Lecture VII
Unix Process Environment

Tevfik Koşar
Louisiana State University
September 14th, 2010

In Today’s Class

- Unix Process Environment
  - Process Concept
  - Creation & Termination of Processes
  - Exec() & Fork()
  - ps -- get process info
  - Shell & its implementation

Process Concept

- a Process is a program in execution;
- A process image consists of three components:
  1. an executable program
  2. the associated data needed by the program
  3. the execution context of the process, which contains all information the O/S needs to manage the process (ID, state, CPU registers, stack, etc.)

Process Control Block

- The Process Control Block (PCB)
  - is included in the context, along with the stack
  - is a “snapshot” that contains all necessary and sufficient data to restart a process where it left off (ID, state, CPU registers, etc.)
  - is one entry in the operating system’s process table (array or linked list)

Process State

- As a process executes, it changes state
  - new: The process is being created
  - ready: The process is waiting to be assigned to a process
  - running: Instructions are being executed
  - waiting: The process is waiting for some event to occur
  - terminated: The process has finished execution
$ ps

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>TIME</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>18684</td>
<td>pts/4</td>
<td>00:00:00</td>
<td>bash</td>
</tr>
<tr>
<td>18705</td>
<td>pts/4</td>
<td>00:00:00</td>
<td>ps</td>
</tr>
</tbody>
</table>

$ ps a

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>STAT</th>
<th>TIME COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>6702</td>
<td>tty7</td>
<td>S+</td>
<td>15:10 /usr/X11R6/bin/X :0 -audit 0</td>
</tr>
<tr>
<td>7024</td>
<td>tty1</td>
<td>S+</td>
<td>/bin/minigetty --nuclear tty1</td>
</tr>
<tr>
<td>7025</td>
<td>tty2</td>
<td>S+</td>
<td>/bin/minigetty tty2</td>
</tr>
<tr>
<td>7026</td>
<td>tty3</td>
<td>S+</td>
<td>/bin/minigetty tty3</td>
</tr>
<tr>
<td>7027</td>
<td>tty4</td>
<td>S+</td>
<td>/bin/minigetty tty4</td>
</tr>
<tr>
<td>7028</td>
<td>tty5</td>
<td>S+</td>
<td>/bin/minigetty tty5</td>
</tr>
<tr>
<td>7029</td>
<td>tty6</td>
<td>S+</td>
<td>/bin/minigetty tty6</td>
</tr>
<tr>
<td>17166</td>
<td>pts/6</td>
<td>S</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17191</td>
<td>pts/6</td>
<td>S</td>
<td>0:00 pico program3.cc</td>
</tr>
<tr>
<td>17484</td>
<td>pts/5</td>
<td>S</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17555</td>
<td>pts/7</td>
<td>S</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17646</td>
<td>pts/8</td>
<td>S</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17809</td>
<td>pts/10</td>
<td>S</td>
<td>0:00 /sbin/mingetty --noclear tty1</td>
</tr>
<tr>
<td>17977</td>
<td>pts/1</td>
<td>S+</td>
<td>0:00 pico prog2.java</td>
</tr>
<tr>
<td>18014</td>
<td>pts/9</td>
<td>S+</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>18151</td>
<td>pts/1</td>
<td>S+</td>
<td>0:00 pico program3.cc</td>
</tr>
<tr>
<td>18259</td>
<td>pts/10</td>
<td>T</td>
<td>0:00 a.out</td>
</tr>
<tr>
<td>18443</td>
<td>pts/2</td>
<td>S</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>18684</td>
<td>pts/4</td>
<td>S</td>
<td>0:00 -bash</td>
</tr>
</tbody>
</table>

$ ps la

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PID</th>
<th>PPID</th>
<th>PRI</th>
<th>NI</th>
<th>VSZ</th>
<th>RSS</th>
<th>WCHAN</th>
<th>STAT</th>
<th>TTY</th>
<th>TIME COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>6702</td>
<td></td>
<td>6702</td>
<td>6701</td>
<td>15</td>
<td>0</td>
<td>25416</td>
<td>7204</td>
<td>-</td>
<td>Ss+</td>
<td>tty7</td>
<td>15:10 /usr/X11R6/bin/X :0 -audit 0 -auth /var/lib/g</td>
</tr>
<tr>
<td>7024</td>
<td></td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>3008</td>
<td>4</td>
<td>-</td>
<td>S</td>
<td>tty1</td>
<td>0:00 /sbin/mingetty --noclear tty1</td>
</tr>
<tr>
<td>7025</td>
<td></td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>3008</td>
<td>4</td>
<td>-</td>
<td>S</td>
<td>tty2</td>
<td>0:00 /sbin/mingetty tty2</td>
</tr>
<tr>
<td>7026</td>
<td></td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>3008</td>
<td>4</td>
<td>-</td>
<td>S</td>
<td>tty3</td>
<td>0:00 /sbin/mingetty tty3</td>
</tr>
<tr>
<td>7027</td>
<td></td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>3008</td>
<td>4</td>
<td>-</td>
<td>S</td>
<td>tty4</td>
<td>0:00 /sbin/mingetty tty4</td>
</tr>
<tr>
<td>7028</td>
<td></td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>3008</td>
<td>4</td>
<td>-</td>
<td>S</td>
<td>tty5</td>
<td>0:00 /sbin/mingetty tty5</td>
</tr>
<tr>
<td>7029</td>
<td></td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>3008</td>
<td>4</td>
<td>-</td>
<td>S</td>
<td>tty6</td>
<td>0:00 /sbin/mingetty tty6</td>
</tr>
<tr>
<td>17166</td>
<td></td>
<td>17166</td>
<td>17165</td>
<td>15</td>
<td>0</td>
<td>9916</td>
<td>2300</td>
<td>wait</td>
<td>S</td>
<td>pts/6</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17191</td>
<td></td>
<td>17191</td>
<td>17166</td>
<td>16</td>
<td>0</td>
<td>8688</td>
<td>1264</td>
<td>-</td>
<td>S+</td>
<td>pts/6</td>
<td>0:00 pico program3.cc</td>
</tr>
<tr>
<td>17484</td>
<td></td>
<td>17484</td>
<td>17483</td>
<td>16</td>
<td>0</td>
<td>9916</td>
<td>2300</td>
<td>-</td>
<td>Ss+</td>
<td>pts/5</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17555</td>
<td></td>
<td>17555</td>
<td>17554</td>
<td>15</td>
<td>0</td>
<td>9912</td>
<td>2292</td>
<td>-</td>
<td>S</td>
<td>pts/7</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17646</td>
<td></td>
<td>17646</td>
<td>17644</td>
<td>16</td>
<td>0</td>
<td>9912</td>
<td>2300</td>
<td>wait</td>
<td>S</td>
<td>pts/8</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17809</td>
<td></td>
<td>17809</td>
<td>17808</td>
<td>15</td>
<td>0</td>
<td>9916</td>
<td>2304</td>
<td>wait</td>
<td>S</td>
<td>pts/10</td>
<td>0:00 -bash</td>
</tr>
<tr>
<td>17977</td>
<td></td>
<td>17977</td>
<td>17976</td>
<td>16</td>
<td>0</td>
<td>8688</td>
<td>1340</td>
<td>-</td>
<td>S</td>
<td>pts/1</td>
<td>0:00 -bash</td>
</tr>
</tbody>
</table>

$ ps -ax

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>STAT</th>
<th>TIME COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>?</td>
<td>S</td>
<td>0:02 init [5]</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
<td>S</td>
<td>0:00 [migration/0]</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>SN</td>
<td>0:01 [ksoftirqd/0]</td>
</tr>
<tr>
<td>4</td>
<td>?</td>
<td>S</td>
<td>0:00 [migration/1]</td>
</tr>
<tr>
<td>5</td>
<td>?</td>
<td>SN</td>
<td>0:00 [ksoftirqd/1]</td>
</tr>
<tr>
<td>6</td>
<td>?</td>
<td>S</td>
<td>0:00 [migration/2]</td>
</tr>
<tr>
<td>7</td>
<td>?</td>
<td>SN</td>
<td>0:16 [ksoftirqd/2]</td>
</tr>
<tr>
<td>8</td>
<td>?</td>
<td>S</td>
<td>0:00 [migration/3]</td>
</tr>
<tr>
<td>9</td>
<td>?</td>
<td>SN</td>
<td>0:16 [ksoftirqd/3]</td>
</tr>
<tr>
<td>10</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [events/0]</td>
</tr>
<tr>
<td>11</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [events/1]</td>
</tr>
<tr>
<td>12</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [events/2]</td>
</tr>
<tr>
<td>13</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [events/3]</td>
</tr>
<tr>
<td>14</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [khelper]</td>
</tr>
<tr>
<td>15</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [kthread]</td>
</tr>
<tr>
<td>653</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [kmpf]</td>
</tr>
<tr>
<td>994</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [kblockd/0]</td>
</tr>
<tr>
<td>995</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [kblockd/1]</td>
</tr>
<tr>
<td>996</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:01 [kblockd/2]</td>
</tr>
<tr>
<td>997</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>0:00 [kblockd/3]</td>
</tr>
</tbody>
</table>

Process Creation

- Some events that lead to process creation (enter)
  - When the system is initialized, several background processes or “daemons” are started (email, logon, etc.)
  - A user requests to run an application
    - By typing a command in the CLI shell or double-clicking in the GUI shell, the user can launch a new process
  - An existing process spawns a child process
    - For example, a server process (print, file) may create a new process for each request it handles
  - The init daemon waits for user login and spawns a shell
    - A batch system takes on the next job in line

Process creation by spawning

A tree of processes on a typical UNIX system
### Process Creation

```c
int main(...) {
    ... if ((pid = fork()) == 0) // create a process
        fprintf(stdout, "Child pid: %i\n", getpid());
        err = execvp(command, arguments); // execute child
        fprintf(stderr, "Child error: %i\n", errno);
        exit(err);
    else if (pid > 0) // we are in the parent process
        fprintf(stdout, "Parent pid: %i\n", getpid());
        pid2 = waitpid(pid, &status, 0); // wait for child
    ... return 0;
}
```

### Process Creation

1. **Clone child process**
   - `pid = fork()`
2. **Replace child’s image**
   - `execvp(name, ...)`

### Fork Example 1

```c
#include <stdio.h>

main() {
    int ret_from_fork, mypid;
    mypid = getpid(); /* who am i? */
    printf("Before: my pid is %d\n", mypid);
    ret_from_fork = fork();
    sleep(1);
    printf("After: my fork returns pid : %d, said %d\n",
              ret_from_fork, getpid());
    }
```

### Fork Example 2

```c
#include <stdio.h>

main() {
    fork();
    fork();
    fork();
    printf("my pid is %d\n", getpid());
}
```

**How many lines of output will this produce?**

### Shell

- A tool for process and program control
- Three main functions
  - Shells run programs
  - Shells manage I/O
  - Shells can be programmed
- **Main Loop of a Shell**

  ```c
  while (!end_of_input){
      get command
      execute command
      wait for command to finish
  }
  ```

### How does a Program run another Program?

- Program calls `execvp`

  ```c
  int execvp(const char *file, char *const argv[]);
  ```
- Kernel loads program from disk into the process
- Kernel copies `arglist` into the process
- Kernel calls `main(argc, argv)`
Exec Family

int exec(const char *path, const char *arg, ...);
int execvp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);

execvp is like a Brain Transplant

- execvp loads the new program into the current process, replacing the code and data of that process!

Running “ls -l”

```c
#include <unistd.h>
#include <stdio.h>
main()
{
    char *arglist[3];
    arglist[0] = "ls";
    arglist[1] = "-l";
    arglist[2] = 0;
    printf("* * * About to exec ls -l\n");
    execvp( "ls" , arglist );
    printf("* * * ls is done. bye\n");
}
```

Writing a Shell v1.0

```c
int execute( char *arglist[] )
{
    execvp(arglist[0], arglist);
    perror("execvp failed");
    exit(1);
}
```

Writing a Shell v2.0

```c
execute( char *arglist[] )
{
    int pid, exitstatus; /* of child*/
    pid = fork(); /* make new process */
    switch( pid ){
    case -1:
        perror("fork failed");
        exit(1);
    case 0:
        execvp(arglist[0], arglist); /* do it */
        perror("execvp failed");
        exit(1);
        default:
            while( wait(&exitstatus) != pid ) ;
        printf("child exited with status %d,%d\n", exitstatus>>8, exitstatus&0377);
    }
}
Process Termination

- Some events that lead to process termination (exit)
  - regular completion, with or without error code
    - the process voluntarily executes an `exit(err)` system call to indicate to the O/S that it has finished
  - fatal error (uncatchable or uncaught)
    - service errors: no memory left for allocation, I/O error, etc.
    - total time limit exceeded
    - arithmetic error, out-of-bounds memory access, etc.
  - killed by another process via the kernel
    - the process receives a SIGKILL signal
    - in some systems the parent takes down its children with it

Exercise

Improve the Shell v2.0 by:
- Allow the user to type all the arguments on one line
- Allow the user to quit by typing exit

Summary

- Unix Process Environment
  - Process Concept
  - ps -- get process info
  - Shell & its Implementation
  - Exec() & Fork()
  - Creation & Termination of Processes

- Next Class: Process Control
  - Try “fork” and “shell” examples

Acknowledgments

- Advanced Programming in the Unix Environment by R. Stevens
- The C Programming Language by B. Kernighan and D. Ritchie
- Understanding Unix/Linux Programming by B. Molay
- Lecture notes from B. Molay (Harvard), T. Kuo (UT-Austin), G. Pierre (Vrije), M. Matthews (SC), and B. Knicki (WPI).