Process Management

Unix/Linux Commands

Topics

- Shell Commands
  - ps
  - kill
- Unix/Linux System Calls in C/C++
  - getpid () →process id
  - getppid() →parent process id
  - System()→info. About system
  - Fork ()
  - exec() / execv() / execl() / execve() / execvp() / execlp()
  - spawn()
  - wait() / waitpid()
  - kill()

Process Creation

- Once the OS decides to create a new process it:
  - Assigns a unique process identifier
  - Allocates space for the process
  - Initializes process control block (POC)
  - Sets up appropriate linkages
  - Creates or expand other data structures

System Call: Executing a command

- int **system** (*const char *command*)
  - This function executes *command* as a shell command

Sample Code : getpid() , getppid()

Listing 3.1 (*print-pid.c*) **Printing the Process ID**

```
#include <stdio.h>
#include <unistd.h>

int main ()
{
  printf ("The process ID is %d\n", (int) getpid ());
  printf ("The parent process ID is %d\n", (int) getppid ()
  return 0;
}
```

saeed@saeed-virtual-machine: ~/PM

```
saeed@saeed-virtual-machine:~$ cd PM
saeed@saeed-virtual-machine:~/PM$ gcc p1_pid.c -o p1_pid.out
saeed@saeed-virtual-machine:~/PM$ ./p1_pid.out
The process ID is 4028
The parent process ID is 3760
saeed@saeed-virtual-machine:~/PM$
```

Sample Code : system()

Listing 3.2 (*system.c*) **Using the *system* Call**

```
#include <stdlib.h>

int main ()
{
  int return_value;
  return_value = system ("ls -l /");
  return return_value;
}
```

saeed@saeed-virtual-machine: ~/PM

```
saeed@saeed-virtual-machine:~/PM$ gcc p2_system.c -o p2_system.out
saeed@saeed-virtual-machine:~/PM$ ./p2_system.out
total 88
drwxr-xr-x   2 root root  4096 Feb 11 21:15 bin
drwxr-xr-x   3 root root  4096 Feb 11 21:15 boot
drwxr-xr-x   2 root root  4096 Feb 11 21:07 cdrom
drwxr-xr-x  15 root root  4240 Mar  5 22:00 dev
drwxr-xr-x 138 root root 12288 Mar  5 22:00 etc
```

Sample Code : system()

Listing 3.2 (*system.c*) **Using the *system* Call**

```
#include <stdlib.h>

int main ()
{
  int return_value;
  return_value = system ("ls -l /");
  return return_value;
}
```

saeed@saeed-virtual-machine: ~/PM

```
saeed@saeed-virtual-machine:~/PM$ gcc p2_system.c -o p2_system.out
saeed@saeed-virtual-machine:~/PM$ ./p2_system.out
total 88
drwxr-xr-x   2 root root  4096 Feb 11 21:15 bin
drwxr-xr-x   3 root root  4096 Feb 11 21:15 boot
drwxr-xr-x   2 root root  4096 Feb 11 21:07 cdrom
drwxr-xr-x  15 root root  4240 Mar  5 22:00 dev
drwxr-xr-x 138 root root 12288 Mar  5 22:00 etc
```

System Call: Creating a Process

- pid_t **fork** (*void*) / pid_t **vfork** (*void*)

fork() makes a complete copy of the calling process's address space and allows **both the parent and child to execute independently**

vfork() does not make copy. Instead, the child process created with vfork shares its parent's address space until it calls **_exit** or one of the **exec** functions. In the meantime, the **parent process suspends execution.**

fork()	vfork()
Separate Address Space / 2 images	Shared address space / 1 image
Both parent & child run	Parent suspended

Process Creation

- Process creation is by means of the kernel system call,fork( ).
- This causes the OS, in Kernel Mode, to:
  1. Allocate a slot in the process table for the new process.
  2. Assign a unique process ID to the child process.
  3. Copy of process image of the parent, with the exception of any shared memory.

Process Creation

4. Increment the counters for any files owned by the parent, to reflect that an additional process now also owns those files.
5. Assign the child process to the Ready to Run state.
6. Returns the ID number of the child to the parent process, and a 0 value to the child process.

After Creation

- After creating the process the Kernel can do one of the following, as part of the dispatcher routine:
  - Stay in the parent process.
  - Transfer control to the child process
  - Transfer control to another process.

Sample Code : fork()

Listing 3.3 (*fork.c*) Using *fork* to Duplicate a Program's Process

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

int main ()
{
  pid_t child_pid;

  printf ("the main program process ID is %d\n", (int) getpid ());

  child_pid = fork ();
  if (child_pid != 0) {
    printf ("this is the parent process, with id %d\n", (int) getpid ());
    printf ("the child's process ID is %d\n", (int) child_pid);
  }
  else
    printf ("this is the child process, with id %d\n", (int) getpid ());

  return 0;
}
```

saeed@saeed-virtual-machine: ~/PM

```
saeed@saeed-virtual-machine:~/PM$ gcc p3_fork.c -o p3_fork.out
saeed@saeed-virtual-machine:~/PM$ ps
  PID TTY          TIME CMD
 3760 pts/0    00:00:00 bash
 4095 pts/0    00:00:00 ps
saeed@saeed-virtual-machine:~/PM$ ./p3_fork.out
the main program process ID is 4096
this is the parent process, with id 4096
the child's process ID is 4097
saeed@saeed-virtual-machine:~/PM$ this is the child process, with id 4097
```

System Call: Executing a file

executes the file named by *filename* as a **new process image**.

int **execv** (*const char *filename, char *const argv*[])

int **execvp** (*const char *filename, char *const argv*[])

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

int **execl** (*const char *filename, const char *arg0, ...*)

int **execlp** (*const char *filename, const char *arg0, ...*)

Sample Code : fork() , exec() . Listing:3.4

```
int main ()
{
  /* The argument list to pass to the "ls" command.  */
  char* arg_list[] = {
    "ls",     /* argv[0], the name of the program.  */
    "-l",
    "/",
    NULL      /* The argument list must end with a NULL.  */
  };

  /* Spawn a child process running the "ls" command.  Ignore the
     returned child process ID.  */
  spawn ("ls", arg_list);

  printf ("done with main program\n");

  return 0;
}
```

```
int spawn (char* program, char**
{
  pid_t child_pid;

  /* Duplicate this process.  */
  child_pid = fork ();
  if (child_pid != 0)
    /* This is the parent process
    return child_pid;
  else {
    /* Now execute PROGRAM, searching for it in the path.  */
    execvp (program, arg_list);
    /* The execvp function returns only if an error occurs.  */
    fprintf (stderr, "an error occurred in execvp\n");
    abort ();
  }
}
```

Output : fork() , exec() . Listing:3.4

```
saeed@saeed-virtual-machine:~/PM$ gcc p4_fork_exec.c -o p4_fork_exec.out
saeed@saeed-virtual-machine:~/PM$ ./p4_fork_exec.out
done with main program
saeed@saeed-virtual-machine:~/PM$ total 88
drwxr-xr-x   2 root root  4096 Feb 11 21:15 bin
drwxr-xr-x   3 root root  4096 Feb 11 21:15 boot
drwxr-xr-x   2 root root  4096 Feb 11 21:07 cdrom
drwxr-xr-x  15 root root  4240 Mar  5 22:00 dev
drwxr-xr-x 138 root root 12288 Mar  5 22:00 etc
drwxr-xr-x   4 root root  4096 Mar  5 20:20 home
lrwxrwxrwx   1 root root    33 Feb 11 21:13 initrd.img -> boot/initrd.img-3.11.0
15-generic
drwxr-xr-x  20 root root  4096 Feb 11 21:15 lib
drwx------   2 root root 16384 Feb 11 21:00 lost+found
drwxr-xr-x   4 root root  4096 Mar  5 22:00 media
drwxr-xr-x   2 root root  4096 Apr 19  2012 mnt
drwxr-xr-x   2 root root  4096 Feb  4 18:42 opt
dr-xr-xr-x 163 root root     0 Mar  5 20:16 proc
drwx------   5 root root  4096 Mar  5 21:23 root
drwxr-xr-x  21 root root   760 Mar  5 20:17 run
drwxr-xr-x   2 root root  4096 Feb 11 21:15 sbin
drwxr-xr-x   2 root root  4096 Mar  5  2012 selinux
drwxr-xr-x   2 root root  4096 Feb  4 18:42 srv
dr-xr-xr-x  13 root root     0 Mar  5 20:16 sys
drwxrwxrwt  11 root root  4096 Mar  5 22:17 tmp
drwxr-xr-x  10 root root  4096 Feb  4 18:42 usr
drwxr-xr-x  13 root root  4096 Feb 11 21:53 var
lrwxrwxrwx   1 root root    30 Feb 11 21:13 vmlinuz -> boot/vmlinuz-3.11.0-15-ge
eric
saeed@saeed-virtual-machine:~/PM$ ps
```

System Call

Waiting for a process completion

- pid_t **waitpid** (*pid_t pid, int *status-ptr, int options*)
- pid_t **wait** (*int *status-ptr*)

calling process is **suspended** until the **child process** makes status information available by **terminating**.

Sample Code : wait()

```
int main ()
{
  int child_status;

  /* The argument list to pass to the "ls" command.  */
  char* arg_list[] = {
    "ls",     /* argv[0], the name of the program.  */
    "-l",
    "/",
    NULL      /* The argument list must end with a NULL.  */
  };

  /* Spawn a child process running the "ls" command.  Ignore the
     returned child process ID.  */
  spawn ("ls", arg_list);

  /* Wait for the child process to complete.  */
  wait (&child_status);
  if (WIFEXITED (child_status))
    printf ("the child process exited normally, with exit code %d\n",
            WEXITSTATUS (child_status));
  else
    printf ("the child process exited abnormally\n");

  return 0;
}
```

Sample Code : wait()

```
saeed@saeed-virtual-machine: ~/PM
saeed@saeed-virtual-machine:~/PM$ gcc p5_fork_exec_wait.c -o p5_fork_exec_wait.o
ut
saeed@saeed-virtual-machine:~/PM$ ./p5_fork_exec_wait.out
total 88
drwxr-xr-x   2 root root  4096 Feb 11 21:15 bin
drwxr-xr-x   3 root root  4096 Feb 11 21:15 boot
drwxr-xr-x   2 root root  4096 Feb 11 21:07 cdrom
drwxr-xr-x  15 root root  4240 Mar  5 22:00 dev
drwxr-xr-x 138 root root 12288 Mar  5 22:00 etc
drwxr-xr-x   4 root root  4096 Mar  5 20:20 home
lrwxrwxrwx   1 root root    33 Feb 11 21:13 initrd.img -> boot/initrd.img-3.11.0
-15-generic
drwxr-xr-x  20 root root  4096 Feb 11 21:15 lib
drwx------   2 root root 16384 Feb 11 21:00 lost+found
drwxr-xr-x   4 root root  4096 Mar  5 22:00 media
drwxr-xr-x   2 root root  4096 Apr 19  2012 mnt
drwxr-xr-x   2 root root  4096 Feb  4 18:42 opt
dr-xr-xr-x 165 root root     0 Mar  5 20:16 proc
drwx------   5 root root  4096 Mar  5 21:23 root
drwxr-xr-x  21 root root   760 Mar  5 20:17 run
drwxr-xr-x   2 root root  4096 Feb 11 21:15 sbin
drwxr-xr-x   2 root root  4096 Mar  5  2012 selinux
drwxr-xr-x   2 root root  4096 Feb  4 18:42 srv
dr-xr-xr-x  13 root root     0 Mar  5 20:16 sys
drwxrwxrwt  11 root root  4096 Mar  5 22:23 tmp
drwxr-xr-x  10 root root  4096 Feb  4 18:42 usr
drwxr-xr-x  13 root root  4096 Feb 11 21:53 var
lrwxrwxrwx   1 root root    30 Feb 11 21:13 vmlinuz -> boot/vmlinuz-3.11.0-15-ge
neric
the child process exited normally, with exit code 0
saeed@saeed-virtual-machine:~/PM$
```

Sample Code : zombie process

Listing 3.6 (*zombie.c*) Making a Zombie Process

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main ()
{
  pid_t child_pid;

  /* Create a child proc
  child_pid = fork ();
  if (child_pid > 0) {
    /* This is the parent process.  Sleep for a minute.*/
    sleep (60);
  }
  else {
    /* This is the child process.  Exit immediately.  */
    exit (0);
  }
  return 0;
}
```

saeed@saeed-virtual-machine: ~/PM

```
saeed@saeed-virtual-machine:~/PM$ gcc p6_zombie.c -o p6_zombie.out
saeed@saeed-virtual-machine:~/PM$ ./p6_zombie.out &
[1] 4260
saeed@saeed-virtual-machine:~/PM$ ps
  PID TTY          TIME CMD
 3760 pts/0    00:00:00 bash
 4260 pts/0    00:00:00 p6_zombie.out
 4261 pts/0    00:00:00 p6_zombie.out <defunct>
 4262 pts/0    00:00:00 ps
saeed@saeed-virtual-machine:~/PM$ ps
  PID TTY          TIME CMD
 3760 pts/0    00:00:00 bash
 4263 pts/0    00:00:00 ps
[1]+  Done                    ./p6_zombie.out
```

reference

- http://www.advancedlinuxprogramming.com/alp-folder/alp-ch03-processes.pdf