Introduction to Unix

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
What is an OS?

- **OS is a resource manager**
  - Sharing
  - Protection
  - Fairness
  - Performance

- **OS provides the program execution environment**
  - Hides the messy details which must be performed.
  - Presents users with a virtual machine, easier to use.
Unix (1)

- Unix is
  - Interactive
  - Time-sharing
  - Multi-tasking
  - Multi-user

- Flavors of Unix
  - System V (AT&T→USL→Novell→SCO→Caldera→SCO)
  - BSD (UC Berkeley)
  - SunOS, Solaris (Sun)
  - IRIX (SGI), AIX (IBM), HP-UX (HP), Mac OS X (Apple)
  - Linux, FreeBSD, NetBSD, etc.
Unix (2)

- Unix history and motivation
  - Originally developed at AT&T Bell Labs for internal use in the early 1970s.
  - Borrowed best ideas from other OS’s.
  - Unix was developed by programmers and for programmers.
    - cryptic names
    - special characters and notation
    - around 400 standard utilities
    - documentation is short on examples and tutorials
  - Unix is designed so that users can extend the functionality – to build new tools easily and efficiently.
Unix (3)

- Why Unix?
  - Used in many scientific and industrial settings
  - Huge number of free and well-written software programs
  - Open-source OS (Linux, FreeBSD, etc.)
  - Internet servers and services run on Unix
  - Largely hardware-independent
  - Based on standards
  - Many important OS concepts are developed on Unix.
  - Android!
OS Internals (1)

User Application

C Library (libc)

System Call Interface

Kernel

Arch-dependent kernel code

Hardware Platform

User space

Kernel space
OS Internals (2)

- System Call Interface
  - File System Management
  - Memory Management
  - Process Management
    - scheduler
    - IPC
    - synchronization
  - Hardware Control (Interrupt handling, etc.)

- Hardware
- User space
  - shell
  - ls

- Trap

- Kernel space
  - ps
OS Internals (3)

- OS is a special software between applications and hardware.

- When does the OS take control of the system?
  - Bootstrapping
  - System calls
  - Interrupts

- OS is an event-driven software.
 Exceptions

- An **exception** is a transfer of control to the OS in response to some event.

```
User Process       OS

event  ->  current  ->  exception  ->  exception processing by exception handler
         |            |            |                               |
next     |            |            |                               |
         |            |            |                               |
```

exception return (optional)
Handling Exceptions

- **Vector table**
  - Each type of event has a unique exception number $k$.
  - Index into jump table (interrupt vector)
  - Jump table entry $k$ points to a function (exception handler)
  - Handler $k$ is called each time exception $k$ occurs.
Asynchronous Exceptions

- **Interrupts**
  - Caused by events external to the processor
    - Indicated by setting the processor’s interrupt pin
    - Handler returns to “next” instruction.
  - Examples
    - I/O interrupts
      » Hitting a key at the keyboard
      » Arrival of a packet from a network
      » Arrival of a data sector from a disk
    - Hard reset interrupt
      » Hitting the reset button
    - Soft reset interrupt
      » Hitting CTRL-ALT-DELETE on a PC
Synchronous Exceptions

- **Traps**
  - Intentional
  - **System calls**, breakpoint traps, special instructions, etc.
  - Returns control to “next” instruction

- **Faults**
  - Unintentional but possibly recoverable
  - page faults (recoverable), protection faults (unrecoverable), etc.
  - Either re-executes faulting (“current”) instruction or aborts

- **Aborts**
  - Unintentional and unrecoverable
  - Parity error, machine check, etc.
  - Aborts current program
Fault Example (1)

- **Memory reference: invalid**
  - User writes to memory location
  - Address is not valid
  - Page handler detects invalid address
  - Sends **SIGSEGV** signal to user process
  - User process exits with “segmentation fault”

```c
int a[1000];
main ()
{
    a[5000] = 13;
}
```

![Diagram](https://via.placeholder.com/150)
Fault Example (2)

- Memory reference: valid but not resident
  - User writes to memory location
  - That portion (page) of user’s memory is currently on disk.
  - Page handler must load page into physical memory
  - Returns to faulting instruction
  - Successful on second try.

User Process

\[ \text{event} \rightarrow \text{movl} \rightarrow \text{page fault} \rightarrow \text{return} \]

OS

\[ \text{Create page and load into memory} \]

```c
int a[1000];
main ()
{
    a[500] = 13;
}
```
### Trap Example

- **Opening a file**
  - User calls open (filename, options)
    - Function open executes system call instruction `int`
  - OS must find or create file, get it ready for reading or writing.
  - Returns integer file descriptor

```
0804d070 <__libc_open>:
  ...
  804d082:   cd  80
  804d084:   5b
  804d085:   int  $0x80
  804d086:   pop  %ebx
```

User Process

```
int
pop
```

OS

```
exception
return
```
POSIX Standards

- **IEEE Standard 1003.x**
  - POSIX is a standard that describes a single interface to a Unix-like operating system.
    - 1003.1 System Application Program Interface (Kernel)
    - 1003.2 Shell and Utilities
    - 1003.4 Real-time Extensions
    - 1003.7 System Administration, etc.
  - POSIX is not an implementation – it is a description!
  - Most system vendors are now conforming to POSIX standards (specifically IEEE 1003.1)
    - Even Microsoft provides a set of POSIX utilities with the Windows NT 4.0 Resource Kit.
## POSIX System Calls

<table>
<thead>
<tr>
<th>Process Management</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td></td>
<td>Create a new process</td>
</tr>
<tr>
<td>waitpid</td>
<td></td>
<td>Wait for a process to exit</td>
</tr>
<tr>
<td>execve</td>
<td></td>
<td>Load a new binary image</td>
</tr>
<tr>
<td>exit</td>
<td></td>
<td>Terminate execution</td>
</tr>
<tr>
<td>kill</td>
<td></td>
<td>Send a signal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Management</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td></td>
<td>Create a file or open an existing file</td>
</tr>
<tr>
<td>close</td>
<td></td>
<td>Close a file</td>
</tr>
<tr>
<td>read</td>
<td></td>
<td>Read data from a file</td>
</tr>
<tr>
<td>write</td>
<td></td>
<td>Write data to a file</td>
</tr>
<tr>
<td>lseek</td>
<td></td>
<td>Move the file pointer</td>
</tr>
<tr>
<td>stat</td>
<td></td>
<td>Get various file attributes</td>
</tr>
<tr>
<td>chmod</td>
<td></td>
<td>Change the file access permission</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File System Management</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkdir</td>
<td></td>
<td>Create a new directory</td>
</tr>
<tr>
<td>rmdir</td>
<td></td>
<td>Remove an empty directory</td>
</tr>
<tr>
<td>link</td>
<td></td>
<td>Make a link to a file</td>
</tr>
<tr>
<td>unlink</td>
<td></td>
<td>Destroy an existing file</td>
</tr>
<tr>
<td>mount</td>
<td></td>
<td>Mount a file system</td>
</tr>
<tr>
<td>umount</td>
<td></td>
<td>Unmount a file system</td>
</tr>
<tr>
<td>chdir</td>
<td></td>
<td>Change the current working directory</td>
</tr>
</tbody>
</table>

...
Summary

- You need to master Unix/Linux.
- OS is an event-driven software.
- Trap is a pathway to the kernel.
- OS provides various services via system calls.