

OS Interfaces, Services, and Structures

Hui Chen ^a

^aCUNY Brooklyn College

February 13, 2025

Outline

- 1 Objectives
- 2 OS Operations
- 3 OS Services
- 4 OS User Interface
- 5 OS Programmer Interface (System Calls)
- 6 OS Design and Structures
- 7 OS Debugging and Monitoring

Outline

- 1 Objectives
- 2 OS Operations
- 3 OS Services
- 4 OS User Interface
- 5 OS Programmer Interface (System Calls)
- 6 OS Design and Structures
- 7 OS Debugging and Monitoring

Objectives

- ▶ Identify services provided by an operating system
- ▶ Use system calls to access operating system services
- ▶ Use user interface to access operating systems
- ▶ Compare and contrast monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems.
- ▶ Apply tools for monitoring operating system performance.
- ▶ Design and implement kernel modules for interacting with a Linux kernel.

Outline

- 1 Objectives
- 2 OS Operations**
- 3 OS Services
- 4 OS User Interface
- 5 OS Programmer Interface (System Calls)
- 6 OS Design and Structures
- 7 OS Debugging and Monitoring

Interrupt Driven Operations

OSes are driven by interrupts (software and hardware)

- ▶ Hardware interrupt by one of the devices
- ▶ Software interrupt (exception or trap):
 - ▶ Software error (e.g., division by zero)
 - ▶ Request for operating system service (system calls)
 - ▶ Other process problems include infinite loop, processes modifying each other or the operating system

Dual-mode Operations

OSes operate in two modes (dual-mode operation) in order to allow OSes to protect themselves and other system components (from what?)

- ▶ User mode and kernel mode
 - ▶ Mode bit provided by hardware
 - ▶ Provides ability to distinguish when system is running user code or kernel code
 - ▶ Some instructions designated as privileged, only executable in kernel mode
 - ▶ System call (OS services provided by OS kernel) changes mode to kernel, return from call resets it to user
- ▶ Increasingly CPUs support multi-mode operations i.e. virtual machine manager (VMM) mode for guest VMs

Start-up of OSES (Bootstrap/Boot)

Bootstrap program is loaded at power-up or reboot

- ▶ Typically stored in ROM or EPROM, generally known as firmware
- ▶ Initializes all aspects of system
- ▶ Loads operating system kernel and starts execution

Let's do an exercise ...

Outline

- 1 Objectives
- 2 OS Operations
- 3 OS Services**
- 4 OS User Interface
- 5 OS Programmer Interface (System Calls)
- 6 OS Design and Structures
- 7 OS Debugging and Monitoring

Overview of OS Services

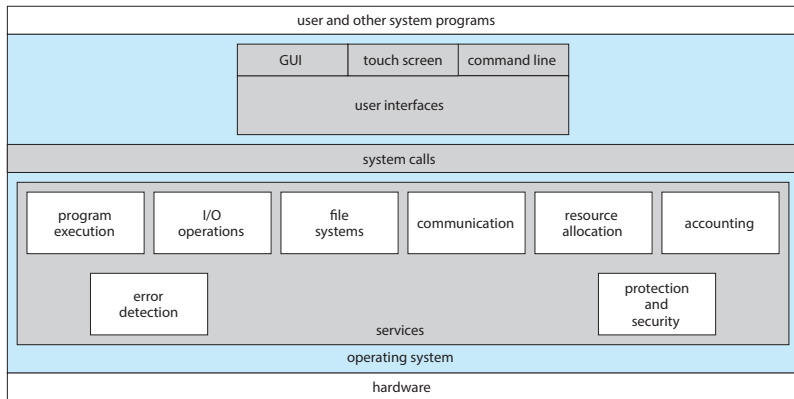


Figure: A view of operating system services¹.

¹Silberschatz, Galvin, and Gagne, *Operating system concepts*.

Outline

- 1 Objectives
- 2 OS Operations
- 3 OS Services
- 4 OS User Interface**
- 5 OS Programmer Interface (System Calls)
- 6 OS Design and Structures
- 7 OS Debugging and Monitoring

User Interface

- ▶ Command interpreters and command line interface
- ▶ Graphical user interface
- ▶ Touch-Screen interface
- ▶ Voice user interface
- ▶ Conversational user interface

Shell Scripts

- ▶ Shell Script. If a frequent task requires a set of command line steps, those steps can be recorded into a file, and that file can be run just like a program.
- ▶ The Shell Script program is not compiled into executable code but rather is interpreted by the command-line interface
- ▶ Example Command Line Interpreters with programmability.
 - ▶ Powershell on Windows²
 - ▶ Bash on UNIX³

²Jones and Hicks, *Learn Windows PowerShell 3 in a month of lunches*.

³Newham and Rosenblatt, *Learning the bash shell: Unix shell programming*.

Bash Shell Script Example

Let's count total number of lines of all `asm` files.

Choice of User Interface

Consider two perspectives, user perspective and system design perspective

- ▶ User perspective. Using a command-line or GUI interface can be one of personal preference.
- ▶ Design perspective. System designers may consider multiple factors.
 - ▶ Automation?
 - ▶ Resources?
 - ▶ Ease of use?
 - ▶ Human errors?

“Although the interface equipment and principles used for the Therac-25 are obsolete, there are still potential issues even with today’s more sophisticated interface tools. Sometimes making the interface easy to use conflicts with safety

...

One general design principle is that actions to get into or maintain a safe state should be easy to do. Actions that can lead to an unsafe state (hazard) should be hard to do.”^a

^aLeveson, “The Therac-25: 30 Years Later”.

Outline

- 1 Objectives
- 2 OS Operations
- 3 OS Services
- 4 OS User Interface
- 5 OS Programmer Interface (System Calls)**
- 6 OS Design and Structures
- 7 OS Debugging and Monitoring

System Calls

- ▶ System calls are the interface to the services made available by an operating system.
- ▶ These calls are generally available as functions written mostly in C and C++ and sometimes in an assembly language or using assembly-language instructions
 - ▶ For certain low-level tasks, e.g., tasks where hardware must be accessed directly may have to be written using assembly-language instructions.

System Call Example

Let's take a look at the following example⁴

```
#include <unistd.h>

int main(int argc, char *argv[]) {
    _exit(0);
}
```

Compile and disassemble⁵

```
gcc -static-libgcc -static-pie -l:libc.a exitex.c -o exitex
objdump --disassemble exitex > exitex.asm
```

Then examine exitex.asm (look for `_exit` in exitex.asm)

⁴[X86 Assembly/Interfacing with Linux](#).

⁵Tested on Debian Linux 10 and gcc 8.3.0

Invoking System Calls

- ▶ System calls occur in different ways, depending on the computer in use.
 - ▶ For example, on Linux x86 systems, make a system call by calling interrupt 0x80 using the `int 0x80` command or by issuing the `sysenter` instruction.
- ▶ We need to pass the identity of the desired system call and often additional information to the OS.
 - ▶ For example, to get input, we may need to specify the file or device to use as the source, as well as the address and length of the memory buffer into which the input should be read.
 - ▶ On Linux x86 systems, we pass parameters by setting the general purpose registers as following:

Syscall #	Param 1	Param 2	Param 3	Param 4	Param 5	Param 6
eax	ebx	ecx	edx	esi	edi	ebp
Return value						
eax						

Passing Parameters to OS for System Calls

There are 3 general methods to pass parameters to the operating system.

- ▶ (Register method) Passing the parameters in registers.
- ▶ (Block method) Passing the parameters stored in a block or a table in memory, and passing the address of the block or the table as a parameter in a register.
 - ▶ Often using the combination of the two.
 - ▶ For example, Linux uses this approach. If 5 or fewer parameters, use the register method; otherwise, the block method.
- ▶ (Stack method) Passing parameters via stack. The program pushes the parameters in a stack, and the OS pops the parameters off the stack.

Application Programming Interface

```
_exit(0);
```

- ▶ Application developers design programs according to an application programming interface (API).
- ▶ The API specifies a set of functions that are available to an application programmer.
- ▶ Example APIs.
 - ▶ Windows API, POSIX API, and Java API
- ▶ A programmer accesses an API via a library of code provided by the operating system.
 - ▶ Example. The libc library in UNIX and Linux.
- ▶ The functions that make up an API typically invoke the actual system calls on behalf of the application programmer.
 - ▶ Windows API function `CreateProcess()` invokes the `NTCreateProcess()` system call in the Windows kernel.
 - ▶ libc API function `open()` invokes the `open()` system call in the Linux kernel.

Run-time Environment (RTE)

RTE is the full suite of software needed to execute applications written in a given programming language, including

- ▶ its compilers or interpreters,
- ▶ libraries,
- ▶ loaders, and others

RTE and System Calls

- ▶ The RTE provides a system call interface allowing programmers to access system calls availed by the operating system.
 - ▶ Each system call is typically assigned a system call number
 - ▶ The system call interface maintains a table indexed according to the system call numbers.
 - ▶ The system call interface invokes the intended system call in the operating-system kernel and returns the status of the system call.

Linker and Loader

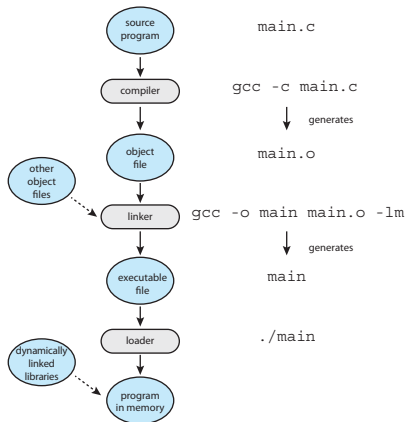


Figure: The role of the linker and loader.⁶

⁶Silberschatz, Galvin, and Gagne, *Operating system concepts*.

API Documentation on UNIX and Linux

Use `man` to query API documentation

- ▶ Example. Querying the documentation of the libc functions `_exit()`, `exit()`, and `write()`.

```
man 2 _exit
```

```
man 3 exit
```

```
man 2 write
```

- ▶ The libc functions like `write()` are wrapper functions for the `write()` system call, and sometimes, we don't make a distinction of the two.

UNIX Manual Pages

`man man`

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions eg /etc/passwd
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. `man(7)`, `groff(7)`
8. System administration commands (usually only for root)
9. Kernel routines [Non-standard]

Manual Pages for UNIX Manual Pages

```
man man
man 1 intro
man 2 intro
man 3 intro
...
man 8 intro
man 2 syscall
man 2 syscalls
```

Types of System Calls

- ▶ Process control.
 - ▶ create/terminate process, load, execute; get/set process attributes; wait/signal event; allocate/free memory
- ▶ File management
 - ▶ create/delete file; open, close; read, write, reposition; get/set file attributes
- ▶ Device management
 - ▶ request/release device; read, write, reposition; get/set device attributes; logically attach or detach devices
- ▶ Information maintenance
 - ▶ get/set time or date; get/set system data; get/set process, file, or device attributes
- ▶ Communications
 - ▶ create/delete communication connection; send/receive messages; transfer status information; attach/detach remote devices
- ▶ Protection
 - ▶ get/set file permissions

Example Windows and Linux System Calls

	Windows	Unix
Process control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File management	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device management	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communications	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Using System Calls and APIs in an Example Program

Write a CopyFile program to copy a file, e.g.,

```
CopyFile SourceFile.bin DestFile.bin
```

Outline

- 1 Objectives
- 2 OS Operations
- 3 OS Services
- 4 OS User Interface
- 5 OS Programmer Interface (System Calls)
- 6 OS Design and Structures**
- 7 OS Debugging and Monitoring

Design and Implementation Consideration

- ▶ Design goals
- ▶ Policies vs mechanisms
- ▶ Implementation

OS Structures

- ▶ Monolithic structure, i.e., placing all of the functionality of the kernel into a single, static binary file that runs in a single address space.
- ▶ Layered structure, i.e., breaking an OS into a number of layers (levels), e.g., the bottom layer (layer 0) is the hardware, the highest (layer N) is the user interface.
- ▶ Microkernel, i.e., removing all nonessential components from the kernel and implementing them as user level programs that reside in separate address spaces.
- ▶ Loadable kernel modules (LKM), i.e., the kernel has a set of core components and can link in additional services via modules, either at boot time or during run time.
- ▶ Hybrid systems.

Example Loadable Kernel Module in Linux

Device drivers often exist as loadable kernel modules in Linux systems. See the tutorial for a simple character device driver.

Outline

- 1 Objectives
- 2 OS Operations
- 3 OS Services
- 4 OS User Interface
- 5 OS Programmer Interface (System Calls)
- 6 OS Design and Structures
- 7 OS Debugging and Monitoring**

Failure Analysis

- ▶ Program failure.
 - ▶ Using log files and core dumps. A core dump is a capture of the memory of the process stored in a file for later analysis
 - ▶ Using a debugger to probe running programs or core dumps.
- ▶ Kernel failure (called crash). When a crash occurs, OS kernel saves error information to a log file and the memory state to a crash dump (stored in a file).

Performance Tuning and Monitoring

Using counters and tracing.

Using Counters

- ▶ For example, on Linux systems,
 - ▶ Per-Process
 - ▶ `ps` reports information for a single process or selection of processes
 - ▶ `top` reports real-time statistics for current processes
 - ▶ System-Wide
 - ▶ `vmstat` reports memory-usage statistics
 - ▶ `netstat` reports statistics for network interfaces
 - ▶ `iostat` reports I/O usage for disks ⁷

Most counter-based tools on Linux systems read statistics from the `/proc` file system.

⁷On Debian Linux systems, run as root, `apt-get install sysstat`

Tracing

- ▶ For example, on Linux systems,
 - ▶ Per-Process
 - ▶ `strace` traces system calls invoked by a process ⁸
 - ▶ `ltrace` traces library calls invoked by a process ⁹
 - ▶ `gdb` is a source-level debugger
 - ▶ System-Wide
 - ▶ `perf` is a collection of Linux performance tools ¹⁰
 - ▶ `tcpdump` collects network packets ¹¹

⁸To install it, run as root, `apt-get install strace`

⁹To install it, run as root, `apt-get install ltrace`

¹⁰To install it, run as root, `apt-get install linux-perf`






¹¹To install it, run as root, `apt-get install tcpdump`

Tracing Toolkits

- ▶ BCC stands for BPF Compiler Collection, a toolkit that provides tracing features for Linux systems.
- ▶ BCC is a Python front end for eBPF (extended Berkeley Packet Filter)
- ▶ Developers have been leveraging eBPF to write kernel-mode applications ¹²

¹²See discussion and presentation at <https://news.ycombinator.com/item?id=21691024>

References I

-  Jones, Don and Jeffery Hicks. *Learn Windows PowerShell 3 in a month of lunches*. Manning Publications Co., 2012.
-  Leveson, Nancy G. "The Therac-25: 30 Years Later". In: *Computer* 50.11 (2017). Available: <https://ieeexplore.ieee.org/iel7/2/8102264/08102762.pdf>, pp. 8–11.
-  Newham, Cameron and Bill Rosenblatt. *Learning the bash shell: Unix shell programming*. 3rd edition. " O'Reilly Media, Inc.", 2009.
-  Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. *Operating system concepts*. 10th edition. John Wiley & Sons, 2018.
-  *X86 Assembly/Interfacing with Linux*. Available: https://en.wikibooks.org/wiki/X86_Assembly/Interfacing_with_Linux, retrieved on February 5, 2020.