Advance Network Programming

Processes

Outline

- Process definition
- Process handling in Unix
- Process creation
- Process termination
- Process synchronization

Process Definition

- Definition: A process is an instance of a running program.
  - One of the most fundamental concepts in computer science.
  - Not the same as “program” or “processor”
- A program is a set of instructions and initialized data in a file, usually found on a disk.
- A process is an instance of that program while it is running, along with the state of all the CPU registers and the values of data in memory.
- A single program can correspond to many processes; for example, several users can be running a shell.

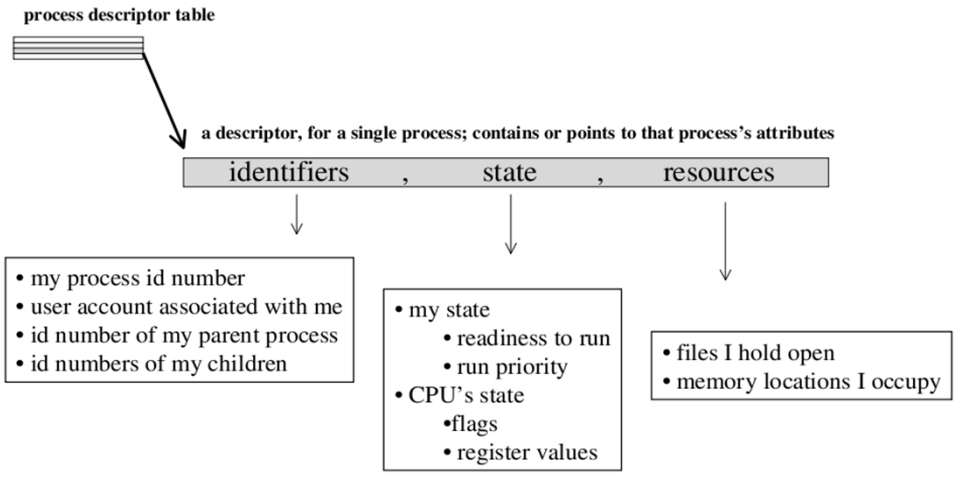
Process Handling

- Constituents of a process
  - Its code
  - Data
    - its own
    - OS’s data used by/for process
  - Various attributes OS needs to manage it

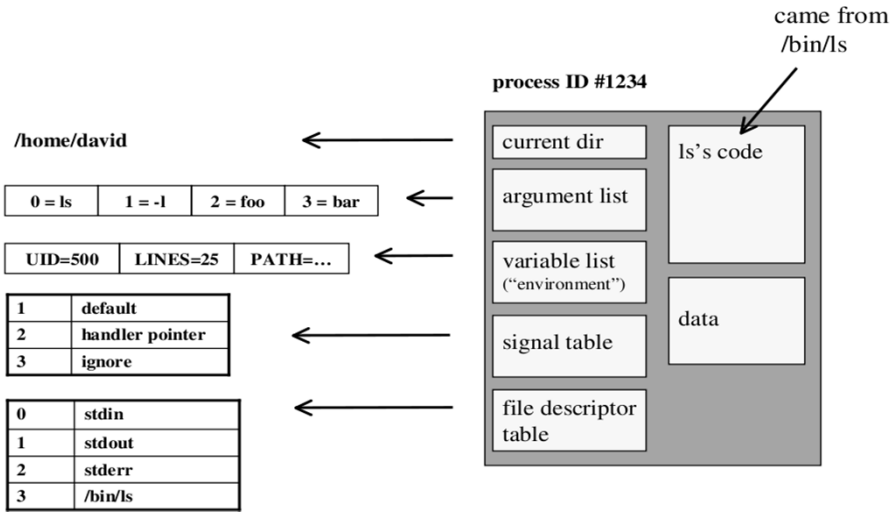
Process Handling

- OS keeps track of all processes
  - Process table/array/list
  - Elements are process descriptors (aka control blocks)
  - Descriptors reference code & data

Process Descriptor



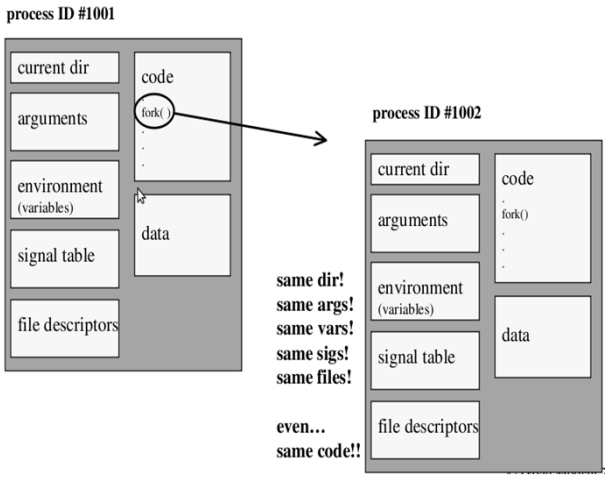
ls -l foo bar



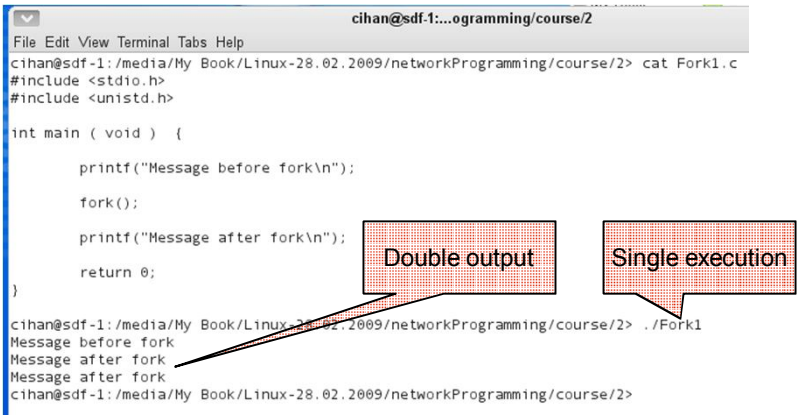
Process Creation in Unix

- fork() function
- exec() family of functions
- spawn()
- wait()
- exit()

fork() system call



fork() example



Process Differentiation

- identical? not what we had in mind!
- more useful if child does different stuff
- can we give it different behavior?

provide different behavior

- in the form of source code
- in the form of an existing binary executable
  - exec() family of functions

fork - how to self-identify?

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

```
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> cat Fork2.c
#include <stdio.h>
#include <unistd.h>

int main ( void )  {

        int forkResult;

        printf("process id : %i\n",getpid());
        forkResult = fork();
        printf("process id : %i - result : %d\n",getpid(), forkResult);

        return 0;
}

cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ./Fork2
process id : 308
process id : 309 - result : 0
process id : 308 - result : 309
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2>
```

If 0, I must be the child copy

If not, I must be the parent copy

By source code

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

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

int main ( void )  {

        printf("(%i) Parent does something...\n", getpid());

        if(fork()) {    // Parent
                printf("(%i) Parent do completely different stuff\n",getpid());
        } else {        // Child
                printf("(%i) Child can do some stuff\n",getpid());
        }


        exit(0);
}
```

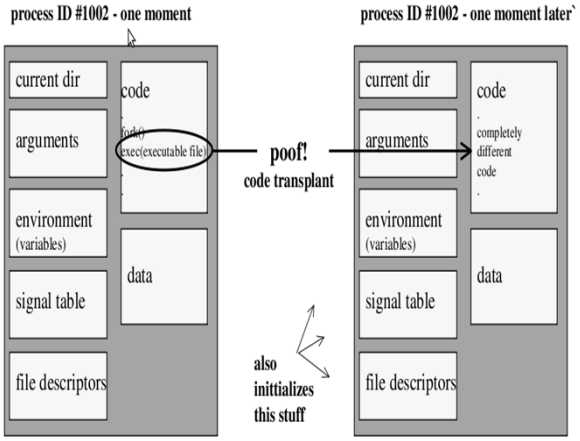
conditional, on whether parent or child

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

```
cihan@sdf-1:/media/My Book__/Linux-28.02.2009/networkProgramming/course/2> ./Fork3
(6872) Parent does something...
(6873) Child can do some stuff
(6872) Parent do completely different stuff
cihan@sdf-1:/media/My Book__/Linux-28.02.2009/networkProgramming/course/2>
```

By exec() function



exec() example

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

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

int main ( void )  {

        printf("Parent does stuff and then calls fork...\n");

        if(fork())  { // Parent
                printf("... parent do something completely different\n");
        } else  {       // Child
                printf("Child runs an executable...\n");
                execl("/bin/ls","/bin/ls","-l","/etc/apache2/conf.d/",NULL);
        }

        exit(0);
}
```

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

```
cihan@sdf-1:/media/My Book__/Linux-28.02.2009/networkProgramming/course/2> ./Exec
Parent does stuff and then calls fork...
Child runs an executable...
total 16
-rw-r--r-- 1 root root 646 2008-12-03 12:14 apache2-manual.conf
-rw-r--r-- 1 root root 225 2008-12-04 09:24 gitweb.conf
-rw-r--r-- 1 root root 709 2008-12-03 12:48 mod_perl.conf
-rw-r--r-- 1 root root 451 2008-12-03 13:07 php5.conf
... parent do something completely different
cihan@sdf-1:/media/My Book__/Linux-28.02.2009/networkProgramming/course/2>
```

ls -l /etc/apache2/conf.d/

exec() Family of Functions

- int execl(const char *pathname, const char *arg0, …);
- int execv(const char *pathname, char *const argv[]);
- int execle(const char *pathname, const char *arg0, ..., 0, char *const envp[]);
- int execlp(const char *filename, const char *arg0, …);
- int execvp(const char *filename, char *const argv[]);
- int execve(const char *pathname, char *const argv[], char *const envp[]);

For example...

- Shell is running
- You type “ls” and Enter
- Shell is parent, spawns ls as child

A simple shell example

```
int main(void)
{
    char inputBuffer[MAX_LINE]; /* buffer to hold the command entered */
    int background;             /* equals 1 if a command is followed by '&' */
    char *args[MAX_LINE/2+1];/* command line (of 80) has max of 40 arguments */


    while (1){            /* Program terminates normally inside setup */
        background = 0;
        printf("COMMAND->");
        fflush(0);
        setup(inputBuffer, args, &background);       /* get next command */

        /* the steps are:
         (1) fork a child process using fork()
         (2) the child process will invoke execvp()
         (3) if background == 0, the parent will wait,
                otherwise returns to the setup() function. */
    }
}
~
~
"shell.c" 87L, 2813C written                                  80,0-1        Bot
```

Process Termination

- void exit (int status);
  - exits a process
  - normally return with status 0
- int atexit (void (*function)(void));
  - registers function to be executed on exit

Process Termination Example

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

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

void parentCleaner ( void );

int main ( void )  {

        if(fork())  { // parent process
                atexit(parentCleaner);
                printf("this is parent %i\n",getpid());
        } else  {       // child process
                printf("this is child %i\n",getpid());
        }
        exit(0);
}

void parentCleaner ( void )  {

        printf("cleaning up parent...\n");

}
~
~
~
"exit.c" 23L, 373C written
```

File Edit View Terminal Tabs H

```
cihan@sdf-1:/media/My Bo
this is parent 6540
cleaning up parent...
this is child 6539
cihan@sdf-1:/media/My Bo
```

output

Zombie Processes

- When process terminates, still consumes system resources
  - Various tables maintained by OS
- Called a zombie
  - Living corpse, half alive, half dead

Zombie Processes

- Reaping
  - Performed by parent on terminated child
  - Parent is given exit status information
  - Kernel discards process
- What if parent does not reap ?
  - if any parent terminates without reaping a child, then childwill be reaped by init process
  - so, only need explicit reaping in long-running processes

Zombie : non-terminating parent

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

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

int main ( void ) {

        if(fork()) {  // Parent
                printf("Running parent, pid : %i\n",getpid());
                while(1);
        } else {    // Child
                printf("Terminating child, pid : %i\n", getpid());
                exit(0);
        }

        exit (0);
}
```

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

```
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ./Zombie &
Terminating child, pid : 7423
Running parent, pid : 7422
[1] 7422
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ps
  PID TTY          TIME CMD
 7197 pts/3    00:00:00 bash
 7422 pts/3    00:00:02 Zombie
 7423 pts/3    00:00:00 Zombie <defunct>
 7427 pts/3    00:00:00 ps
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2>
```

Ps shows chil as defunct

Killing parent allows child to be reaped by shell

Zombie : non-terminating child

```
cihan@sdf-1:...ogramming/course/2
File Edit View Terminal Tabs Help
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main ( void ) {

        if(fork())  {  // Parent
                printf("Running parent, pid : %i\n",getpid());
                exit(0);
        } else  {    // Child
                printf("Terminating child, pid : %i\n", getpid());
                while(1);
        }

        exit (0);
}
```

Child process still active,
even though parent has terminated

```
cihan@sdf-1:...ogramming/course/2
File Edit View Terminal Tabs Help
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ./Zombie2
Terminating child, pid : 7911
Running parent, pid : 7910
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ps
  PID TTY          TIME CMD
 7197 pts/3    00:00:00 bash
 7911 pts/3    00:00:01 Zombie2
 7918 pts/3    00:00:00 ps
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2>
```

Must be killed explicitly, or
Else will keep running indefinitely

Synchronizing with child

- int wait(int *child_status)
  - suspends current process until one of its children terminates
  - return value is the pid of the child process that terminated
  - If the child has already terminated, then wait returns its pid immediately
  - If child_status != NULL, then the object it points to will be set to a status indicating why the child process terminated

wait() example

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

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

#define numOfChilds 5
int main ( void )  {

        int i;
        int child_status;
        pid_t pid[numOfChilds];
        pid_t wpid;

        for (i = 0; i < numOfChilds; i++)  {
                if ((pid[i] = fork()) == 0)  {
                        exit(100+i);            // create & exit child
                }
        }

        for (i = 0; i < numOfChilds; i++) {
                wpid = wait(&child_status);     // wait for child
                if (WIFEXITED(child_status))  { // check exit status
                        printf("Child %d terminated with exit status %d\n",wpid, WEXITSTATUS(child_status));
                } else {
                        printf("Child %d terminate abnormally\n", wpid);
                }
        }
        exit(0);
}
```

cihan@sdf-1:...ogramming/course/2

File Edit View Terminal Tabs Help

```
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ./Wait
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ./Wait
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ./Wait
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2> ./Wait
Child 8958 terminated with exit status 100
Child 8959 terminated with exit status 101
Child 8960 terminated with exit status 102
Child 8961 terminated with exit status 103
Child 8962 terminated with exit status 104
cihan@sdf-1:/media/My Book/Linux-28.02.2009/networkProgramming/course/2>
```

references

- man pages
- http://www.cs.princeton.edu/courses/archive/fall01/cs217/slides/process.pdf
- http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15213-f08/www/lectures/lecture-11.pdf
- http://csapp.cs.cmu.edu
- http://homepage.smc.edu/morgan_david/linux/a12-processes.pdf