ram -> ram1
```

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

6. Go to /etc directory.

```
[kyny1670@venus /]$ cd /etc
```

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

```
[kyny1670@venus etc]$ ls -l

total 4132

drwxr-xr-x  2 root   root       4096 Aug  5  2010 bluetooth

drwxr-xr-x  4 root   root       4096 Aug  5  2010 fonts

-rw-r--r--  1 root   root     137405 Oct  5 00:17 passwd

drwxr-xr-x  3 root   root       4096 Sep 23 16:31 mail

-rw-r--r--  1 root   root       1044 Sep 21  2009 csh.cshrc

-rw-------  1 root   root          6 Aug 23  2010 shutdown.allow
```

8. What is the most used permission? What does it mean?

The most used permission is : -rw-r--r—

This is a permission and its of 10 characters. The first character shows the file type the next 9 are permissions. These can be formed in a group of 3, owner, group, others. It means the owner has permission to read and write, the group has permission to read and the others have permission to read only. That's the common setting for data files that everybody may read, but only the owner may change.

9. Using the cat command, take a look at the profile and login.defs files.

```
[kyny1670@venus etc]$ cat profile

# /etc/profile

# System wide environment and startup programs, for login setup

# Functions and aliases go in /etc/bashrc

pathmunge () {

    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then

       if [ "$2" = "after" ] ; then

            PATH=$PATH:$1
            else
              PATH=$1:$PATH
            fi
        fi
}

# ksh workaround
if [ -z "$EUID" -a -x /usr/bin/id ]; then
        EUID=`id -u`
        UID=`id -ru`
fi

# Path manipulation
if [ "$EUID" = "0" ]; then
        pathmunge /sbin
        pathmunge /usr/sbin
        pathmunge /usr/local/sbin
fi

# No core files by default
ulimit -S -c 0 > /dev/null 2>&1

if [ -x /usr/bin/id ]; then
        USER="`id -un`"
        LOGNAME=$USER
        MAIL="/var/spool/mail/$USER"
fi

HOSTNAME=`/bin/hostname`
```

```
  HISTSIZE=1000

  if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]; then
     INPUTRC=/etc/inputrc
  fi

  export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC

  for i in /etc/profile.d/*.sh ; do
     if [ -r "$i" ]; then
        if [ "$PS1" ]; then
           . $i
        else
           . $i >/dev/null 2>&1
        fi
     fi
  done

  unset i
  unset pathmunge
 [kyny1670@venus etc]$ cat login.defs
  # *REQUIRED*
  #   Directory where mailboxes reside, _or_ name of file, relative to the
  #   home directory.  If you _do_ define both, MAIL_DIR takes precedence.
  #   QMAIL_DIR is for Qmail
  #
  #QMAIL_DIR      Maildir
  MAIL_DIR        /var/spool/mail
#MAIL_FILE      .mail
```

```
# Password aging controls:
#
#       PASS_MAX_DAYS   Maximum number of days a password may be used.
#       PASS_MIN_DAYS   Minimum number of days allowed between password changes.
#       PASS_MIN_LEN    Minimum acceptable password length.
#       PASS_WARN_AGE   Number of days warning given before a password expires.
#
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_MIN_LEN    5
PASS_WARN_AGE   7

#
# Min/max values for automatic uid selection in useradd
#
UID_MIN                   500
UID_MAX                 60000

#
# Min/max values for automatic gid selection in groupadd
#
GID_MIN                   500
GID_MAX                 60000

#
# If defined, this command is run when removing a user.
# It should remove any at/cron/print jobs etc. owned by
   # the user to be removed (passed as the first argument).
```

```
#
#USERDEL_CMD    /usr/sbin/userdel_local


#
# If useradd should create home directories for users by default
# On RH systems, we do. This option is overridden with the -m flag on
# useradd command line.
#
CREATE_HOME     yes


# The permission mask is initialized to this value. If not specified,
# the permission mask will be initialized to 022.
UMASK           077


# This enables userdel to remove user groups if no members exist.
#
USERGROUPS_ENAB yes


# Use MD5 or DES to encrypt password? Red Hat use MD5 by default.
MD5_CRYPT_ENAB yes


 ENCRYPT_METHOD MD5
```

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

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

```
[kyny1670@venus etc]$ cat passwd
kyny1670:x:3475:800:Nyein Chan Kyaw:/home/sp12/340/kyny1670:/bin/bash
```

/lib

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

```
[kyny1670@venus /]$ cd /lib

[kyny1670@venus lib]$ ls -l

 total 6120
```

`/tmp`
Contains temporary files.

```
[kyny1670@venus /]$ cd /tmp

[kyny1670@venus tmp]$ ls -l

total 360

drwx------ 2 seda2064 underg   4096 Feb 28 10:40 gconfd-seda2064

drwx------ 2 aban3858 underg   4096 Dec 20 18:36 hsperfdata_aban3858

drwx------ 2 andrew   faculty  4096 Dec 15 22:28 hsperfdata_andrew
```

Determine the absolute pathname for your home directory

11. Type:
echo $HOME

```
[kyny1670@venus /]$ echo $HOME

/home/sp12/340/kyny1670
```

12. Type:
pwd

```
[kyny1670@venus /]$ pwd

 /
```

C. Shell(s) and Shell Environment variables

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

```
[kyny1670@venus /]$ echo $SHELL

/bin/bash

[kyny1670@venus /]$
```

2. Use the `chsh` command and find a list of available shells.

```
[kyny1670@venus /]$ chsh -l
```

```
/bin/sh
/bin/bash
/sbin/nologin
/bin/tcsh
/bin/csh
/bin/ksh
/bin/zsh
/usr/bin/ksh
/usr/bin/pdksh
```

3. Change the current shell to a tcsh shell.

```
[kyny1670@venus /]$ chsh -s /bin/tcsh
Changing shell for kyny1670.
Password:
Shell changed.
[kyny1670@venus /]$
[kyny1670@venus /]$
  PID TTY          TIME CMD
 9631 pts/24   00:00:00 bash
11451 pts/24   00:00:00 ps
```

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

Checking new shell by logging in to Venus account again,

```
[kyny1670@venus ~]$ echo $SHELL
/bin/tcsh
```

5. Type ps (process status – gives a lists of running processes). What do you observe?

```
[kyny1670@venus ~]$ ps
  PID TTY          TIME CMD
11555 pts/15   00:00:00 tcsh
11672 pts/15   00:00:00 ps
```

The new process status shows the current shell which is the new shell that I’ve changed.

6. At the shell prompt, type set | more and then press <enter>. What is displayed on your screen?

```
 [kyny1670@venus ~]$ set|more
BASH=/bin/tcsh
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="3" [1]="2" [2]="25" [3]="1" [4]="release" [5]="x86_64-redhat
-linux-gnu")
BASH_VERSION='3.2.25(1)-release'
COLORS=/etc/DIR_COLORS
COLUMNS=80
CVS_RSH=ssh
DIRSTACK=()
EUID=3475
GROUPS=()
G_BROKEN_FILENAMES=1
HISTFILE=/home/sp12/340/kyny1670/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/home/sp12/340/kyny1670
HOSTNAME=venus
HOSTTYPE=x86_64
IFS=$' \t\n'
INPUTRC=/etc/inputrc
LANG=en_US.UTF-8
LESSOPEN='|/usr/bin/lesspipe.sh %s'
```

```
LINES=24
LOGNAME=kyny1670
LS_COLORS='no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:cd=40;33;
01:or=01;05;37;41:mi=01;05;37;41:ex=01;32:*.cmd=01;32:*.exe=01;32:*.com=01;32:*.
btm=01;32:*.bat=01;32:*.sh=01;32:*.csh=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31
:*.taz=01;31:*.lzh=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:
*.bz=01;31:*.tz=01;31:*.rpm=01;31:*.cpio=01;31:*.jpg=01;35:*.gif=01;35:*.bmp=01;
35:*.xbm=01;35:*.xpm=01;35:*.png=01;35:*.tif=01;35:'
MACHTYPE=x86_64-redhat-linux-gnu
MAIL=/var/spool/mail/kyny1670
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/home/faculty/tyler/bin:/hom
e/faculty/tyler/turnin:/home/sp12/340/kyny1670/bin
PIPESTATUS=([0]="127")
PPID=9630
PS1='[\u@\h \W]\$ '
PS2='> '
PS4='+ '
PWD=/home/sp12/340/kyny1670
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:moni
tor
SHLVL=1
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_CLIENT='149.4.115.3 58981 22'
SSH_CONNECTION='149.4.115.3 58981 149.4.211.180 22'
```

```
SSH_TTY=/dev/pts/27
TERM=vt100
UID=3475
USER=kyny1670
_=cwd
consoletype=pty
mpi_selection=
mpi_selector_dir=/var/lib/mpi-selector/data
mpi_selector_homefile=/home/sp12/340/kyny1670/.mpi-selector
mpi_selector_sysfile=/etc/sysconfig/mpi-selector
tmpid=3475
```

7. Identify and list the settings for the variables shown above.

PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/home/faculty/tyler/bin:/home/faculty/tyler/turnin:/home/sp12/340/kyny1670/bin

(PATH shows the whole bin directory of my account)

HOME=/home/sp12/340/kyny1670

(HOME shows the home directory of my venus account)

HOSTNAME=venus

(HOSTNAME is Venus which is venus.cs.qc.edu)

HOSTTYPE=x86_64

(HOSTYPE shows the machine that I installed ssh for my venus account)

PWD=/home/sp12/340/kyny1670

(PWD means the password)

TERM=vt100

D. 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.

```
[kyny1670@venus ~]$ man ps
PS(1)                  Linux Userâs Manual                  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.


     This version of ps accepts several kinds of options:
     1   UNIX options, which may be grouped and must be preceded by a dash.
     2   BSD options, which may be grouped and must not be used with a dash.
     3   GNU long options, which are preceded by two dashes.


     Options of different types may be freely mixed, but conflicts can
     appear. There are some synonymous options, which are functionally
     identical, due to the many standards and ps implementations that this
     ps is compatible with.
     Note that "ps -aux" is distinct from "ps aux". The POSIX and UNIX
```

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

```
 [kyny1670@venus ~]$ ps -l

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

```
 [kyny1670@venus ~]$ echo $SHELL

          /bin/tcsh
```

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

```
[kyny1670@venus ~]$ ps -l

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

0 S  3475 11757  9630  0  75   0 - 16524 wait   pts/27   00:00:00 bash

0 R  3475 12494 11757  0  77   0 - 15884 -      pts/27   00:00:00 ps
```

F   means   extra full format.

S   means sum up some information, such as CPU usage, from dead child processes into their parent. This is useful for examining a system where a parent process repeatedly forks off short-lived children to do work.

UID   is the alias of euid, which means effective user ID.

PID is process ID number of the process.

PPID is parent process ID. This selects the processes with a parent process ID in pidlist. That is, it selects processes that are children of those listed in pidlist.

C means processor utilization. Currently, this is the integer value of the percent usage over the lifetime of the process.

PRI means priority of the process. Higher number means lower priority.

NI means nice value. This ranges from 19 (nicest) to -20 (not nice to others),

SZ means size in physical pages of the core image of the process. This includes text, data, and stack space. Device mappings are currently excluded; this is subject to change.

WCHAN is the name of the kernel function in which the process is sleeping, a "-" if the process is running, or a "*" if the process is multi-threaded and ps is not displaying threads.

TTY means controlling tty (terminal)., same as tname and tt.

TIME is the allias of CPU time which means cumulative CPU time, in "[dd-]hh:mm:ss" format.

CMD is the alias of args, comm, which means command with all its arguments as a string. Modifications to the arguments may be shown. The output in this column may contain spaces. A process marked <defunct> is partly dead, waiting to be fully destroyed by its parent. Sometimes the process args will be unavailable; when this happens, ps will instead print the executable name in brackets. (alias cmd, command). See also the comm format keyword, the -f option, and the c option.

4. 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 <ctrl-C>.

```
[kyny1670@venus ~]$ top

 top - 18:12:29 up 105 days,  8:20, 26 users,  load average: 0.03, 0.06, 0.02

 Tasks: 273 total,   1 running, 272 sleeping,   0 stopped,   0 zombie

 Cpu(s):  0.5%us,  0.2%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st

 Mem:   3967188k total,  3851668k used,   115520k free,   245700k buffers

 Swap:  4104596k total,   323772k used,  3780824k free,  2911020k cached


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

 12589 kyny1670  15   0 12892 1236  820 R  0.7  0.0   0:00.87 top

 12672 pele0345  16   0 95928 2780 2128 S  0.3  0.1   0:00.04 vim

 12954 oracle    16   0 1778m  47m  43m S  0.3  1.2   0:00.19 oracle

     1 root      15   0 10368  624  532 S  0.0  0.0   0:15.74 init

     2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0

     3 root      34  19     0    0    0 S  0.0  0.0   0:00.35 ksoftirqd/0

     4 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0

     5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.96 migration/1

     6 root      34  19     0    0    0 S  0.0  0.0   0:00.43 ksoftirqd/1

     7 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/1

     8 root      10  -5     0    0    0 S  0.0  0.0   0:00.21 events/0

     9 root      10  -5     0    0    0 S  0.0  0.0   0:00.24 events/1
```

```
   10 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
   51 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
   56 root      10  -5     0    0    0 S  0.0  0.0   0:00.53 kblockd/0
   57 root      10  -5     0    0    0 S  0.0  0.0   0:02.40 kblockd/1
   58 root      14  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
```

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

```
Tasks: 273 total,   1 running, 272 sleeping,   0 stopped,   0 zombie
```

6. Do some research and in about 1 page explain the meaning of a zombie process.

A zombie process is a process that has completed execution but still has an entry in the process table. This entry is still needed to allow the parent process to read its child's exit status. In the term's metaphor, the child process has "died" but has not yet been "reaped". Also, unlike normal processes, the kill command has no effect on a zombie process. When a program forks and the child finishes before the parent, the kernel still keeps some of its information about the child in case the parent might need it -- for example, the parent may need to check the child's exit status. To be able to get this information, the parent calls wait(); when this happens, the kernel can discard the information. In the interval between the child terminating and the parent calling wait(), the child is said to be a `zombie'. (If you do `ps', the child will have a `Z' in its status field to indicate this.) Even though it's not running, it's still taking up an entry in the process table. (It consumes no other resources, but some utilities may show bogus figures for e.g. CPU usage; this is because some parts of the process table entry have been overlaid by accounting info to save space.) This is not good, as the process table has a fixed number of entries and it is possible for the system to run out of them. Even if the system doesn't run out, there is a limit on the number of processes each user can run, which is usually smaller than the system's limit. This is one of the reasons why you should always check if fork() failed.

If the parent terminates without calling wait(), the child is `adopted' by init, which handles the work necessary to cleanup after the child. (This is a special system program with process ID 1 -- it's actually the first program to run after the system boots up).

To remove zombies from a system, the SIGCHLD signal can be sent to the parent manually, using the kill command. If the parent process still refuses to reap the zombie, the next step would be to remove the parent process. When a process loses its parent, init becomes its new parent. Init periodically executes the wait system call to reap any zombies with init as parent.

E.

1. Use *man* to find out more about: fork( ), execve( ), wait( ) commands in Unix.

fork( )

```
[kyny1670@venus ~]$ man fork
   Linux Programmerâs Manual                    FORK(2)
NAME
      fork - create a child process
SYNOPSIS
      #include <sys/types.h>
      #include <unistd.h>
      pid_t fork(void);
DESCRIPTION
      fork()  creates  a  child  process that differs from the parent process
      only in its PID and PPID, and in the fact  that  resource  utilizations
      are set to 0.  File locks and pending signals are not inherited.
      Under  Linux,  fork()  is implemented using copy-on-write pages, so the
      only penalty that it incurs is the time and memory required  to  dupli-
      cate  the  parentâs  page tables, and to create a unique task structure
      for the child.
RETURN VALUE
      On success, the PID of the child process is returned  in  the  parentâs
```

execve ( )

```
[kyny1670@venus ~]$ man execve

EXECVE(2)            Linux Programmerâs Manual            EXECVE(2)

NAME

       execve - execute program

SYNOPSIS

       #include <unistd.h>

       int execve(const char *filename, char *const argv[],

                  char *const envp[]);

DESCRIPTION

       execve() executes the program pointed to by filename.  filename must be

       either a binary executable, or a script starting with  a  line  of  the

       form  "#! interpreter [arg]".  In the latter case, the interpreter must

       be a valid pathname for an executable which is  not  itself  a  script,

       which will be invoked as interpreter [arg] filename.

       argv  is  an array of argument strings passed to the new program.  envp

       is an array of strings, conventionally of the form key=value, which are

       passed  as  environment to the new program.  Both argv and envp must be

       terminated by a null pointer.  The argument vector and environment  can

       be  accessed  by the called programâs main function, when it is defined.
```

wait( )

```
[kyny1670@venus ~]$ man wait
```

```
NAME
    bash,  :,  ., [, alias, bg, bind, break, builtin, cd, command, compgen,
    complete, continue, declare, dirs, disown, echo,  enable,  eval,  exec,
    exit,  export,  fc,  fg, getopts, hash, help, history, jobs, kill, let,
    local, logout, popd, printf, pushd, pwd, read, readonly,  return,  set,
    shift,  shopt,  source,  suspend,  test,  times,  trap,  type, typeset,
    ulimit, umask, unalias, unset,  wait  -  bash  built-in  commands,  see
    bash(1)

BASH BUILTIN COMMANDS
    Unless otherwise noted, each builtin command documented in this section
    as accepting options preceded by - accepts -- to signify the end of the
    options.   For  example,  the  :, true, false, and test builtins do not
    accept options.  Also, please note that while executing in non-interac-
    tive  mode  and  while  in  posix mode, any special builtin (like ., :,
    break, continue, eval,  exec,  exit,  export,  readonly,  return,  set,
    shift,  source,  times,  trap,  unset)  exiting  with a non-zero status
    causes the shell to stop execution.
    : [arguments]
        No effect; the command does nothing beyond  expanding  arguments
        and  performing any specified redirections.  A zero exit code is
        returned.
:
PWD=/home/sp12/340/kyny1670
```

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

When it comes to creating a process, Windows Operating System works differently from UNIX. UNIX has fork() to create a process, in the child process, fork() appears to have returned 0 and In the parent process, fork() appears to have returned a non-zero integer. However, Win32 does not have fork(). In Windows, Win32 has two APIs that can be used: 'CreateProcess' and 'CreateThread' to create a new "process" depending on the use of fork and the code base .'Create Process' Windows API call is commonly used. The originating process called 'Create Process which then constructs a new running program image out of whole cloth. Some attributes are "inherited" of course from the creating process (the user ID) but this is all handled by Windows, not really the Process::Create call.

3. In a Unix environment, execute *parent.c*, *child.c* and *orphan.c* as follows:
Note: first you need to upload the 3 files in your venus 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)
Don't worry about warning messages.

```
./parent
```

Orphan:
- compile and run the orphan:

```
gcc orphan.c –o orphan
./orphan
```

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