Behavior of Linux Kernel and Process Manipulation

Azzam Mourad

www.encs.concordia.ca/~mourad

mourad@encs.concordia.ca

COEN 346

Agenda

- Role of the kernel
- Kernel State in the /proc system directory
- /proc directory organization
- How to access information in /proc
- Assignment 2 (part 1)

Linux Kernel

- Kernel is the central component of an OS
- Its responsibilities include managing the processes and system's resources
- Communication between hardware and software components
- Almost all of memory, file, and device management is implemented in it
- Control access to physical devices and schedule when and how processes interact with these devices

Linux Kernel

- OS uses various data structures
- The values that determine the state are stored in its data structures
- We can study the behavior of an OS system by observing values in the kernel data structures

Kernel State in the /proc system directory

- Kernel data structures may be manipulated by widely disparate parts of the kernel
- This makes it difficult to locate specific data structures and then determine the OS state
- Linux provides the /proc file system
- Data structures needed to inspect the kernel state are saved in this directory

/proc organization

- The /proc/ directory contains a hierarchy of special files which represent the current state of the kernel
- Allows applications and users to peer into the kernel's view of the system
- Information about system hardware and processes can be found
- Information are always up to date

/proc organization

- All date are stored as files
- The type of files in /proc are of type virtual (different than text or binary)
- This is why we refer to /proc as virtual file system
- Most of these virtual files are listed as 0 bytes in size, although they contain large amount of information

/proc organization

- Most of the time and date settings on these virtual files reflect the current date and time
- This means that they are constantly updated
- For organizational purposes, files containing information on a similar topic are grouped into virtual directories and sub-directories
- Process directories contain information about each running process on the system

How to access info in /proc

- By using the cat, more, or less commands on files within the /proc/ directory, you can view the content of the virtual files

- CPU information

```
processor: 0
vendor_id: AuthenticAMD
cpu family: 5
model: 9
model name: AMD-K6(tm) 3D+ Processor
stepping: 1
cpu MHz: 400.919
cache size: 256 KB
…
```

How to access info in /proc

- Those files are read just like ordinary ASCII files
- You can use stdio or iostream libraries routines such as fgets(), fscanf()…
- After reading the date from the file, parse the string to extract the information

Assignment 2 (Part 1)

- Write a C/C++ program, called *Asg2i.cpp or Asg2i.c*, which reports the behavior of the Linux kernel.
- Your program should print the following value on the standard output:
  - Processor type
  - Kernel version
  - Memory status
  - Any three information about the kernel
  - Any three information about a current running process

Process Manipulation Agenda

- How to create and terminate a process
- Relation between a parent and child process
- The use of fork() and exec() functions
- Assignment 2 (part 2)

Assignment 2 (part 2 – a)

- Write a C/C++ program, called *Asg2iia.cpp or Asg2iia.c* that does the following:
  - Executes as a parent process, which occurs naturally.
  - The parent process must output the following statement: "*Parent process is running and about to fork to a child process*".
  - The parent process) must then create a child process (using *fork()*).
  - The child will simply print out to the standard output the following statement: *"I am the child process"*.
  - You are NOT allowed to use the *exec* calls in this part.
  - That is, you must make sure that the child will still run the proper code to perform what it needs to do without the executions of any of the "*exec*" calls.

Assignment 2 (part 2 – a)

- Once the child starts, the parent must wait for the child to die before it continues.
- Output:

Parent process is running and about to fork to a child process

I am the child process

Parent acknowledges child termination Parent will terminate now

Assignment 2 (part 2 – b)

- Write a C/C++ program, called "*outsider.cpp*" or "*outsider .c*" that outputs the following statement: "Outsider program is running.
- Write a C/C++ program called *Asg2iib.cpp or Asg2iib.c*, which is similar to the one you created in Part II-A above, with the following exceptions:
  - The child process must execute the code of the *Outsider* program using the *exec* system call
- Output:

Parent process is running and about to fork to a child process
Outsider program is running. Time now is Mon Jan 29 01:16:26 EST 2007
Parent acknowledges child termination Parent will terminate now

Process Manipulation

- A process is created for you program when you run it from a shell
- This is the parent process
- You can create child processes inside the program using the fork() command