Course resources

Text book:

OPERATING SYSTEMS: DESIGN AND IMPLEMENTATION
Second Edition
ANDREW S. TANENBAUM
ALBERT S. WOODHULL

Reference:

OPERATING SYSTEMS : internals and design principles
William Stallings

My experiences as system programmer

DOS protected programming
Writing Device Deriver
- SYS for DOS
- KMD for WinNT
- Kernel mode module for linux
- VxD for Win95/98
- WDM for Win2k/XP

Concurrent programming (windows & linux)
Using Multimedia interface for windows ( audio , video and stillimage )
VFW , DirectX and Windows multimedia API
Direct programming of Graphic card, video blaster, TV Tuner and DSP
PC-Interface programming
Hacking TDI layer of Win32 to build RTC and monitor network events

Course content: text book (70%) + system code review (30%)
Midterm(40%) + Final(60%)+project(10%)
Chapter 1

INTRODUCTION
1.1 WHAT IS AN OPERATING SYSTEM?
1.2 HISTORY OF OPERATING SYSTEMS
1.3 OPERATING SYSTEM CONCEPTS
1.4 SYSTEM CALLS
1.5 OPERATING SYSTEM STRUCTURE

![Diagram showing the components of a computer system]

**Figure 1-1.** A computer system consists of hardware, system programs, and application programs.
- virtual machine, microprogram, architecture
- example: reading a disk (by direct control of hardware & by system call)
- kernel or supervisor mode & user mode

1.1 what is an operating system?

1.1.1 extended machine (Virtual machine)
- top down
- floppy disk I/O by NEC FD765 (FDC)
- File Oriented interface

1.1.2 Resource Manager
- Bottom up
- Resources: (processors + mem + timers + disks + mice + network adaptor + printer + modem + …)

1.2 History of Operating Systems
- First G: 1945-1955 vacuum tubes & Plugsboards
  20000 tubes, no OS
- Second G: 1955-1965 Trans & batch systems
  computer rooms, jobs and fortran.: FMS

![Diagram](image-url)

**Figure 1-2.** An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.
Third G: 1965-1980 ICs & Multiprogramming
- IBM System/360 series (1401->7094)
  - JCL: job control language, compatible systems, millions of asm lines->bugs
- Multiprogramming

**Figure 1-3.** Structure of a typical FMS job.

**Figure 1-4.** A multiprogramming system with three jobs in memory.
Spooling & timesharing
CTSS (time sharing system) on MIT success
MIT & Bell Lab & GE -> MULTICS
Ken Thompson + PDP7 + MULTICS + Bell Lab -> UNIX

UNIX source available -> incompatible version & academic works ->
IEEE standard for UNIX (POSIX)

Fourth G: 1980 present personal computers (PCs)
LSI & VLSI -> low price PC
Mini (PDP11) -> micro (XT/AT) & SUN Workstation
Intel: 8088/286/386/486 -> pentum -> pentum pro
   -> PII-PIII-P4
   -> Celeron
Microsoft MSDOS -> Win31 (on msdos) -> Win95 -> win98 -> win2000
   -> windows for work group (Win311) -> WinNT3.5 -> WinNT4
SUN Micro Systems: -> SUN workstation (RISC tech.), power mini, net server
   -> UNIX (solaris) + XWindows
mid 1980s -> Net Operating Systems, net interface (NIC), low level drivers
   -> distributed Operating systems -> complex OS (delay + multiple CPU +
   communication problems + parallelism & concurrency + ...)

History of MINIX (LINUX)
Unix 6.0+ under AT&T license + source available + John Lions 1996 booklet
   -> Unix src study in Universities
Unix 7.0 + legal problem -> only theory { scheduling alg, deadlock, I/O and FILE
Tanenbaum + Unix 7.0 + POSIX -> miniUNIX (MINIX)
   For IBM PC, Atari, SPARC, MAC,...
   30 Meg Hard
Linux Torvalds (Finish student) + usenet on MINIX -> Linux (efficient Minix
on PC)

1.3 Operating system concepts
process
   an executing program
   processes = _TEXT + _DATA + _BSS + _STACK + REGs
   timesharing sys -> process table, creatProcess, KillProcess
   child process, IPC, Synchronization.
Figure 1-5. A process tree. Process A created two child processes, B and C. Process B created three child processes, D, E, and F.

FILE:
Dir, i-node, working directory, path(/home/ali)
Protection read, write, execute → 3 bit (rwx)
Owner, group member, others → 3 field rwxr-x—x
Figure 1-6. A file system for a university department.

Figure 1-7. (a) Before mounting, the files on drive 0 are not accessible. (b) After mounting, they are part of the file hierarchy.
Operating systems  R. Azmi

System calls

Fig. 1-17. The 11 steps in making the system call read(fd, buffer, nbytes). (REF)
### Process management

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid = fork()</td>
<td>Create a child process identical to the parent</td>
</tr>
<tr>
<td>pid = waitpid(pid, &amp;statloc, options)</td>
<td>Wait for a child to terminate</td>
</tr>
<tr>
<td>s = execve(name, argv, environp)</td>
<td>Replace a process' core image</td>
</tr>
<tr>
<td>exit(status)</td>
<td>Terminate process execution and return status</td>
</tr>
</tbody>
</table>

### File management

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd = open(file, how, ...)</td>
<td>Open a file for reading, writing or both</td>
</tr>
<tr>
<td>s = close(fd)</td>
<td>Close an open file</td>
</tr>
<tr>
<td>n = read(fd, buffer, nbytes)</td>
<td>Read data from a file into a buffer</td>
</tr>
<tr>
<td>n = write(fd, buffer, nbytes)</td>
<td>Write data from a buffer into a file</td>
</tr>
<tr>
<td>position = lseek(fd, offset, whence)</td>
<td>Move the file pointer</td>
</tr>
<tr>
<td>s = stat(name, &amp;buf)</td>
<td>Get a file's status information</td>
</tr>
</tbody>
</table>

### Directory and file system management

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = mkdir(name, mode)</td>
<td>Create a new directory</td>
</tr>
<tr>
<td>s = rmdir(name)</td>
<td>Remove an empty directory</td>
</tr>
<tr>
<td>s = link(name1, name2)</td>
<td>Create a new entry, name2, pointing to name1</td>
</tr>
<tr>
<td>s = unlink(name)</td>
<td>Remove a directory entry</td>
</tr>
<tr>
<td>s = mount(special, name, flag)</td>
<td>Mount a file system</td>
</tr>
<tr>
<td>s = umount(special)</td>
<td>Unmount a file system</td>
</tr>
</tbody>
</table>

### Miscellaneous

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = chdir(dirname)</td>
<td>Change the working directory</td>
</tr>
<tr>
<td>s = chmod(name, mode)</td>
<td>Change a file's protection bits</td>
</tr>
<tr>
<td>s = kill(pid, signal)</td>
<td>Send a signal to a process</td>
</tr>
<tr>
<td>seconds = time(&amp;seconds)</td>
<td>Get the elapsed time since Jan. 1, 1970</td>
</tr>
</tbody>
</table>

Fig. 1-18. Some of the major POSIX system calls. The return code 
$s$ is $-1$ if an error has occurred. The return codes are as follows: 
$pid$ is a process id, $fd$ is a file descriptor, $n$ is a byte count, $position$
is an offset within the file, and $seconds$ is the elapsed time. The 
parameters are explained in the text. (REF)
while (TRUE) { /* repeat forever */
read_command(command, parameters); /* read input from terminal */
if (fork() != 0) { /* fork off child process */
  /* Parent code. */
waitpid(-1, &status, 0); /* wait for child to exit */
} else {
  /* Child code. */
execve(command, parameters, 0); /* execute command */
}

Figure 1-10. A stripped-down shell. Throughout this book, TRUE is assumed to be defined as 1.
Figure 1-11. Processes have three segments: text, data, and stack. In this example, all three are in one address space, but separate instruction and data space is also supported.
struct stat {
    short st_dev;       /* device where i-node belongs */
    unsigned short st_ino; /* i-node number */
    unsigned short st_mode; /* mode word */
    short st_nlink;     /* number of links */
    short st_uid;       /* user id */
    short st_gid;       /* group id */
    short st_rdev;      /* major/minor device for special files */
    long st_size;       /* file size */
    long st_atime;      /* time of last access */
    long st_mtime;      /* time of last modification */
    long st_ctime;      /* time of last change to i-node */
};

Figure 1-12. The structure used to return information for the \texttt{STAT} and \texttt{FSTAT} system calls. In the actual code, symbolic names are used for some of the types.
Figure 1-13. A skeleton for setting up a two-process pipeline.

special files:
device mode=char mode(raw mode), block mode(cooked mode)
fd=open(...); read(fd,...), write(fd,...), ioctl(fd,command,...)
1.5 OPERATING SYSTEM STRUCTURE

Figure 1-16. How a system call can be made: (1) User program traps to the kernel. (2) Operating system determines service number required. (3) Operating system calls service procedure. (4) Control is returned to user program.
General layered system, E. W. Dijkstra → THE system

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>The operator</td>
</tr>
<tr>
<td>4</td>
<td>User programs</td>
</tr>
<tr>
<td>3</td>
<td>Input/output management</td>
</tr>
<tr>
<td>2</td>
<td>Operator-process communication</td>
</tr>
<tr>
<td>1</td>
<td>Memory and drum management</td>
</tr>
<tr>
<td>0</td>
<td>Processor allocation and multiprogramming</td>
</tr>
</tbody>
</table>

**Figure 1-17.** A simple structuring model for a monolithic system.

**Figure 1-18.** Structure of the THE operating system.
Exokernels

No file system, allow Ring0 File I/O as a library procedure
Based on end-to-end argument (Engler et al., 1995)
operating system should do is securely allocate resources (e.g., the CPU and the disks) among the competing users

Virtual Machine

Figure 12-1. One possible design for a modern layered operating system.

Figure 1-19. The structure of VM/370 with CMS.

two approach for VM (in start or in all)
Client/Server Model

- moving code to higher layer $\rightarrow$ minimal kernel
- Client and server user mode process $\rightarrow$ small OS

**Figure 1-20.** The client-server model.
Figure 1-21. The client-server model in a distributed system.

- Mechanism for kernel
- Policy for servers