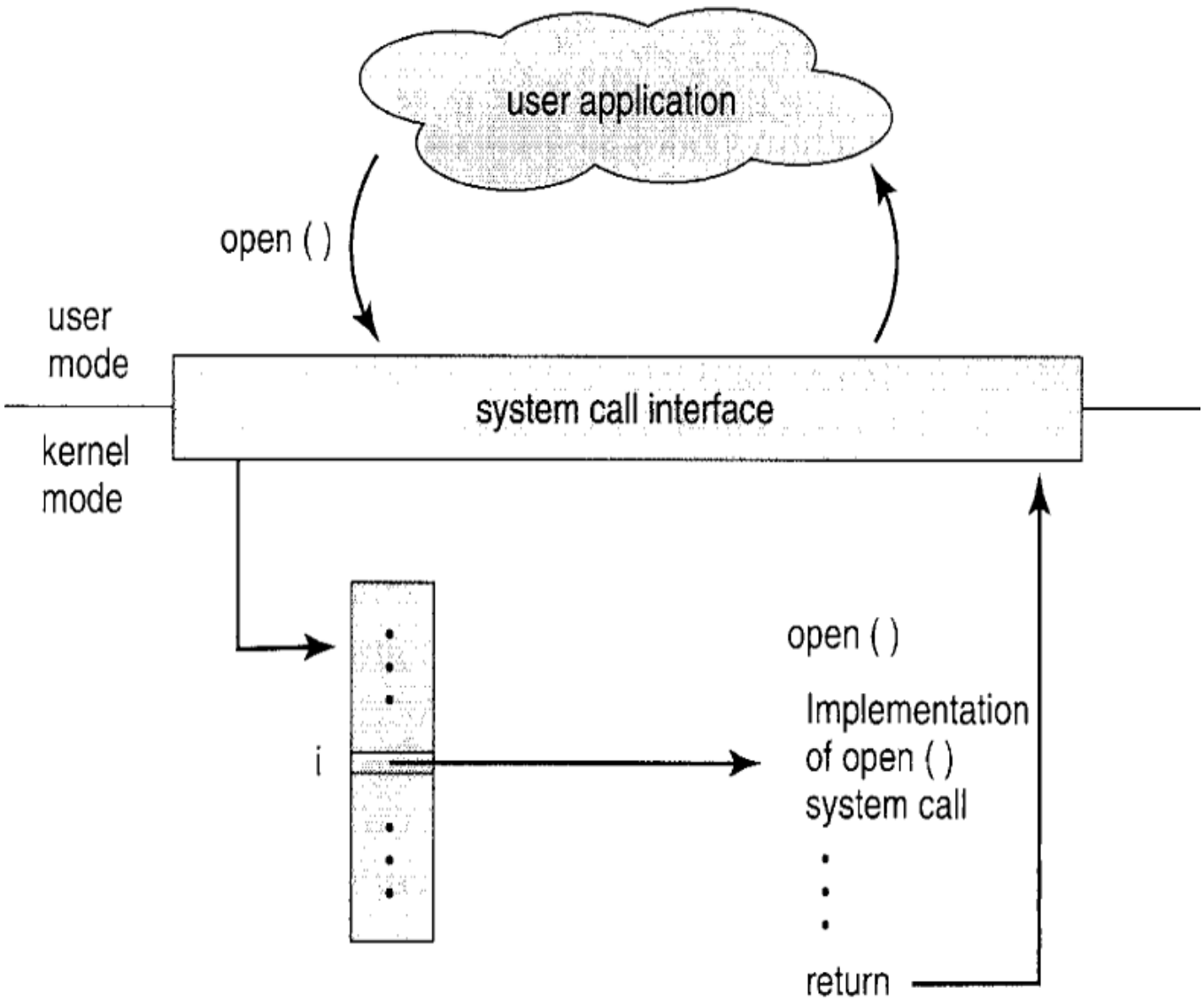
API del SO

Rodolfo Baader

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

Sistemas Operativos, segundo cuatrimestre de 2015

(2) Llamadas al sistema



(3) Tipos de llamadas al sistema

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

(4) POSIX

- POSIX: Portable Operating System Interface; X: UNIX.
- IEEE 1003.1/2008 `http://goo.gl/k7WGnP`
- Core Services:
  - Creación y control de procesos
  - Pipes
  - Señales
  - Operaciones de archivos y directorios
  - Excepciones
  - Errores del bus.
  - Biblioteca C
  - Instrucciones de E/S y de control de dispositivo (ioctl).

(5) Ejemplos de llamadas al sistema

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

(6) Overhead system call

En una AMD Athlon 64 1.8 GHz

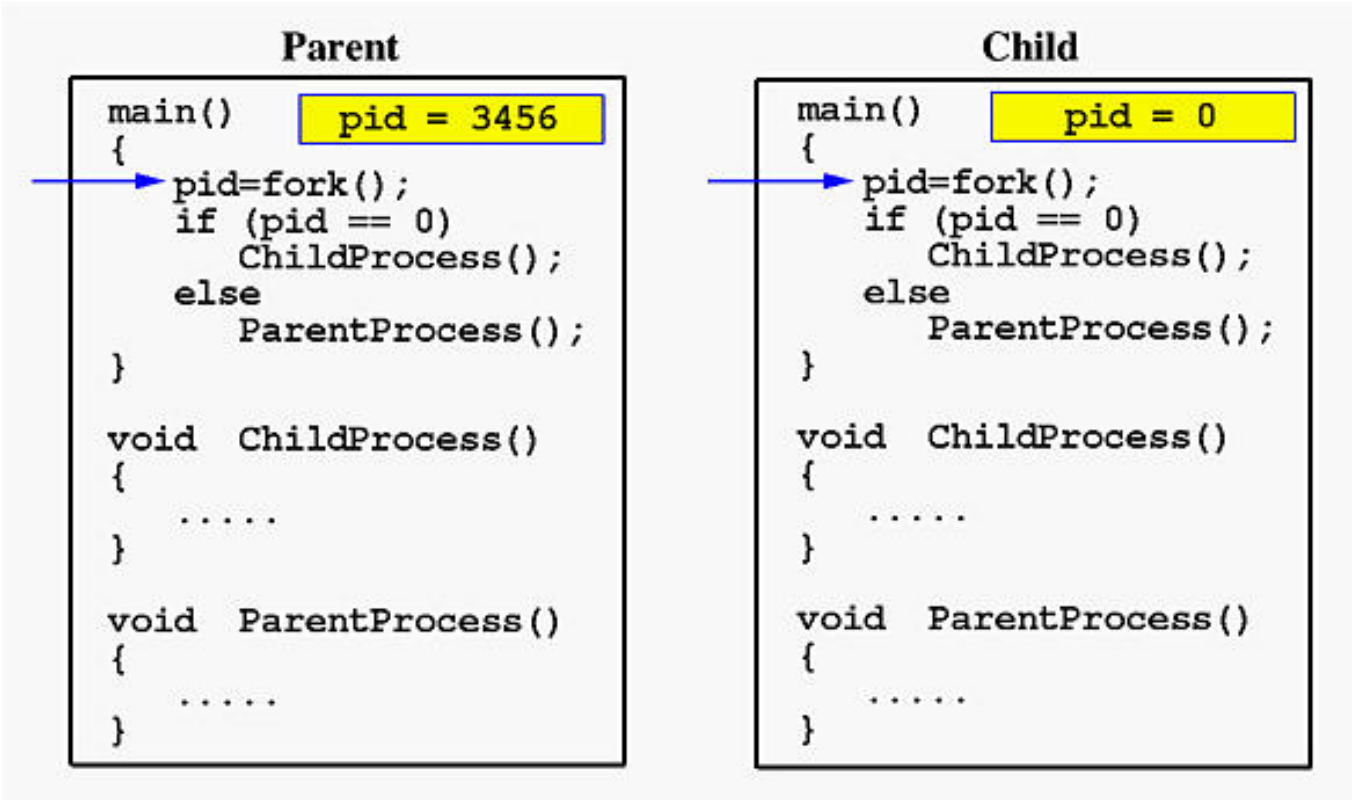
```
Tiempo en microsegundos (10^-6 segundos) de 1000000
iteraciones:
system call:              200127
llamada a función:        17358
asignación en arreglos: 15175

Llamada es 1.143855 veces más cara que asignación.
System call es 11.529381 veces más cara que llamada.
```

(7) API

- Creación y control de procesos

```
pid_t fork(void);
pid_t vfork(void);
// vfork crea un hijo sin copiar la memoria del padre
// El hijo tiene que hacer exec

int execve(const char *fn, char *const argv[],
char *const envp[]);

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

void exit(int status);

// Linux, no POSIX
int clone(...)
// El hijo comparte parte del contexto con el padre
// Usado para implementar threads
```

(8) Creación de procesos (fork)



(9) Creación de procesos (fork)

- Ejemplo tipo

```
1  int main(void) {
2    int foo = 0;
3    pid_t pid = fork();
4    if (pid == -1) exit(EXIT_FAILURE);
5    else if (pid == 0) {
6       printf("%d: Hello world\n", getpid());
7       foo = 1;
8    }
9    else {
10      printf("%d: %d created\n", getpid(), pid);
11      int s; (void)waitpid(pid, &s, 0);
12      printf("%d: %d finished(%d)\n", getpid(), pid, s);
13   }
14   printf("%d: foo(%p)= %d\n", getpid(), &foo, foo);
15   exit(EXIT_SUCCESS);
16 }
```

(10) Creación de procesos (fork)

- Ejemplos de ejecuciones posibles

```
$ ./main
3724: 3725 created
3725: Hello world
3725: foo(0x7fff5431fb6c)= 1
3724: 3725 finished(0)
3724: foo(0x7fff5431fb6c)= 0

$ ./main
3815: Hello world
3815: foo(0x7fff58c3eb6c)= 1
3814: 3815 created
3814: 3815 finished(0)
3814: foo(0x7fff58c3eb6c)= 0
```

- Manejo de archivos

```
// creación y apertura
int open(const char *pathname, int flags);

// lectura
ssize_t read(int fd, void *buf, size_t count);

// escritura
ssize_t write(int fd, const void *buf, size_t count);

// actualiza la posición actual
off_t lseek(int fd, off_t offset, int whence);
// whence = SEEK_SET -> comienzo + offset
// whence = SEEK_CUR -> actual + offset
// whence = SEEK_END -> fin + offset
```

(12) Inter-Process Communication (IPC)

- Hay varias formas de IPC:
  - Memoria compartida.
  - Algún otro recurso compartido (archivo, base de datos, etc.).
  - Pasaje de mensajes
- BSD/POSIX *sockets*
  - Un socket es el extremo de una comunicación (enchufe)
  - Para usarlo hay que conectarlo
  - Una vez conectado se puede leer y escribir de él
  - Hacer IPC es como hacer E/S. 
  - Los detalles los vamos a ver en la práctica.

(13) IPC: modos

- Sincrónico
  - 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.
- Asincrónico
  - 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).

(14) 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.