CS354: Machine Organization and Programming

Lecture 27
Wednesday the November 04th 2015

Section 2
Instructor: Leo Arulraj

Class Announcements

Programming Assignment 3 was due by 9 AM today. You can submit it upto 48 hours after the deadline with penalties.

Programming Assignment 4 has been released and it is due by 11/25 (Wednesday). This assignment involves coding and the theme is signal handlers. Start early! (but after the midterm)

Lecture Overview

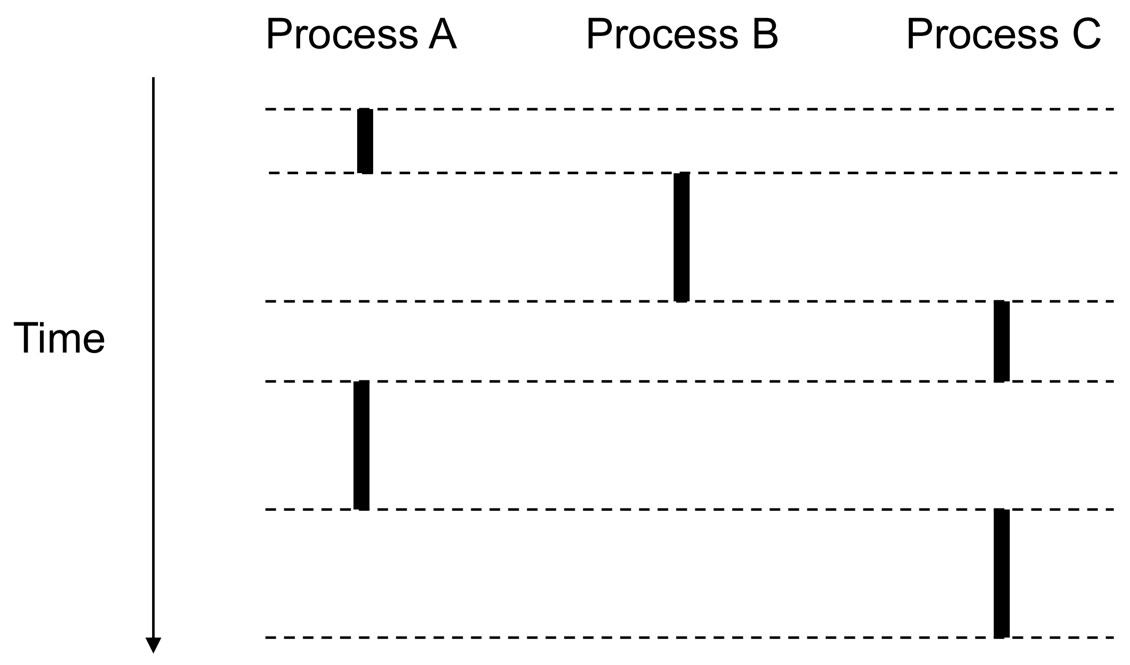
1. Intro to Processes
2. Signal Handlers

Processes

1. Process: an instance of a program in execution
2. Give the illusion that our program is the only one currently running in the system
3. Process Context consists of state including: Virtual Memory Layout, CPU registers, file descriptors, environment variables etc.
4. Key abstraction provided by a process:
    a. Independent logical control flow
    b. Private address space

Logical Control Flow

The single physical control flow of CPU is partitioned into logical control flows of several processes.



Concurrent and Parallel Flows

1. A logical flow whose execution overlaps in time with another flow is called a concurrent flow.
2. E.g. A & B are concurrent in previous slide, A & C are also concurrent while B & C are not concurrent.
3. Parallel flows: A subset of concurrent flows where the individual flows run on multiple cores or machines in parallel.

Private Address Space

1. A process provides each program the illusion that it has exclusive use of the system's address space through virtual memory.

2. This concept of private address space per process makes writing programs much easier rather than dealing with physical memory addresses. (e.g. frees the programmer from managing the physical memory resources)

Privileged Mode

1. User Mode: Cannot execute privileged instructions like one that halts the CPU. Also cannot access kernel area of address space.

2. Kernel Mode (Privileged/Supervisor Mode): Can execute any instruction and access any memory location.

Process runs application code in user mode and switches to kernel mode only via an exception like interrupt, system call etc.

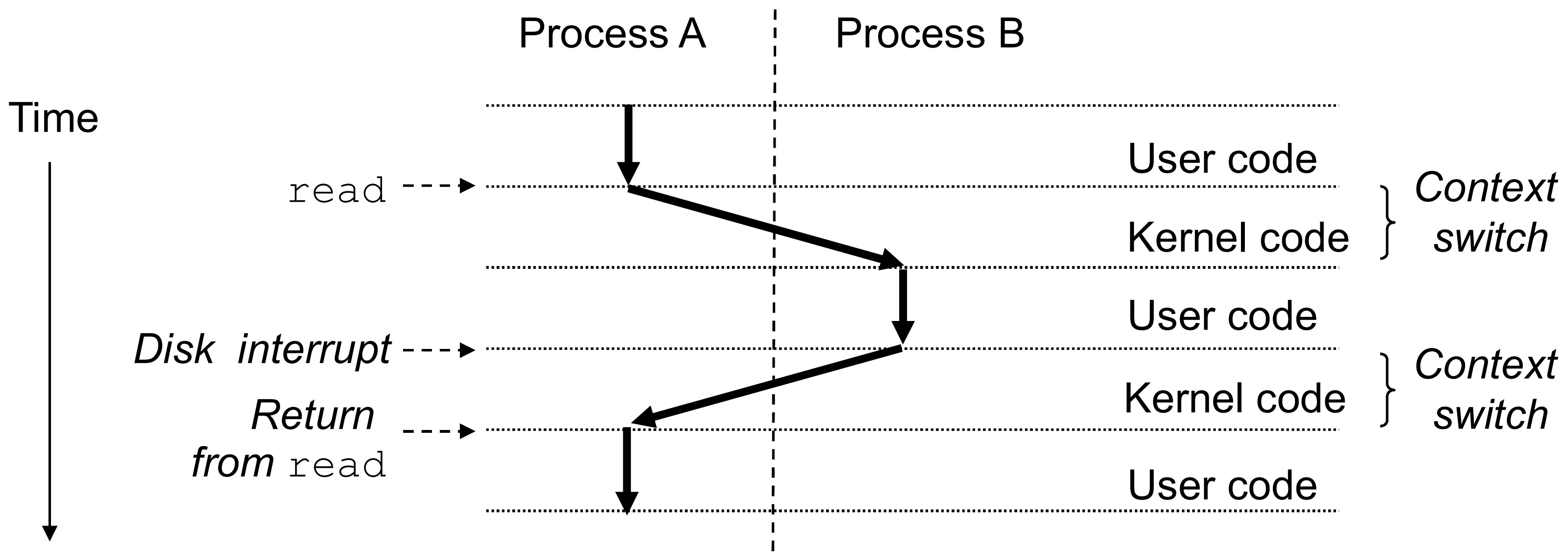
Process Context Switch

Operating System pre-empts the process currently executing on the CPU and schedules another process through context switch.

Context switch involves:

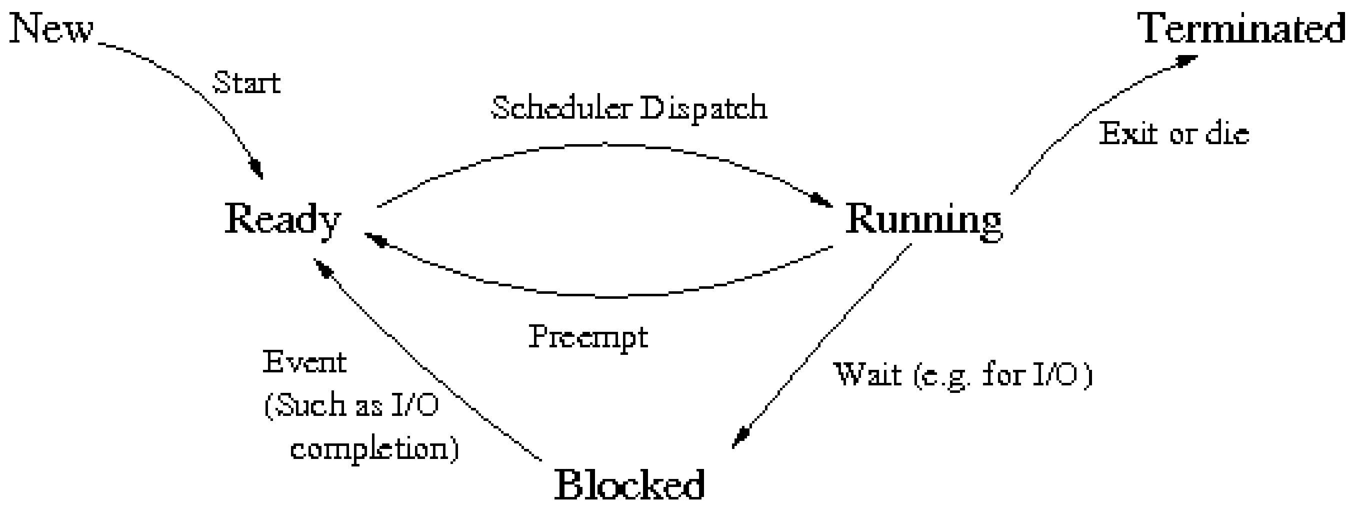
1) Saving the context of the current process.
2) Restoring the saved context of some previously pre-empted process
3) Passing control to this newly restored process.

Process Context Switch



Process States

Processes move through different states during their life cycle.



Process Control

Obtaining Process IDs

```
pid_t getpid(void);
pid_t getppid(void);
```

Terminating a Process:

```
void exit(int status);
```

Process Control

Creating a new process:

pid_t fork(void);

- Call once, return twice
- Concurrent execution
- Duplicate but separate address spaces
- Shared files

Process Control

Reaping Child Processes:

OS defers removing a terminated process until its parent process reaps it.

A terminated process that has not yet been reaped is called a zombie.

The init process with pid 1 that is created during system initialization reaps any unreaped child processes if its parent process dies without reaping the children.

Process Control

Though zombie processes are not running they consume memory resources and it is good practice to reap the child processes.

Parent can wait for child to terminate by:

pid_t waitpid(pid_t pid, int *status, int options);
If pid > 0, wait for specific child process
If pid = -1, then wait for all child prcesses

If a child process has already terminated then waitpid returns immediately.

man waitpid

Process Control

Process can sleep for a period of time using:

```
unsigned int sleep(unsigned int secs);
```

Process can pause until a signal is received using:

```
int pause(void);
```

Process Control

Loading and Running Programs

int execve(const char* filename, const char *argv[], const char *envp[]);

Writing a **simple shell program** using fork() and execve().