
Sistemas Operativos

GSyC

nemo@gsync.escet.urjc.es



2002

Llamadas al sistema

¿Qué es esto?

- El sistema da abstracciones
- Cada abstracción tiene un conjunto de llamadas al sistema
- Las típicas
 - Ficheros
 - Procesos
 - Memoria
 - Entrada/Salida

Ficheros

- Datos en disco o dispositivos con un nombre.
- Tenemos directorios (tablas de nombres).
- Tenemos un directorio actual.
camino absoluto/relativos

Llamadas al sistema

`int open(path,CREAT|RDWR|EXCL|TRUNC)` abre un fichero

`link(path,path)` añade un nombre para un fichero.

`unlink(path)` elimina un nombre (puede borrar).

`close(fd)` lo cierra.

`read(fd,buff,len)` lee a partir de la posición actual.

`write(fd,buff,len)` escribe a partir de la posición actual.

`lseek(fd,off,SET|CUR|END)` cambia la posición actual.

`chmod(path,mode)` Cambia permisos.

`chown(path,owner,group)` Cambia propietarios

`stat(path,stat)` Devuelve metadatos.

Ejemplo

```
ficheros(){  
    int fd;  
  
    fd = open("/a/b", OWRITE|OCREATE);  
    seek(fd, 1*1024, 0);  
    write(fd, "hola", 4);  
    close(fd);  
    link("/a/b", "/c/d");  
    unlink("/a/b");  
}
```

Procesos

- Programas ejecutando
- Cada uno ejecuta independientemente.

Llamadas al sistema

`pid_t fork()` Creamos un proceso hijo

`pid_t getpid()` Devuelve nuestro pid.

`int {set,get}uid(uid)`

`int execve(file,argv,envp)` En nuestro lugar, ejecutamos otro prog.

Reminiscencias de la PDP, pero útil para “configurar” el entorno del proceso hijo...

`pid_t wait(status)` Esperamos a que un hijo termine.

`exit(status)` Terminamos.

`int kill(pid,sig)` Enviamos una señal al un proceso.

Ejemplo

```
procesos(){  
    print("Hola\n");  
    fork();  
    print("Mundo\n");  
}
```

Ejemplo

```
procesos(){  
    print("Hola\n");  
    fork();  
    fork();  
    print("Mundo\n");  
}
```

Ejemplo

```
procesos(){
    int pid;
    pid = fork();
    if (pid > 0){
        print("El padre sigue...");
    } else {
        print("El hijo hace un ls");
        execl("/bin/ls", "ls", "-l", 0);
    }
}
```

Ejemplo

```
procesos(){
    int pid;
    int status;
    pid = fork();
    if (pid == 0){
        print("El hijo hace un ls");
        execl("/bin/ls", "ls", "-l", 0);
    }
    wait(&status);
    print("Al hijo le fue %s", (status==0?"ok":"mal"));
}
```

Memoria

- Trozos de memoria usados por un programa.
- Posiblemente compartidos entre varios procesos.
- Posiblemente correspondiendo con un fichero.

Llamadas al sistema

`addr_t sbrk(int)` Hace crecer el segmento de datos

`int mmap(addr, len, perm, SHARED..., fd, offset)` Crea un segmento de datos compartido (o no), posiblemente correspondiendo con un fichero.

Ejemplo

```
memoria(){
    int fd;
    fd = open("/dev/zero", OREAD);
    addr = mmap(0, 16*1024, PROT_READ, SHARED, fd, 0);
    close(fd);
    if (fork()){
        ...el padre usa el segmento
    } else {
        ...el hijo tambien
    }
}
```

Entrada/Salida

- Formas de hacer entrada/salida a otros sitios.
- Fuertemente relacionada con ficheros.

Llamadas al sistema

`select(set of fd, set of fd)` Se bloquea hasta que podemos leer o escribir de alguno de los fds en los conjuntos.

`ioctl(fd, op, arg)` Permite ejecutar operaciones en dispositivos (ej. eject cdrom).

`pipe(int fd[2])` Crea una tubería. Lo que escribes en un descriptor es lo que se lee del otro.

Ejemplo

```
pipes(){
    int fd[2];
    char buf[];
    pipe(fd);
    if (fork()){
        close(fd[1]);
        read(fd[0], buf, BUFLen);
        close(fd[0]);
    } else {
        close(fd[0]);
        write(fd[1], "hola", 4);
        close(fd[1]);
    }
}
```