

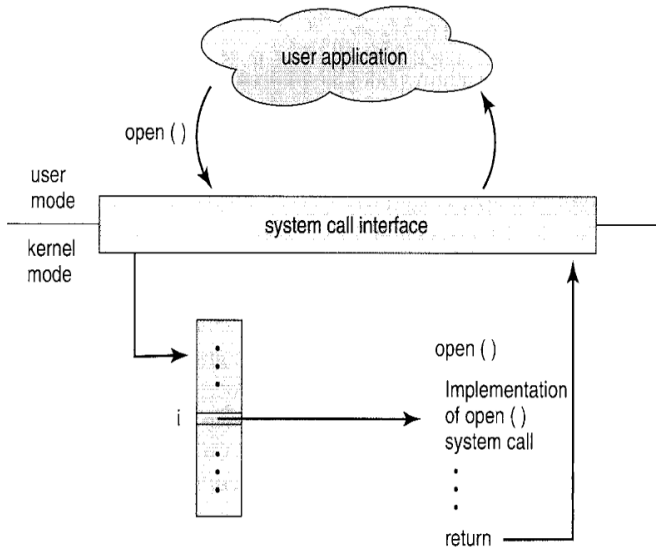
API del SO

Rodolfo Baader

Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, segundo cuatrimestre de 2014

(2) Llamadas al sistema



(3) Tipos de llamadas al sistema

- Control de Procesos
- Administración de archivos
- Administración de dispositivos
- Mantenimiento de información
- Comunicaciones

(4) Ejemplos de llamadas al sistema

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

(5) Overhead system call

En una AMD Athlon 64 1.8 GHz

Tiempo en microsegundos (10^{-6} segundos) de 1000000 iteraciones:

system call: 200127

llamada a función: 17358

asignación en arreglos: 15175

Llamada es 1.143855 veces más cara que asignación.

System call es 11.529381 veces más cara que llamada.

(6) POSIX

- POSIX: Portable Operating System Interface; X: UNIX.
- El término fue sugerido por Richard Stallman.
- POSIX.1, Core Services:
 - Creación y control de procesos.
 - Pipes
 - Señales.
 - Operaciones de archivos y directorios
 - Excepciones
 - Errores del bus.
 - Biblioteca C
 - Instrucciones de entrada/salida y de control de dispositivo (ioctl).

(7) Creación de procesos

fork - create a new process

```
#include <unistd.h>

pid_t fork(void);
```

The fork() function shall create a new process. The new process (child process) shall be an exact copy of the calling process (parent process) except as detailed below:

- The child process shall have a unique process ID.
- The child process shall have a different parent process ID, which shall be the process ID of the calling process.
- The child process shall have its own copy of the parent's file descriptors. Each of the child's file descriptors shall refer to the same open file description with the corresponding file descriptor of the parent.
- The child process shall have its own copy of the parent's open directory streams. Each open directory stream in the child process may share directory stream positioning with the corresponding directory stream of the parent.
- File locks set by the parent process shall not be inherited by the child process.
- Any semaphores that are open in the parent process shall also be open in the child process.
- The child process shall not inherit any address space memory locks established by the parent process via calls to mlockall() or mlock().
- ...

(8) Creación de procesos

Parent

```
main()
{
    pid = 3456
    pid=fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
}

void ChildProcess()
{
    .....
}

void ParentProcess()
{
    .....
}
```

Child

```
main()
{
    pid = 0
    pid=fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
}

void ChildProcess()
{
    .....
}

void ParentProcess()
{
    .....
}
```


(9) Reemplazo de procesos

execl, execv, execl, execve, execlp, execvp - execute a file

```
#include <unistd.h>

int execv(const char *path, char *const argv[]);
```

The exec family of functions shall replace the current process image with a new process image. The new image shall be constructed from a regular, executable file called the new process image file. There shall be no return from a successful exec, because the calling process image is overlaid by the new process image.

(10) Espera de procesos

wait, waitpid, waitid - wait for process to change state

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);

pid_t waitpid(pid_t pid, int *status, int options);
```

All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state.

(11) Terminación de procesos

exit - cause normal process termination

```
#include <stdlib.h>

void exit(int status);
```

The `exit()` function causes normal process termination and the value of `status` is returned to the parent.


(12) Manejo de archivos

- open - open and possibly create a file or device
- read - read from a file descriptor
- write - write to a file descriptor
- lseek - reposition read/write file offset


```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int open(const char *pathname, int flags);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);
```


(13) IPC

- Los procesos aislados no son de demasiada utilidad en el mundo real.
- Además de hacer E/S hacia los dispositivos, es deseable que puedan intercambiar información entre sí.
- La comunicación entre procesos suele llamarse *IPC*, por *InterProcess Communication*. 
- Hay varias formas de IPC:
 - Memoria compartida.
 - Algún otro recurso compartido (por ejemplo, archivo, base de datos, etc.).
 - Pasaje de mensajes.
- Vamos a concentrarnos en el pasaje de mensajes.
- En particular, entre procesos de la misma máquina (aunque esto no es obligatorio).

(14) IPC (cont.)

- Los SO brindan varias interfaces para hacer IPC:
 - Unix SysV *Transport Layer Interface*.
 - BSD Sockets. Esta es la línea evolutivamente ganadora.
- La idea de los sockets es la siguiente:
 - Un socket es el extremo de una comunicación. Piensen en el enchufe de la pared.
 - Luego de crearlo tengo que unirlo a algo, a una forma de comunicación concreta. Sería como conectar el enchufe a los cables.
 - Una vez que tengo un socket conectado puedo leer y escribir de él como si fuese un dispositivo.
 - Es decir, vale todo lo que vimos relacionado con bloqueo y scheduling.
 - Los detalles los vamos a ver en la práctica.
 - Lo importante es que para un proceso, hacer IPC es como hacer E/S. 

(15) IPC sincrónico/asincrónico

- La comunicación entre procesos puede ser sincrónica o asincrónica: 
- Sincrónica
 - El emisor no termina de enviar hasta que el receptor no recibe.
 - Si el mensaje se envió sin error suele significar que también se recibió sin error.
 - En general involucra bloqueo del emisor.
- Asincrónica
 - El emisor envía algo que el receptor va a recibir en algún otro momento.
 - Requiere algún mecanismo adicional para saber si el mensaje llegó.
 - Libera al emisor para realizar otras tareas, no suele haber bloqueo, aunque puede haber un poco (por ejemplo, para copiar el mensaje a un buffer del SO).

(16) Dónde estamos

- Vimos
 - El concepto de proceso en detalle.
 - Sus diferentes actividades.
 - Qué es una system call.
 - Una introducción al scheduler.
 - Hablamos de multiprogramación, y vimos su relación con E/S.
 - Introdujimos IPC.
- En el taller:
 - Vamos a entender IPC más en detalle.
 - Vamos a ver en la práctica varios de los conceptos de hoy.
- En un rato:
 - Nos metemos a fondo con scheduling.