

ResumParcial.pdf



abcd_1234



Sistemas Operativos



2º Grado en Ingeniería Informática



Facultad de Informática de Barcelona (Fib)
Universidad Politécnica de Catalunya



[Accede al documento original](#)



Escuela de
Organización
Industrial

Contigo que evolucionas.
Contigo que lideras. Contigo que transformas.

Esto es EOI.
Mismo propósito,
nueva energía.



Descubre más aquí



EOI Escuela de
Organización
Industrial

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

SO: Resumen Parcial

TEORÍA Y DEFINICIONES

- **argc:** es un entero que contiene el número de elementos del array **argv**.
- **argv:** es un array de strings que contiene los parámetros de entrada cuando el programa se pone en ejecución.
- **atoi:** convierte un string en un número entero

INCLUDES

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <unistd.h>
```

COMANDOS Y FUNCIONES CONSOLA

- **man:** acceder al manual
- **ls:** Muestra el contenido del directorio
- Si ejecutamos "ls -l" nos muestra el contenido + los permisos
- **alias:** Definir un nombre alternativo
- **mkdir:** Crea un directorio
- **rmdir:** Elimina un directorio vacío
- **mv:** Cambia el nombre de un directorio o lo mueve a otro directorio
- **cp:** Copia ficheros y directorios
- **rm:** Borra ficheros y directorios
- **echo:** Visualiza un texto
- **less:** Muestra fichero en formato apto para la terminal
- **cat:** Concatena ficheros y los muestra en su salida estándar
- **grep:** Busca texto o patrones de texto en ficheros
- **gedit:** Editor de texto para GNOME
- **env:** Ejecuta un comando con un entorno modificado
- **chmod:** Modifica los permisos de los ficheros
- **which:** Localiza un comando
- **help:** Ofrece información sobre comandos internos de la SHELL
- **export:** Define una variable de entorno
- **cd:** cambia de directorio
- **make:** Ejecutar "makefile"
- **gcc:** Compilador de C de GNU

ESCRITURA EN C:

```
void main(int argc, char *argv[ ]) {
    char buf[80];
    for (i=0; i<argc; i++){
        sprintf(buf, "El argumento %d es %s\n", i, argv[i]);
        write(1, buf, strlen(buf));
    }
    return 0;
}
```

El **buf** o **búffer** es un vector de chars de propósito general que sirve para almacenar las frases. El número es la cantidad de caracteres que contiene la oración.

sprintf:

- Se utiliza para formatear un texto y almacenar el resultado en un búffer.
- Su formato es: `sprintf(búffer(char), cadena de caracteres (el texto con variables))`. (Mirar el ejemplo previo)
- **Declaración de variables:**
 - %d=int
 - %c = char
 - %s = string
- Extra: al final de cada oración se añade "\n" que ejecuta el salto de línea.

strlen: devuelve el número de caracteres de una cadena, incluido los espacios.

write:

- Es una llamada al sistema para escribir el buffer en el dispositivo
- Se declara tal que: `write(1,buffer,strlen(buffer));`

INICIALIZAR UN PROGRAMA

Desde la consola llamas al programa y mandas los parámetros.

#programa a b

//argc = 3

//Argv = [programa, a, b]

El argumento 0 es el programa

El argumento 1 es a

El argumento 2 es b

GESTIÓN DE PROCESOS

FUNCIONES

- **getpid:** Retorna el PID del proceso que la ejecuta
- **getppid:** Devuelve el PID del proceso padre
- **fork:** crea un nuevo proceso, hijo del que la ejecuta
- **exit:** Termina el proceso que ejecuta la llamada
- **waitpid:** Espera la finalización de un proceso hijo
- **exec (execlp):** Cambiar ejecutable = Mutar proceso
- **perror:** Escribe un mensaje del último error producido
- **ps:** Devuelve información de los procesos
- **proc:** Pseudo-file system que ofrece información de los datos del kernel

CREACIÓN DE PROCESOS Y HERENCIA

Cuando utilizamos una función "**int fork()**" lo que estamos haciendo que un proceso cree un nuevo proceso. Se crea en relación a la jerarquía de padre-hijo.

El padre y el hijo se ejecutan de forma

concurrente.

La memoria del hijo se inicializa con una copia de la memoria padre.

El hijo inicia la ejecución en el punto en el que estaba el padre en el momento de la ejecución. Por tanto: PC hijo = PC padre. El valor de retorno fork:

- Padre recibe PID del hijo
- Hijo recibe 0

HEREDA:

- El espacio de direcciones lógico (código, datos, pila, etc)
- La tabla de programación de signals
- Los dispositivos virtuales
- El usuario/grupo (credenciales)
- Variables de entorno

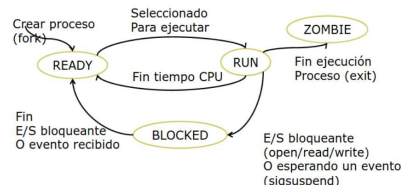
NO HEREDA:

- PID, PPID
- Contadores internos de utilización
- Alarmas y signals pendientes (son propias del proceso)

Un proceso puede acabar su ejecución de manera **voluntaria** o **involuntaria (signals)**.

Si queremos sincronizar la ejecución de un padre con la finalización de su hijo podemos

ESTADOS DE LOS HIJOS



Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

usar **waitpid**.

WAITPID: Bloquea el proceso hasta que acaba un hijo (cualquiera o en concreto) y el padre recoger la información mediante la función. Es decir el waitpid o wait sirve para recoger la información y liberar a los hijos que se quedan en estado zombie.

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

- waitpid(-1,NULL,0) → Espera (con bloqueo si es necesario a un hijo **cualquiera**
- waitpid(pid_hijo,NULL,0) → Esperar a un hijo con pid == pid_hijo

Parámetro pid==-1 Estado hijos al hacer waitpid	options==0	options==WNOHANG
Algún hijo zombie	No se bloquea Trata la muerte de un zombie (no está especificado cual) Devuelve el pid del hijo tratado	Idem que options==0
Todos los hijos vivos	Se bloquea hasta que uno acaba (cualquiera) Trata la muerte del que haya acabado Devuelve el pid del hijo tratado	No se bloquea Devuelve 0
Parámetro pid==pidh Estado de pidh al hacer waitpid	options==0	options==WNOHANG
Zombie	No se bloquea Trata la muerte del hijo Devuelve pidh	Idem que flags=0
Vivo	Se bloquea hasta que acaba Trata la muerte del hijo Devuelve pidh	No se bloquea Devuelve 0

EXEC (o execlp)

Al hacer el fork(), el espacio de direcciones es el mismo. Si queremos ejecutar otro código, el proceso debe MUTAR.

execlp: Un proceso cambia su propio ejecutable por otro ejecutable, pero el proceso es el mismo.

- Todo el contenido del espacio de direcciones cambia
- Se mantiene todo lo relacionado con la identidad del proceso: Contadores de usos internos, signals pendientes...
- Se modifican aspectos
- relacionados con el ejecutable o el espacio de direcciones.




























```
int execlp(const char *file, const char *arg, ...);
```

EJEMPLOS DE CREACIÓN DE PROCESOS

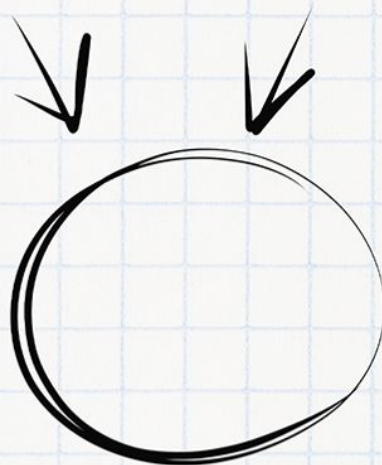
```
Caso1: queremos que hagan líneas de código diferentes.
int ret = fork();
if (ret == 0) { //Estas líneas solo las ejecuta el hijo, tenemos dos
}
else if (ret < 0) { //En este caso ha fallado el fork, solo hay 1 pr
}
else { //Estas líneas solo las ejecuta el padre
}
//Estas líneas las ejecutan los dos
Caso2: queremos que hagan lo mismo.
fork(); //Aquí si no hay error, hay dos procesos.
-----
SECUENCIAL: Forzamos que el padre espere que se termine un hijo ante
int i, ret;
for(i = 0; i < num_procs; ++i) {
    ret = fork();
    if (ret == 0) { //Estas líneas solo las ejecuta el hijo
        codigoHijo();
        exit(0);
    }
    else if (ret < 0) { control_error();
    }
    waitpid(-1, NULL, 0);
}
-----
CONCURRENTE: Primero creamos todos los errores y después esperamos a
int i, ret;
for(i = 0; i < num_procs; ++i) {
    ret = fork();
    if (ret == 0) { //Estas líneas solo las ejecuta el hijo codigoHi
        exit(0);
    }
    else if (ret < 0) control_error();
}
while(waitpid(-1, NULL, 0) > 0);
```

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	 PLAN TURBO	 PLAN PRO	 PLAN PRO+
 Descargas sin publi al mes	10 	40 	80 
 Elimina el video entre descargas			
 Descarga carpetas			
 Descarga archivos grandes			
 Visualiza apuntes online sin publi			
 Elimina toda la publi web			
 Precios Anual <input type="checkbox"/>	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo,
¿Qué nota vas a sacar?



WUOLAH

COMUNICACIÓN ENTRE PROCESOS (Signals)

Signal: Eventos enviados por otros procesos del mismo usuario o por el kernel para indicar determinadas condiciones.

El proceso puede capturar todos los tipos de signal menos SIGKILL y SIGSTOP.

TIPOS DE SIGNALS

SIGCHILD: Ignorar → Un proceso hijo ha terminado o sido parado

SIGCONT: Continúa si estaba parado

SIGSTOP: STOP → Parar proceso

SIGINT: Terminar → Interrumpido desde teclado (Ctrl + C)

SIGALRM: Terminar → El contador definido por la llamada alarm ha terminado

SIGKILL: Terminar → Terminar proceso

SIGSEGV: CORE → Referencia inválida a memoria

SIGUSR1: Terminar → Definido por el usuario (proceso)

SIGUSR2: Terminar → Definido por el usuario (proceso)

INFORMACIÓN DEL LAB

- **sigaction:** reprograma la acción asociada a un evento concreto
- **kill:** envía a un proceso concreto a un proceso
- **sigsuspend:** Bloquea el proceso que la ejecuta hasta que recibe un signal
- **sigprocmask:** Permite modificar la máscara de signals bloqueados por el proceso
- **Alarm:** Programa el envío de un SIGALRM al cabo de N segundos
- **Sleep:** Función que bloquea el proceso durante el tiempo que se le pasa como parámetro
- **/bin/kill:** Envía un evento a un proceso
- **Ps:** Muestra información sobre los procesos del sistema
- **Waitpid:** Espera a que finalice un proceso

DEFINICIÓN DE STRUCT SIGACTION

Nos fijaremos en tres campos:

1. **sa_handler:** puede tomar tres valores:

SIG_IGN: ignorar el signal recibido

SIG_DLF: usar el tratamiento por defecto

función de usuario definida por cabecera predefinida: void function(int s)

2. **sa_mask:** signal que se añaden a la máscara de signals que el proceso tiene bloqueados

3. **sa_flags:** para configurar el comportamiento (si vale 0 se usa la configuración por defecto)

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins?

Plan Turbo: barato

Planes pro: más coins

perdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

MANIPULACIÓN DE MÁSCARAS DE SIGNALS

- sigemptyset: inicializa una máscara sin signals → `int sigemptyset(sigset_t *mask)`
- sigfillset: inicializa una máscara con todos los signal → `int sigfillset(sigset_t *mask)`
- sigaddset: añade signal a la mascara que se pasa como parámetro → `int sigaddset(sigset_t *mask, int signum);`
- sigdelset: elimina el signal de la máscara que se pasa como parámetro → `int sigdelset(sigset_t *mask, int signum)`
- sigismember: devuelve cierto si el signal está en la máscara → `int sigismember(sigset_t *mask, int signum)`

EJEMPLO CAPTURA DE SIGNALS:

```
void main() {
    char buffer[128];
    struct sigaction trat;
    sigset_t mask;
    sigemptyset(&mask);
    trat.sa_mask=mask;
    trat.sa_flags=0;
    trat.sa_handler = f_sigint;
    sigaction(SIGINT, &trat, NULL); // Cuando llegue SIGINT se ejecu
    // f_sigint
    while(1) {
        sprintf(buffer, "Estoy haciendo cierta tarea\n");
        write(1, buffer, strlen(buffer));
    }
}

void f_sigint(int s) {
    char buffer[128];
    sprintf(buffer, "SIGINT RECIBIDO!\n");
    exit(0);
}
```

DES/BLOQUEAR SIGNALS

```
int sigprocmask(int operación, sigset_t *mascara, sigset_t *vieja mascara)
```

Operación =

- SIG_BLOCK: añadir signals que indica la máscara a la máscara de signals bloqueados del proceso
- SIG_UNBLOCK: **quitar** los signals que indica máscara a la máscara de signals bloqueados por el proceso
- SIG_SETMASK: hacer que la máscara de signals bloqueados del proceso pase a ser el parámetro de máscara

ESPERAR UN EVENTO

```
int sigsuspend(sigset_t *mascara) //espera hasta que llegue un evento cualquiera
```

- Bloquea el proceso hasta que llega un evento cuyo tratamiento no sea SIG_IGN
- Mientras el proceso está bloqueado en el sigsuspend máscara será los signals que no se recibirán (signals bloqueado)
- Al salir del sigsuspend automáticamente se recupera la mascara que habia y se tratan los signals pendientes.

PROTECCIÓN UNIX

Los usuarios se identifican mediante username y password (userID)

Los usuarios pertenecen a grupos (groupID)

- Para ficheros:
 - Protección asociada a:
Lectura/Escritura/Ejecución (rwx) →
Comando ls para consultar, chmod para modificar.
 - A nivel proceso: Los procesos tienen un usuario que determina los derechos.
- La excepción es ROOT. Puede acceder a cualquier objeto y puede ejecutar operaciones privilegiadas.

SEGURIDAD

- **Físico:** Las máquinas y los terminales de acceso deben encontrarse en habitaciones / edificios seguros
- **Humano:** Es importante controlar quién tiene acceso a los sistemas
- **Sistema Operativo**
 - Evitar procesos saturen el sistema
 - Asegurar que determinados servicios están siempre funcionando
 - Asegurar que determinados puertos no están operativos
 - Controlar que los procesos no puedan acceder fuera de su propio espacio de direcciones
- **Red:** La mayoría de datos hoy en día se suelen mover por la red. Este es el componente usualmente más atacado.

