INF239 Sistemas Operativos

MODERN OPERATING SYSTEMS
Fourth Edition
by Andrew S. TANENBAUM, Herbert BOS

Section 1.6 SYSTEM CALLS

Clase 3

Viktor Khlebnikov

Ingeniería Informática
Departamento de Ingeniería
Pontificia Universidad Católica del Perú

2018

Contenido

1.6 System calls

- Programming interface to the services provided by the OS
  - request privileged service from the kernel
  - typically written in a high-level *system* language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface** (**API**) rather than direct system call use
  - provides a simpler interface to the user than the system call interface
  - reduces coupling between kernel and application, increases portability
- Common APIs
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS)
  - Win32 API for Windows
- Implementation
  - software trap, register contains system call number
  - *syscall* instruction for fast control transfer to the kernel

1.6 System calls

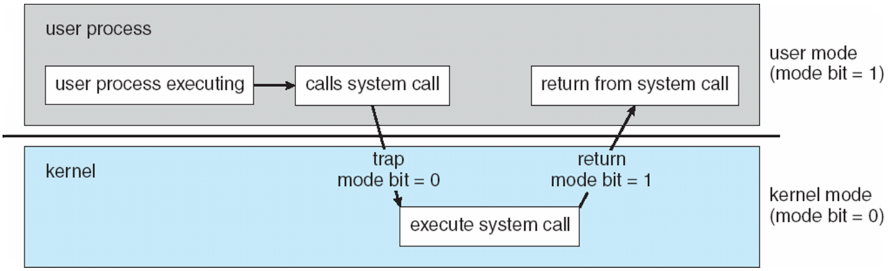
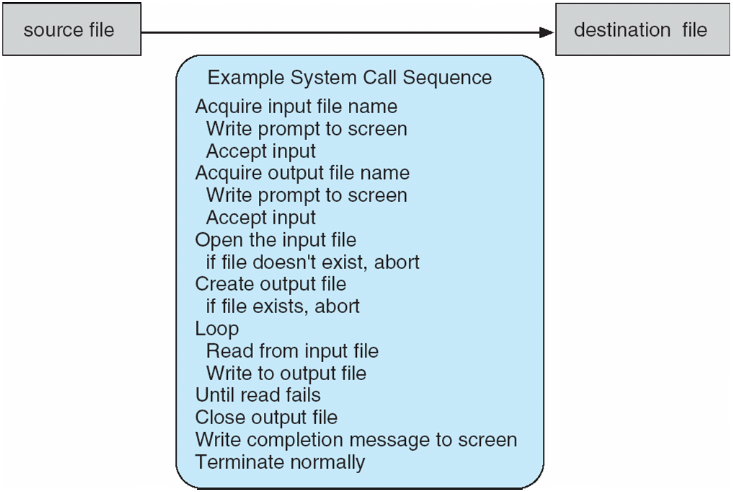


Figura: 1-8 (OSC). Transition from user to kernel mode.

1.6 System calls

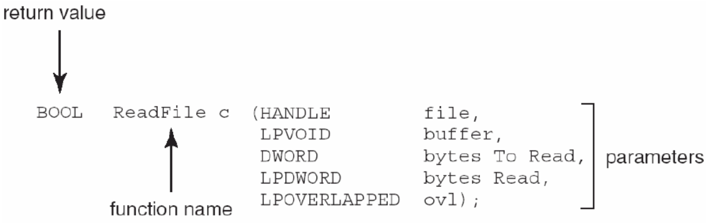


```
Example System Call Sequence
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Figura: 2-1 (OSC). Example of how system calls are used.

1.6 System calls

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



```
BOOL   ReadFile c  (HANDLE          file,
                    LPVOID          buffer,
                    DWORD           bytes To Read,
                    LPDWORD         bytes Read,
                    LPOVERLAPPED    ovl);
```

- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

Figura: 2-2 (OSC). The API for the ReadFile() function.

1.6 System calls

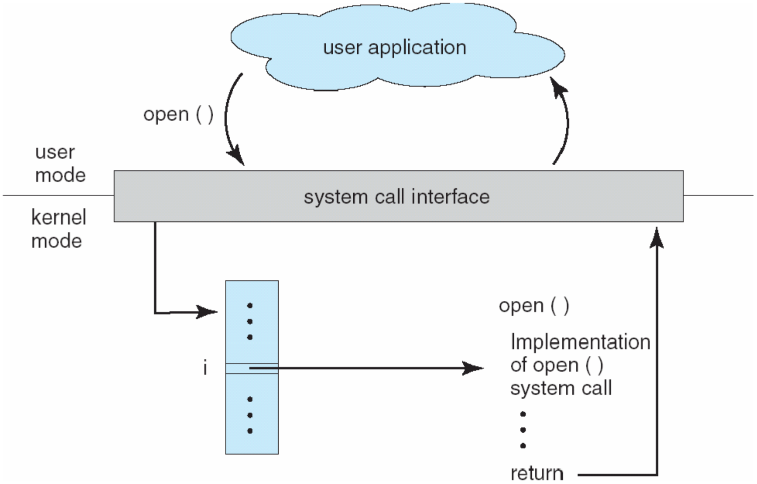


Figura: 2-3 (OSC). The handling of a user application invoking the open() system call.

1.6 System calls

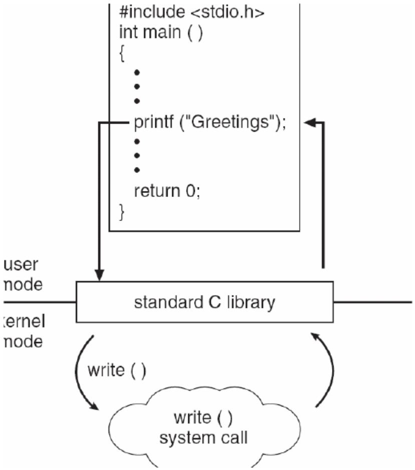


Figura: 2-6 (OSC). C library handling of write().

1.6 SYSTEM CALLS

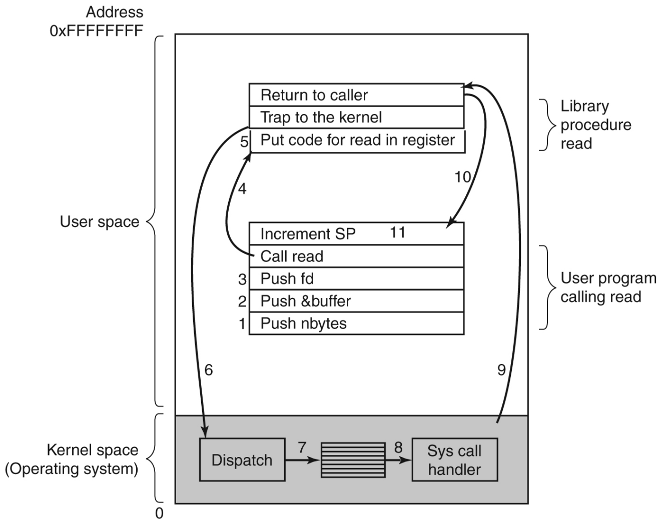
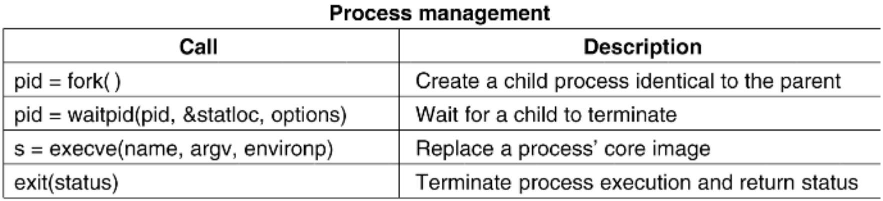


Figura: 1-17. The 11 steps in making the system call **read(fd, buffer, nbytes)**.

1.6.1 System Calls for Process Management



Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Figura: 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. The parameters are explained in the text.

Manuales de referencia: man

```
$ man man

MAN(1)                Útiles de Páginas de Manual              MAN(1)

NOMBRE
       man - una interfaz de los manuales de referencia electrónicos
...
La  siguiente  tabla  muestra  los  números de sección del manual y los
tipos de páginas que contienen.

1   Programas ejecutables y guiones del intérprete de órdenes
2   Llamadas del sistema (funciones servidas por el núcleo)
3   Llamadas de la biblioteca (funciones contenidas en las  bibliotecas
    del sistema)
4   Ficheros especiales (se encuentran generalmente en /dev)
5   Formato de ficheros y convenios p.ej. I/etc/passwd
6   Juegos
7   Paquetes de macros y convenios p.ej. man(7), groff(7).
8   Órdenes  de  admistración  del  sistema (generalmente solo son para
    root)
```

Manuales de referencia: fork

```
$ man fork

FORK(2)               Linux Programmer's Manual               FORK(2)
NAME
       fork - create a child process
SYNOPSIS
       #include <unistd.h>
       pid_t fork(void);
DESCRIPTION
fork()  creates a new process by duplicating the calling process.  The
new process, referred to as the child, is an exact  duplicate  of  the
calling  process,  referred to as the parent, except for the following
points:
*  The child has its own unique process ID,  and  this  PID  does  not
   match the ID of any existing process group (setpgid(2)).
*  The  child's  parent process ID is the same as the parent's process
   ID.
...
```

1.6.1 System Calls for Process Management

```
#define TRUE 1

while (TRUE) {                                      /* repeat forever */
      type_prompt( );                               /* display prompt on the screen */
      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 */
      }
}
```

Figura: 1-19. A stripped-down shell. Throughout this book, *TRUE* is assumed to be defined as 1.

1.6.1 System Calls for Process Management

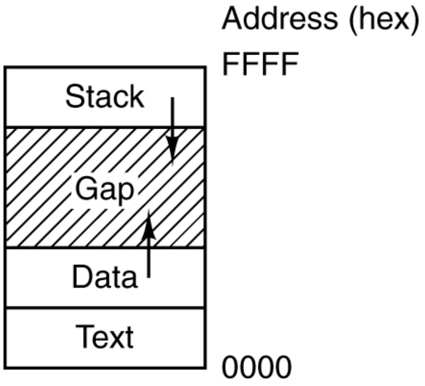
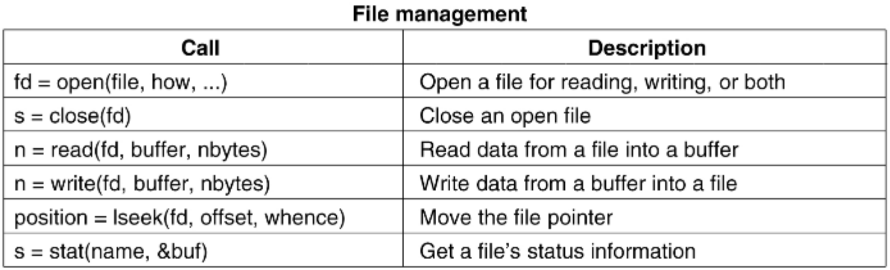


Figura: 1-20. Processes have three segments: text, data, and stack.

1.6.2 System Calls for File Management

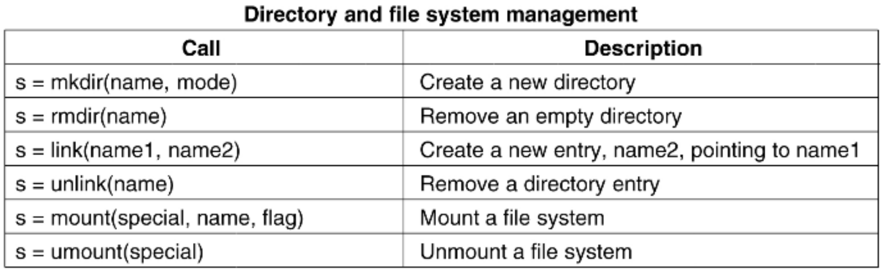


File management

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

Figura: 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: *fd* is a file descriptor, *n* is a byte count, and *position* is an offset within the file. The parameters are explained in the text.

1.6.3 System Calls for Directory Management



Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Figura: 1-18. Some of the major POSIX system calls. The return code *s* is -1 if an error has occurred. The parameters are explained in the text.

1.6.3 System Calls for Directory Management

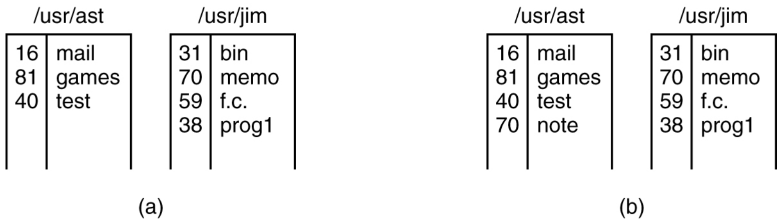


Figura: 1-21. (a) Two directories before linking */usr/jim/memo* to ast's directory. (b) The same directories after linking: `link("/usr/jim/memo","/usr/ast/note");`

1.6.3 System Calls for Directory Management

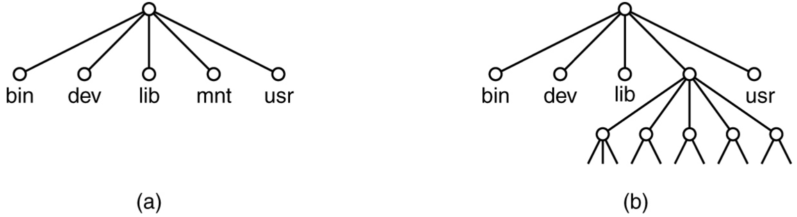
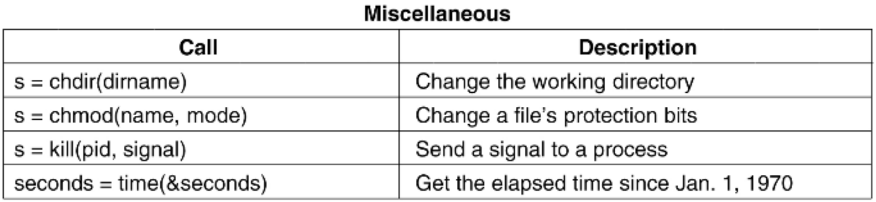


Figura: 1-22. (a) File system before the mount. (b) File system after the mount: `mount("/dev/sdb0","/mnt",0);`

1.6.4 Miscellaneous System Calls



Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Figura: 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: *seconds* is the elapsed time. The parameters are explained in the text.

1.6.5 The Windows Win32 API

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Figura: 1-23. The Win32 API calls that roughly correspond to the UNIX calls of Fig. 1-18.