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, 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
  - I/O Management (device drivers)

- Memory Management

- Process Management
  - scheduler
  - IPC
  - synchronization

Hardware Control (Interrupt handling, etc.)

Hardware

User space

- shell
- ls
- trap
- ps

Kernel space

- shell

Protection
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.

![Diagram showing the process of exception handling:]
- Event
- Current
- Next
- Exception
- Exception processing by exception handler
- 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.
  - Abort 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](image)
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.

```
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

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

---

### Diagram

```
User Process  -->  OS

int       -->         exception
           |                   |
           V                   V
pop       -->         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

**Process Management**
- `fork` - Create a new process
- `waitpid` - Wait for a process to exit
- `execve` - Load a new binary image
- `exit` - Terminate execution
- `kill` - Send a signal

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

**File Management**
- `open` - Create a file or open an existing file
- `close` - Close a file
- `read` - Read data from a file
- `write` - Write data to a file
- `lseek` - Move the file pointer
- `stat` - Get various file attributes
- `chmod` - Change the file access permission

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

**File System Management**
- `mkdir` - Create a new directory
- `rmdir` - Remove an empty directory
- `link` - Make a link to a file
- `unlink` - Destroy an existing file
- `mount` - Mount a file system
- `umount` - Unmount a file system
- `chdir` - Change the current working directory

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