ECS 150 – Operating Systems

Matt Roper

March 29th, 2007

Operating Systems – Some Examples

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
    - Linux

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows
  - Mac OS

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows
  - Mac OS
  - FreeBSD, NetBSD, OpenBSD, . . .

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows
  - Mac OS
  - FreeBSD, NetBSD, OpenBSD, . . .
  - Solaris

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows
  - Mac OS
  - FreeBSD, NetBSD, OpenBSD, . . .
  - Solaris
  - AIX

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows
  - Mac OS
  - FreeBSD, NetBSD, OpenBSD, . . .
  - Solaris
  - AIX
  - Minix

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows
  - Mac OS
  - FreeBSD, NetBSD, OpenBSD, . . .
  - Solaris
  - AIX
  - Minix
  - DOS
  - . . .

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
  - Linux
  - Windows
  - Mac OS
  - FreeBSD, NetBSD, OpenBSD, . . .
  - Solaris
  - AIX
  - Minix
  - DOS
  - . . .
- Embedded or Realtime Operating Systems
  - QNX

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
    - Linux
    - Windows
    - Mac OS
    - FreeBSD, NetBSD, OpenBSD, . . .
    - Solaris
    - AIX
    - Minix
    - DOS
    - . . .
- Embedded or Realtime Operating Systems
    - QNX
    - RTLinux

Operating Systems – Some Examples

- Desktop/Workstation/Server Operating Systems
    - Linux
    - Windows
    - Mac OS
    - FreeBSD, NetBSD, OpenBSD, . . .
    - Solaris
    - AIX
    - Minix
    - DOS
    - . . .
- Embedded or Realtime Operating Systems
    - QNX
    - RTLinux
    - TRON

Purposes of an Operating System

So we can name all these different operating systems. But why do they exist? What is their underlying purpose?

Purposes of an Operating System

So we can name all these different operating systems. But why do they exist? What is their underlying purpose?

An operating system has two primary responsibilities:

- Managing the computer's resources (processor, memory, I/O devices)
- Providing a standard interface for users and user software

Resource Management: Examples

- Time-slicing of many concurrent applications onto a limited number of physical processors.

Resource Management: Examples

- Time-slicing of many concurrent applications onto a limited number of physical processors.
- Allocation of physical memory to several applications.

Resource Management: Examples

- Time-slicing of many concurrent applications onto a limited number of physical processors.
- Allocation of physical memory to several applications.
- Organization and bookkeeping of a disk file system.

Standard Interface: Examples

The operating system hides many low level details from application programmers.

Standard Interface: Examples

The operating system hides many low level details from application programmers.

Suppose you're writing a program that simply opens a file and reads some text. Simple. . .

Standard Interface: Examples

The operating system hides many low level details from application programmers.

Suppose you're writing a program that simply opens a file and reads some text. Simple. . . now suppose that file happens to reside on a floppy disk.

Standard Interface: Examples

The operating system hides many low level details from application programmers.

Suppose you're writing a program that simply opens a file and reads some text. Simple. . . now suppose that file happens to reside on a floppy disk.

- the hardware interface for a floppy disk drive controller supports 16 different commands (reading/writing data, moving the disk head, etc.)

Standard Interface: Examples

The operating system hides many low level details from application programmers.

Suppose you're writing a program that simply opens a file and reads some text. Simple. . . now suppose that file happens to reside on a floppy disk.

- the hardware interface for a floppy disk drive controller supports 16 different commands (reading/writing data, moving the disk head, etc.)
- each one of those commands requires loading several bytes into a register

Standard Interface: Examples

The operating system hides many low level details from application programmers.

Suppose you're writing a program that simply opens a file and reads some text. Simple. . . now suppose that file happens to reside on a floppy disk.

- the hardware interface for a floppy disk drive controller supports 16 different commands (reading/writing data, moving the disk head, etc.)
- each one of those commands requires loading several bytes into a register
- each command requires *13* parameters each

Standard Interface: Examples

The operating system hides many low level details from application programmers.

Suppose you're writing a program that simply opens a file and reads some text. Simple. . . now suppose that file happens to reside on a floppy disk.

- the hardware interface for a floppy disk drive controller supports 16 different commands (reading/writing data, moving the disk head, etc.)
- each one of those commands requires loading several bytes into a register
- each command requires *13* parameters each

Aren't you glad the OS takes care of these details for you?

Consider the following simple C program:

```
#include <stdio.h>
#include <time.h>

int main(void) {
    time_t t = time(NULL);

    printf("Hello world!  It is %s\n", ctime(&t));

    return 0;
}
```

and its output:

```
Hello world!  It is Thu Mar 26 09:16:37 2007
```

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

What operations does the operating system perform from the time we hit `ENTER` in our shell until the program is complete?

- find the executable on the hard drive's file system
- check permission bits (are you allowed to run this program?)
- load the executable from disk into memory
- allocate physical memory for the program's variables and setup a virtual address space
- temporarily stop the shell program and allow the "hello world" program to run
- retrieve the value of the hardware clock
- print characters to the screen
- when the program finishes, free its memory for other programs to use
- allow the shell process to run again

Goals for this Term

- Become familiar with what happens internally in an Operating System
- Understand the basic design principles of various OS subsystems (process scheduling, memory management, file system I/O, etc.)
- Gain the skills to modify a real-world OS (FreeBSD 5.4)

Prerequisites

Prerequisites will not be strictly enforced, but I expect you to be familiar with the following concepts:

- C programming (pointers, arrays, structures, malloc(), etc.) – ECS 30
- Data structures, especially linked lists – ECS 40 & 110
- How parameters are pushed onto the stack when application function calls are made – ECS 50
- General computer architecture (registers, ALU, etc.) – ECS 154A

Brief Course Outline

- System calls and system programming
- Process scheduling and management
- Memory management
- IO & Filesystems