The Process

Overview

- Process Basics
- Process Status:ps
- System processes
- Mechanism of process creation
- Internal and external command
- Running jobs in background
- Job execution with low priority
- Killing processes with signals
- Job control

Processes

- An instance of a running program.
- Example: grep command, process named “grep” is created.
- If multiple processes, Kernel manages the processes(not the shell).
- Attributes of a process :
  - Process-id (PID): Process is uniquely identified
  - Parent PID(PPID): PID of the parent process.

The shell process

- Shell maintains a set of environment variables. The shell's pathname is stored in shell, but its's PID is stored in?

- In a special variable $$

$ echo $$

- PID of your login shell changes, when logged out.

Parent and children

- Example: cat emp.lst

- Shell becomes the parent and cat becomes the child process.
- Since every process has a parent, we can't have a “orphaned” process .
- Like file, process can have only one parent, and can have one or more children.

- Cat emp.lst | grep 'director'

Wait or Not wait?

- Two different attitudes that can be taken by the parent toward child.
- Wait for the child to die.
- It may not wait for child to die (init)
- Built-in commands of the shell like pwd,cd etc don't create processes.

Process Status: ps

- Command to display some process attributes. The command reads through the kernal's data structures and process tables to fetch the characteristics of processes.

$$- The process number of the current shell. For shell scripts, this is the process ID under which they are executing.

ps options

POSIX option	Significance
-f	Full listing showing the PPID of each process
-e or -A	All processes including user and system processes
-u usr	Processes of user usr only
-a	Processes of all users excluding processes not associated with terminal
-l	A long listing showing memory-related information
-t term	Processes running on terminal term (say, /dev /console)

ps options

$ ps -f

```
student@ugadmin22-IBMThink:~/trial$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
student     3029    3002  0 10:16 pts/1    00:00:00 bash
student     3343    3029  0 11:16 pts/1    00:00:00 ps -f
student@ugadmin22-IBMThink:~/trial$
```

Column	Description
UID	User ID that this process belongs to (the person running it).
PID	Process ID.
PPID	Parent process ID (the ID of the process that started it).
C	CPU utilization of process.
STIME	Process start time.
TTY	Terminal type associated with the process
TIME	CPU time taken by the process.
CMD	The command that started this process.

ps options

$ ps -u usr : The system administrator needs to use the -u(user) option to know the activities of any user.

```
student@ugadmin22-IBMThink:~/trial$ ps -u student
  PID TTY          TIME CMD
 1895 ?        00:00:00 gnome-keyring-d
 1922 ?        00:00:00 init
 1979 ?        00:00:00 dbus-launch
 1980 ?        00:00:00 dbus-daemon
 1991 ?        00:00:00 ssh-agent
 1999 ?        00:00:02 dbus-daemon
 2007 ?        00:00:00 upstart-event-b
3002 ?        00:00:06 gnome-terminal
3028 ?        00:00:00 gnome-pty-helpe
3029 pts/1    00:00:00 bash
3093 ?        00:07:55 firefox
3113 ?        00:00:00 unity-webapps-s
3622 pts/1    00:00:00 ps
```

ps options

$ ps -a

Lists the processes of all users but doesn't display the system processes.

```
student@ugadmin22-IBMThink:~/trial$ ps -a
  PID TTY          TIME CMD
 3769 pts/1    00:00:00 ps
```

ps options

$ ps -e

```
pgadmin@pgadmin-HPXW4400:~/test$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:01 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:03 rcu_sched
    8 ?        00:00:01 rcuos/0
    9 ?        00:00:01 rcuos/1
   10 ?        00:00:00 rcuos/2
   11 ?        00:00:00 rcuos/3
   12 ?        00:00:00 rcu_bh
   13 ?        00:00:00 rcuob/0
   14 ?        00:00:00 rcuob/1
   15 ?        00:00:00 rcuob/2
 2516 ?        00:00:00 kworker/1:1H
 2539 ?        00:16:52 firefox
 2565 ?        00:00:00 unity-webapps-s
 2646 ?        00:00:04 gnome-terminal
 2653 ?        00:00:00 gnome-pty-helpe
 2654 pts/0    00:00:00 bash
 2705 ?        00:00:00 unity-scope-hom
 2718 ?        00:00:00 unity-scope-loa
 2722 ?        00:00:00 unity-files-dae
 3023 ?        00:00:00 cupsd
 3027 ?        00:00:00 dbus
 3108 ?        00:00:16 update-manager
 3445 ?        00:00:00 kworker/u8:3
 3508 ?        00:00:00 kworker/u8:0
 3527 ?        00:00:00 kworker/u8:1
 3708 ?        00:00:00 pickup
 3709 pts/0    00:00:00 ps
```

ps options

$ ps x

```
pgadmin@pgadmin-HPXW4400:~/test$ ps x
  PID TTY      STAT   TIME COMMAND
 1614 ?        Sl     0:00 /usr/bin/gnome-keyring-daemon --daemonize --login
 1617 ?        Ss     0:00 init --user
 1671 ?        S      0:00 dbus-launch --autolaunch=0484bc9c7667e409dff83bf65581
 1674 ?        Ss     0:00 //bin/dbus-daemon --fork --print-pid 5 --print-addres
 1686 ?        Ss     0:00 dbus-daemon --fork --session --address=unix:abstract=
 1697 ?        Ss     0:00 upstart-event-bridge
 1703 ?        Ss     0:00 /usr/lib/x86_64-linux-gnu/hud/window-stack-bridge
 1709 ?        S      0:00 upstart-dbus-bridge --daemon --system --user --bus-na
 1711 ?        S      0:00 upstart-file-bridge --daemon --user
 1713 ?        S      0:00 upstart-dbus-bridge --daemon --session --user --bus-n
 1714 ?        Ssl    0:01 /usr/bin/ibus-daemon --daemonize --xim
 1734 ?        Sl     0:00 /usr/lib/gvfs/gvfsd
```

ps options

$ ps aux

```
pgadmin@pgadmin-HPXW4400:~/test$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  33788  2032 ?        Ss   09:39   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    09:39   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    09:39   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   09:39   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    09:39   0:02 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    09:39   0:01 [rcuos/0]
root         9  0.0  0.0      0     0 ?        S    09:39   0:00 [rcuos/1]
root        10  0.0  0.0      0     0 ?        S    09:39   0:00 [rcuos/2]
root        11  0.0  0.0      0     0 ?        S    09:39   0:00 [rcuos/3]
```

Mechanism of process creation

There are three distinct phases and uses three important system calls.

Fork: A new process is created by means of the

fork() - system call

Exec: The parent then overwrites the image with the copy of the program that has to be executed.

Wait: The parent then execute the wait system call to wait for the child process to complete.

Creating a new process

- In UNIX, a new process is created by means of the **fork()** - system call. The OS performs the following functions:
  - It allocates a slot in the process table for the new process
  - It assigns a unique ID to the new process
  - It makes a copy of process image of the parent (except shared memory)
  - It returns the ID of the child to the parent process, and 0 to the child.

  - Note, the fork() call actually is called once but returns twice - namely in the parent and the child process.

Fork()

- Pid_t fork(void) is the prototype of the fork() call.
- Remember that fork() returns twice
  - in the newly created (child) process with return value 0
  - in the calling process (parent) with return value = pid of the new process.
  - A negative return value (-1) indicates that the call has failed

- Different return values are the key for distinguishing parent process from child process!
- The child process is an exact copy of the parent, yet, it is a copy i.e. an identical but

A fork() Example

```
#include <unistd.h>
main()
{
    pid_t pid     /* process id */
    printf("just one process before the fork()\n");
    pid = fork();
    if(pid == 0)
        printf("I am the child process\n");
    else if(pid > 0)
        printf("I am the parent process\n");
    else
        printf(" The fork() has failed\n")
}
```

A fork() Example

```
#include <stdio.h>
#include <string.h>
 #include <sys/types.h>
 #define MAX_COUNT 200
#define BUF_SIZE 100
void main(void)
 { pid_t pid; int i; char buf[BUF_SIZE];
 fork();
 pid = getpid();
 for (i = 1; i <= MAX_COUNT; i++)
 { sprintf(buf, "This line is from pid %d, value = %d\n", pid,
    i); write(1, buf, strlen(buf)); } }
```

```
#include <stdio.h>
 #include <sys/types.h>
 #define MAX_COUNT 200
void ChildProcess(void); /* child process prototype */
 void ParentProcess(void); /* parent process prototype */
void main(void) {
 pid_t pid; pid = fork();
if (pid == 0)
ChildProcess();
 else
ParentProcess(); }
void ChildProcess(void)
 { int i;
 for (i = 1; i <= MAX_COUNT; i++)
printf(" This line is from child, value = %d\n", i);
 printf(" *** Child process is done ***\n"); }
void ParentProcess(void)
 { int i;
for (i = 1; i <= MAX_COUNT; i++)
printf("This line is from parent, value = %d\n", i);
printf("*** Parent is done ***\n"); }
```

Basic Process Coordination

- The exit() call is used to terminate a process.
  - Its prototype is: void exit(int status), where status is used as the return value of the process.
  - exit(i) can be used to announce success and failure to the calling process.
- The wait() call is used to temporarily suspend the parent process until one of the child processes terminates.
  - The prototype is: pid_t wait(int *status), where status is a pointer to an integer to which the child's status information is being assigned.
  - wait() will return with a pid when any one of the children terminates or with -1 when no children exist.

more coordination

- To wait for a particular child process to terminate, we can use the waitpid() call.
  - Prototype: pid_t waitpid(pid_t pid, int *status, int opt)

- Sometimes we want to get information about the process or its parent.
  - getpid() returns the process id
  - getppid() returns the parent's process id
  - getuid() returns the users id
  - use the manual pages for more id information.