3. Linux/Unix processes

Master M1 MOSIG, Grenoble Universities
Arnaud Legrand, Sascha Hunold

2010

Abstract

This lab is about:

- work with processes and pipes in Linux
- `fork` and `pipe` primitives
- shared memory segments
- synchronization with busy waiting

1 Fork/Wait

Write a program that executes n processes using `fork()`. Each child process should do a `sleep()` followed by printing its own pid to stdout. Then each child exits. What do we have to take care of?

2 Fork/Exec

Write a program that creates 2 processes. The second process should be created via `fork()`. After the fork, the child should execute (`exec*`) the binary `/bin/ls`. The parent process should print a hello world message after forking. How many hello world messages will be printed?

3 Simple Pipe

Write a program that creates two processes. Create a pipe and send the pid of the parent process to the child process! Let the child and the parent print the pid of the parent.

4 Pipes 2 – Drawing random numbers

We would like to write a program executed by n processes which pick a random number and print it at the screen. Here is an example:

```
$ ring1 6
processus pid 25387 node 2 val = 1430826605
processus pid 25388 node 3 val = 48523501
processus pid 25389 node 4 val = 822619539
processus pid 25390 node 5 val = 1591287596
processus pid 25385 node 0 val = 2047288621
processus pid 25386 node 1 val = 1731323093
$
```

This program is composed of 6 processes. The number of processes should be a parameter of the program. The processes are identified by their `pid` (returned by `fork`), a number (the order of creation) and a random number (generated with `rand`). In order to have different random values, read carefully `man` and `srand()`.

5 Pipes 3 – Determine the winner

The goal is to adapt the previous program in order to decide which process generates the biggest random number. To do this, we will use an election algorithm. We will interconnect the processes with pipes so as to generate a ring topology. This means that process 0 will be connected to process 1 through a pipe, process 1 will be connected to process 2 through another pipe, etc. The last process will be connected to process 0.

When the processes and the pipes are created, process 0 will send its random value to process 1. Process 1 will compare it with its proper random value and will send the bigger one to process 2. The other processes will work in the same way. At the end, process 0 will receive the biggest value, as well as the information concerning the pid and the number of the process that generated it.

```
$ ring2 6
processus pid 26603 node 1 val = 982695543
processus pid 26604 node 2 val = 679092432
processus pid 26605 node 3 val = 1441867048
processus pid 26606 node 4 val = 1135529882
processus pid 26607 node 5 val = 1906462017
processus pid 26602 node 0 val = 208930720
the winner is 1906462017 pid 26607 node 5
$
```

6 Pipes 4 – Everyone should know the winner

In this exercise the goal is to notify all the processes of the pid of the winning process. In the previous exercise only process 0 knows the identity of the winner. In this exercise, process 0 should pass the pid of the winning process along the ring.

```
$ ring3 6
processus pid 27739 node 1 val = 161183994
processus pid 27740 node 2 val = 925511728
```

```
processus pid 27741 node 3 val = 1709276223
processus pid 27742 node 4 val = 1410188147
processus pid 27743 node 5 val = 1099100972
processus pid 27738 node 0 val = 1532950038
Node 0 the winner is 1709276223 pid 27741 node 3
Node 1 the winner is 1709276223 pid 27741 node 3
Node 2 the winner is 1709276223 pid 27741 node 3
Node 3 the winner is 1709276223 pid 27741 node 3
Node 4 the winner is 1709276223 pid 27741 node 3
Node 5 the winner is 1709276223 pid 27741 node 3
$
```

7 Shared Memory Segments!

It was probably difficult to create the ring using pipes.

We offer you now to create a ring using shared memory.

A shared memory segment is a memory zone, which is shared among multiple processes. If a process writes to this zone, the modification will be automatically visible to the other processes.

In order to create a segment of shared memory, one needs to allocate it (`shmget`) and then attach it (`shmat`).

Here is an example:

```
void * create_shared_array (char **argv)
{
        key_t key ;
        int shmid ;
        void *ptr ;
        key = ftok (argv[0], 'a') ;
        if (key == -1)
        {
                fprintf (stderr, "erreur ftok \n") ;
                exit (-1) ;
        }
        shmid = shmget (key, 3 * sizeof(int) * nb_proc, IPC_CREAT|0666) ;
        if (shmid == -1)
        {
                perror ("shmget") ;
                fprintf (stderr, "erreur shmget \n") ;
                exit (-1) ;
        }
        ptr = shmat (shmid, NULL, 0) ;
        if (ptr == (entry *) -1)
        {
                perror ("shmat") ;
                fprintf (stderr, "erreur shmat \n", argv[0]) ;
                exit (-1) ;
        }
        return ptr ;
```

```
}
```

At the end of the function `create_shared_array`, the pointer points to the shared memory segment. This pointer is returned as the result of the function.

In order to destruct a shared memory segment, one needs only to detach it. Here is an example.

```
void delete_shared_memory (void *ptr)
{
int cr ;
cr = shmdt (ptr) ;
}
```

The segment will be shared among processes. Each process will write its data in the memory zone. It will wait for the other processes to finish writing to the zone. Once all the processes have written their values to the zone, each process consults the data in order to decide who is the winner.

8 List System Processes

This exercise is optional.

The goal is to list the processes running at the system as it is done by top, ps, etc. In order to do this, you will need to consult the directory `proc`.

```
 $ ls /proc
1 1630 2024 2415 2538 2717 2926 2977 3032 332 5015 860 diskstats kcore self
11980 17666 2027 2449 2553 2771 2928 298 3040 3337 5017 861 dma kmsg slabinfo
1291 17667 203 2459 2569 2772 2943 299 3048 334 6 9 driver loadavg stat
1293 19144 204 2499 2578 2773 2947 3 305 337 690 966 execdomains locks swaps
1295 19190 205 2512 2590 2774 2948 3006 3094 3449 7 acpi fb meminfo sys
1297 19330 206 2513 2604 2775 3096 3454 749 asound filesystems misc sysvipc
1299 19352 207 2520 2631 2776 2964 301 3106 3783 762 buddyinfo fs modules tty
13 19360 2120 2527 2640 2867 2967 3012 3107 3803 775 bus interrupts mounts uptime
14 19559 2131 2528 2670 2911 2968 3017 3140 4 8 cmdline iomem mtrr version
15 1986 2132 2533 2705 2914 2969 3022 3239 457 857 cpuinfo ioports [net vmstat
158 1996 2155 2535 2707 2915 297 4992 858 crypto irq partitions zoneinfo
1602 2 2163 2537 2713 2917 2974 3027 3313 5 859 devices kallsyms scsi
$
```

The bold names are those of directories. The directories whose names are numbers correspond to processes. Each process has a corresponding directory whose name is the `pid` of the process. The functions to use are `opendir`, `readdir`, `closedir`.

```
  $ ls 19352
dr-xr-xr-x 4 mehaut mehaut 0 2007-10-04 08:59 .
dr-xr-xr-x 149 root root 0 2007-10-03 19:52 ..
-r-------- 1 mehaut mehaut 0 2007-10-04 09:34 auxv
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 cmdline
lrwxrwxrwx 1 mehaut mehaut 0 2007-10-04 09:34 cwd -> /home/mehaut/CSE/anneau
-r-------- 1 mehaut mehaut 0 2007-10-04 09:34 environ
lrwxrwxrwx 1 mehaut mehaut 0 2007-10-04 08:59 exe -> /usr/bin/xpdf.bin
dr-x------ 2 mehaut mehaut 0 2007-10-04 09:34 fd
```

```
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 maps
-rw------- 1 mehaut mehaut 0 2007-10-04 09:34 mem
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 mounts
-r-------- 1 mehaut mehaut 0 2007-10-04 09:34 mountstats
-rw-r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 oom_adj
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 oom_score
lrwxrwxrwx 1 mehaut mehaut 0 2007-10-04 09:34 root -> /
-rw------- 1 mehaut mehaut 0 2007-10-04 09:34 seccomp
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 smaps
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 stat
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 statm
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 status
dr-xr-xr-x 3 mehaut mehaut 0 2007-10-04 09:34 task
-r--r--r-- 1 mehaut mehaut 0 2007-10-04 09:34 wchan
$
```

So we find:

- the exe link to the executable
- the cwd containing the current directory of the process
- the environ file with its environment variables
- etc.

Starting from this data, you should be able to have a listing close to the result of the `ps` command.