Processes

ECE595, Jan 18

Y. Charlie Hu

Review

- OS is a resource manager
- OS presents an extended machine
- OS is a “giant interrupt handler”
- System calls
- Interrupt

[lec2] Roadmap for ECE595 lectures

Introduction to OS components
Individual components
- Process management
- Memory management
- File management
- Secondary-storage management

- (Device management)
- (shell)
- (Networking)

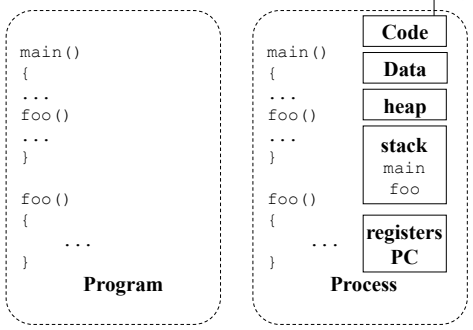
Hardware support for OS interspersed (before related topics)

Users, Programs, Processes

- Users have accounts on the system
- Users launch programs
  - Can many users launch the same program?
  - Can one user launch many instances of the same program?

→ A process is an “instance” of a program

Program vs. Process



Program as A Process Collection, or a Program Collection

- Firefox (output of “ps ux | grep firefox”)
  - [~]$ ps ux | grep firefox
  - ychu 529 0.0 0.2 326756 9836 ? Sl 2012 1:43 /usr/lib64/xulrunner-2/plugin-container /usr/lib64/mozilla/plugins-wrapped/nswrapper_32_64.libflashplayer.so -greomni / usr/lib64/xulrunner-2/omni.ja -appomni /usr/lib64/firefox/omni.ja 25752 plugin
  - ychu 25752 3.9 6.6 1240212 250328 ? Sl 2012 8039:58 /usr/lib64/firefox/firefox
- gcc (via “gcc –pipe –v”)
  - /usr/libexec/cpp |
  - /usr/libexec/cc1 |
  - /usr/libexec/as, followed by
  - /usr/libexec/elf/ld

Users, Programs, Processes

- Users have accounts on the system
- Users launch programs
  - Can many users launch same program?
  - Can one user launch many instances of the same program?

→ A process is an “instance” of a program
→ A “program” can be a set of programs
→ A “program” can be a set of processes

So What Is A Process? (1)

- It’s one instance of a “program”
- Any relationship between two instances?

What Does This Program Do?

```
int myval;

int main(int argc, char *argv[])
{
   myval = atoi(argv[1]);
   while (1)
        printf("myval is %d, loc 0x%lx\n",
                myval, (long) &myval);
}
```


Instances of Programs

- The address was always the same!
- The values were different!
- Implications:
  - Do instances think they're using the same address?
  - Are they seeing each other?
- Conclusion: addresses are not absolute!
- What are the benefits?

So What Is A Process? (2)

- It is one instance of a "program"
- It is separate from other instances

Sequential execution of each process

- No concurrency inside a process
- Everything happens sequentially
- Often with interleaved CPU/IO operations

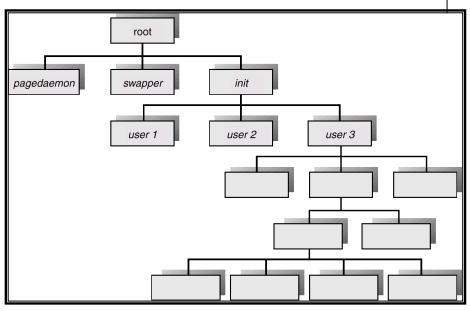
Concurrent Processes

- Processes in a system can execute concurrently (multitasking)
  - Concurrently vs. simultaneously
- Motivations for allowing concurrent execution
  - Physical resource sharing (system utilization)
  - Computational speedup – with several CPUs
  - Modularity/safety (firefox)
  - Convenience (desktop: firefox, xclock, emacs, xterm)

Process Creation

- Who created process cpp?
- Who created cc1?
- Who created gcc?
- Who created shell?

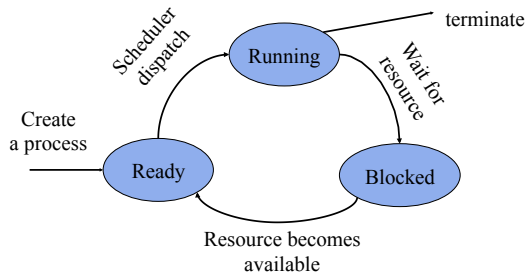
Processes Tree on a UNIX System



So What Is A Process? (3)

- It's one instance of a "program"
- It's separate from other instances
- It can start ("launch") other processes
- It can be launched by another process

Life cycle of a process: Process State Transition



Kernel data structure: Process Control Block (Process Table)

- Process management info
  - State (ready, running, blocked)
  - PC & Registers, parents, etc
  - CPU scheduling info (priorities, etc.)
- Memory management info
  - Segments, page table, stats, etc
- I/O and file management
  - Communication ports, directories, file descriptors, etc.

Process Id

- Every process has an Id – process Id
- Does a program know its process Id?
- When a program is running, how does the process know its id?

Process Creation/Termination

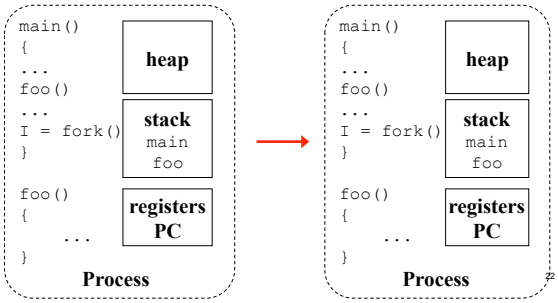
- Who should *actually* create/terminate processes? (bring program to life)
  - Who manages processes?
- But user process decides when and what
- So what should OS provide user process?
- What should the semantics of that be?

Process Creation – interface design options

- The system call creates a child process
- Execution possibilities?
  - Parent and child execute concurrently
  - Parent waits till child finishes
- Address space possibilities?
  - Child has a new program loaded into it
  - Child duplicate of parent (code and data)

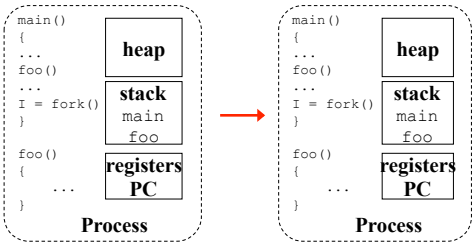
Process Creation –- UNIX interfaces (1)

- *fork()* system call creates a duplicate of the original process



Process Creation –- UNIX interfaces (2)

- *fork()* system call creates a duplicate of the original process
  - What is the major benefit?
  - How to disambiguate who is who?



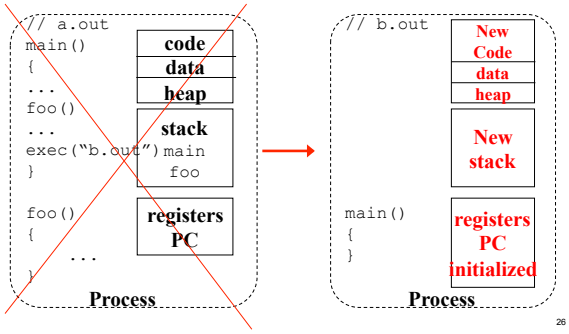
C program forking a new process

```
#include <stdio.h>

void main()
{
  int pid; int was = 3;
  pid = fork(); /* fork another process */

  if (pid < 0) { /* error occurred */        exit(-1); }
  else if (pid == 0) { /* child process */
    printf("child: was = %d", was);
  else { /* pid> 0; parent process ; pid is child process's */
    printf("parent: child process id = %d", pid);
    wait(NULL);    exit(0);
  }
}
```

Process Creation –- UNIX interfaces (3)

- *fork()* system call creates a duplicate of the original process
  - What is the major benefit?
  - How to disambiguate who is who?
- *exec()* system call used after a *fork* to replace the process' code/address space with a new program
  - Important: BOTH code and data, i.e., the whole address space is replaced !

exec("b.out")



Afterwards, only one thing about the process was kept, which is?

Process Creation – UNIX interfaces (4)

UNIX has 4 system calls:

- fork – create a copy of this process
  - Clone would have been a better name!
- exec – replace this process with this program
- wait – wait for child process to finish
- kill – (potentially) end a running process

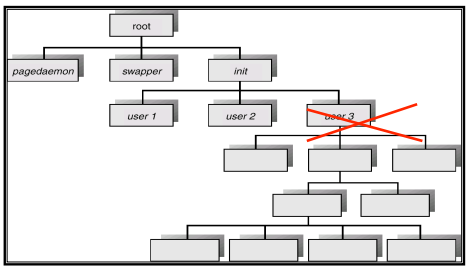
C program forking a new process

```
#include <stdio.h>
void main()
{
 int pid;       int was = 3;
 pid = fork(); /* fork another process */

 if (pid == 0) { /* child process */
   sleep(2);  printf("child: was = %d", was);
   execlp("/bin/ls", "ls", NULL);}
 else { /* pid > 0; parent process */
   was = 4;
   printf("parent: child process id = %d;   was=%d ", pid, was);
   wait(NULL);     exit(0);
 }
}
```

C program forking a new process

```
#include <stdio.h>

void main()
{
 int pid;       int was = 3;
 pid = fork(); /* fork another process */

 if (pid == 0) { /* child process */
   sleep(2); was=9;  printf("child: was = %d", was);
   execlp("/bin/ls", "ls", NULL);       was = 10;
   printf("It's me, your child was = %d". was);}
 else { /* pid > 0; parent process */
   was = 4;
   printf("parent: child process id %d was=%d ", pid, was);
   wait(NULL);     exit(0);
 }
}
```


Processes Tree on a UNIX System



What happens when a parent process disappears?
all child processes are killed by the OS, or
all child processes reset init as their parent

Summary

- What is a process?
- PCB
  - Process management info
  - Memory management info
  - I/O, file management info
- Process state transition
- Concurrent processes
- Process creation and termination
- Quiz: let us implement shell!

[lec2] Roadmap for ECE595 lectures

Introduction to OS components
Individual components
- Process management
- Memory management
- File management
- Secondary-storage management

- (Device management)
- (shell)
- (Networking)

Hardware support for OS interspersed (before relevant topics)

Command-interpreter system

- Interface between user and OS
  - (What about system calls?)
- Many commands are given to the OS by control statements which deal with:
  - process creation and management (a.out, kill –9)
  - secondary-storage management (partition disk)
  - file system
    - access (ls, cat)
    - protection (chmod 700 *, umask 077)
  - networking (telnet, ssh)
  - monioring (ps, top, vmstat, netstat)

Command-interpreter system (cont.)

- The program that reads and interprets control statements is called the *command-line interpreter* (or the *shell* in UNIX)
- What is the shell program's structure like?
- Two prevalent approaches to the interface
  - moused-based windows-and-menus (Mac, Windows)
  - command line (UNIX)

Let us implement Shell!

Reading

- Chapters 3, 5