What does an Operating System do?

Jonathan Geisler

February 5, 2007

Taylor University
CSS

Perspectives on an OS

There are three perspectives on an OS:

1. The *user* is concerned with the services it provides.

Perspectives on an OS

There are three perspectives on an OS:

1. The *user* is concerned with the services it provides.
2. The *programmer* is concerned with the interfaces it provides.

Perspectives on an OS

There are three perspectives on an OS:

1. The *user* is concerned with the services it provides.
2. The *programmer* is concerned with the interfaces it provides.
3. The *designer* is concerned with the internal components and their interconnections.

Services Provided

- User interface (CLI vs. GUI)
- Run programs
- I/O
- Access to files
- Networking
- Error handling
- Protection & Security
- Accounting
- Managing resources

Interfaces Provided (i.e., system calls)

This is how a program asks the OS to perform a specific service. Think back to chapter one, how do they work? Notice how many calls were in the simple example from the book. This requires them to be done very efficiently.

Typical system calls

This is a list of the type of system calls you would expect to find in most OSes:

- end
- abort
- create process
- get/set process attributes
- wait
- signal
- date/time
- open
- read
- write
- get/set file attributes
- close
- acquire resource
- create connection

Interesting system calls

Every operating system has its share of common system calls. Here are a couple unique ones that you may not have thought about before:

- Trace can allow profiling programs to run or `strace` on Linux

Interesting system calls

Every operating system has its share of common system calls. Here are a couple unique ones that you may not have thought about before:

- Trace can allow profiling programs to run or `strace` on Linux
- Step allows debugging programs to work

Interesting system calls

Every operating system has its share of common system calls. Here are a couple unique ones that you may not have thought about before:

- Trace can allow profiling programs to run or `strace` on Linux
- Step allows debugging programs to work
- Terminate and Stay Resident allows MS-DOS to fake multitasking

Components and Interconnections

Components

- Management

Interconnection types

Components and Interconnections

Components

- Management
  - Process

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity
  - Cons: dependencies & efficiency

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity
  - Cons: dependencies & efficiency
- Microkernel (Mach, WinNT, Mac OS X, etc.)

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity
  - Cons: dependencies & efficiency
- Microkernel (Mach, WinNT, Mac OS X, etc.)
  - Pros: extendibility, portability, & security

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity
  - Cons: dependencies & efficiency
- Microkernel (Mach, WinNT, Mac OS X, etc.)
  - Pros: extendibility, portability, & security
  - Cons: efficiency

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity
  - Cons: dependencies & efficiency
- Microkernel (Mach, WinNT, Mac OS X, etc.)
  - Pros: extendibility, portability, & security
  - Cons: efficiency
- Whatever works (Linux, WinXP, etc.)

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity
  - Cons: dependencies & efficiency
- Microkernel (Mach, WinNT, Mac OS X, etc.)
  - Pros: extendibility, portability, & security
  - Cons: efficiency
- Whatever works (Linux, WinXP, etc.)
  - Pro: run-time efficiency

Components and Interconnections

Components

- Management
  - Process
  - Memory
  - File
  - I/O
  - Storage
- Networking
- Protection & Security
- Command Interpreter

Interconnection types

- Layered (OS/2)
  - Pro: modularity
  - Cons: dependencies & efficiency
- Microkernel (Mach, WinNT, Mac OS X, etc.)
  - Pros: extendibility, portability, & security
  - Cons: efficiency
- Whatever works (Linux, WinXP, etc.)
  - Pro: run-time efficiency
  - Cons: modularity & extendibility

Example system call
Starting a new process

This is the bulk of your second lab! There are two distinct ways to start a new process:

1. "MS-DOS" way

Example system call
Starting a new process

This is the bulk of your second lab! There are two distinct ways to start a new process:

1. "MS-DOS" way
   - Like a function call in that the called process returns directly to the calling process

Example system call
Starting a new process

This is the bulk of your second lab! There are two distinct ways to start a new process:

1. "MS-DOS" way
   - Like a function call in that the called process returns directly to the calling process
   - Non-multitasking

Example system call
Starting a new process

This is the bulk of your second lab! There are two distinct ways to start a new process:

1. "MS-DOS" way
   - Like a function call in that the called process returns directly to the calling process
   - Non-multitasking
2. "UNIX" way

Example system call
Starting a new process

This is the bulk of your second lab! There are two distinct ways to start a new process:

1. "MS-DOS" way
   - Like a function call in that the called process returns directly to the calling process
   - Non-multitasking
2. "UNIX" way
   - Start a new process and let it set up communication channels with its parent (if it feels like it)

Example system call
Starting a new process

This is the bulk of your second lab! There are two distinct ways to start a new process:

1. "MS-DOS" way
   - Like a function call in that the called process returns directly to the calling process
   - Non-multitasking
2. "UNIX" way
   - Start a new process and let it set up communication channels with its parent (if it feels like it)
   - Both starter and started processes run concurrently

Separating mechanism & policy

The book states that these two should be separated whenever possible. That sounds like a good idea, but how does that work? Lets use the process scheduler for a concrete example.

Virtual machines

- IBM
  - Initial leaders in this field developing VM
  - Runs Linux this way on big mainframes
- VMWare
- Xen
- bochs
- User Mode Linux
- Java