RESUMEN PRACTICO PARA PARCIAL.pdf

papianxo69

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Politécnica Superior. Campus de Leganés
Universidad Carlos III de Madrid

Accede al documento original

RESUMEN PRACTICO PARA PARCIAL

creat

```
#include <fcntl.h>

int fd = creat("nombre_archivo", permisos); // Crear un archivo
```

Devuelve un descriptor de archivo o `-1` en caso de error

unlink

```
#include <unistd.h>

int resultado = unlink("nombre_archivo"); // Eliminar un archivo
```

Devuelve 0 si tiene éxito y -1 en caso de error

open

```
#include <fcntl.h>

int fd = open("nombre_archivo", flags, permisos); // Abrir un archivo
```

Ejemplos de flags: `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_CREAT`, `O_TRUNC`
Devuelve un descriptor de archivo o -1 en caso de error

close

```
#include <unistd.h>

int resultado = close(fd); // Cerrar un archivo
```

- vuelve 0 si tiene éxito y -1 en caso de error

read

```
#include <unistd.h>
```

```
int bytes_leidos = read(fd, buffer, tamaño); // Leer datos de fd y los
almecena en el buffer buffer de tamaño tamaño
```

Devuelve el número de bytes leídos o -1 en caso de error

write

```
#include <unistd.h>

int bytes_escritos = write(fd, buffer, tamaño); // Escribir datos en fd del
buffer buffer de tamaño tamaño
```

Devuelve el número de bytes escritos o -1 en caso de error

lseek

```
#include <unistd.h>

int nueva_posicion = lseek(fd, desplazamiento, origen); // Mover el puntero
del archivo fd desplazamiento bytes desde origen
```

origen:

- `SEEK_SET`: Desde el inicio del archivo.
- `SEEK_CUR`: Desde la posición actual.
- `SEEK_END`: Desde el final del archivo.
  Devuelve la nueva posición o -1 en caso de error

Fork

```
#include <unistd.h>

int pid = fork(); // Crear un proceso hijo
```

Devuelve:

- En el proceso padre: el PID del proceso hijo
- En el proceso hijo: 0
- En caso de error: -1

Exec

```
#include <unistd.h>

execvp(nombre_del_programa, argumentos); // Ejecutar un programa
```

Wait

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

wait(NULL); // Esperar a que un proceso hijo termine
```

Exit

```
#include <stdlib.h>

exit(codigo_de_salida); // Terminar el proceso
```

Código de salida:

- Éxito: 0
- Error: 1

Señales

```
#include <signal.h>

void manejador(int señal) {
    // Código que se ejecuta cuando se recibe la señal
}

signal(tipo_de_señal, manejador); // Asocia un manejador a una señal
específica
```

Pipes

```
#include <unistd.h>

int fd[2]; // fd[0] para leer, fd[1] para escribir
pipe(fd); // Crear un pipe

// En el proceso de lectura:
close(fd[1]); // Cerrar extremo de escritura
read(fd[0], buffer, tamaño); // Leer datos del pipe
```


```
// En el proceso de escritura:
close(fd[0]); // Cerrar extremo de lectura
write(fd[1], datos, tamaño); // Escribir datos en el pipe
```

`pipe(fd)` devuelve -1 en caso de error