

Sistemas Operativos

sesión 10: procesos

Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Agenda



Linux



Servicios para
procesos



Ejercicios



Agenda



Linux



Servicios para
procesos



Ejercicios



Contenidos



- Control de trabajos
- Control de procesos
- Compresión y descompresión

Contenidos



- **Control de trabajos**
- Control de procesos
- Compresión y descompresión

Motivación

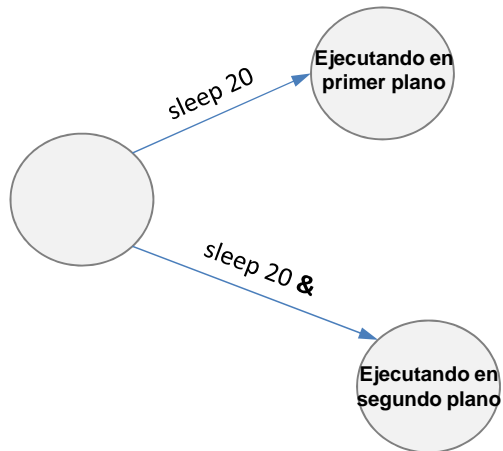
- Trabajos = aplicaciones ejecutadas en un intérprete de mandatos
- ¿Cómo puedo gestionarlos?

```
acaldero@phoenix:~/work$ sleep 25 &  
acaldero@phoenix:~/work$ jobs
```

...

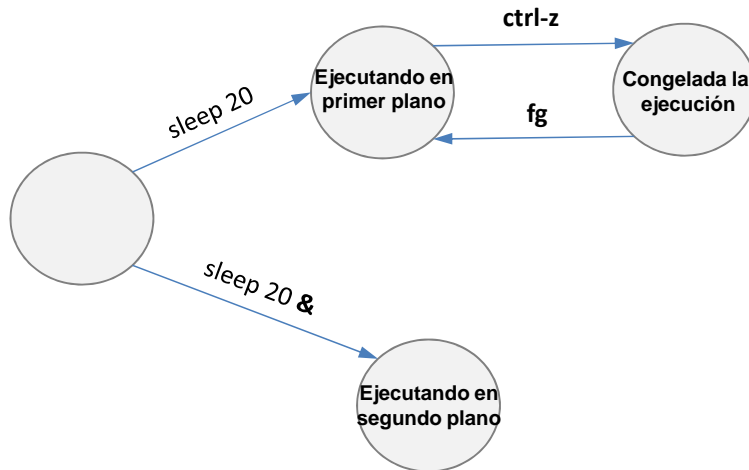
Control de trabajos

ejecución en primer/segundo plano



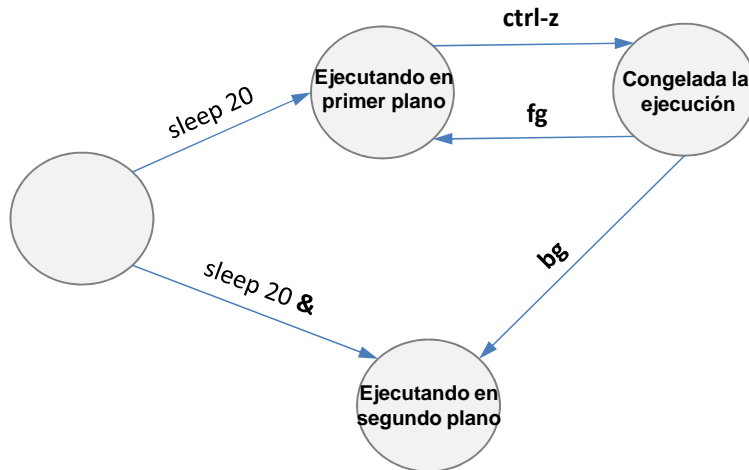
Control de trabajos

de ejecución en primer plano a congelación



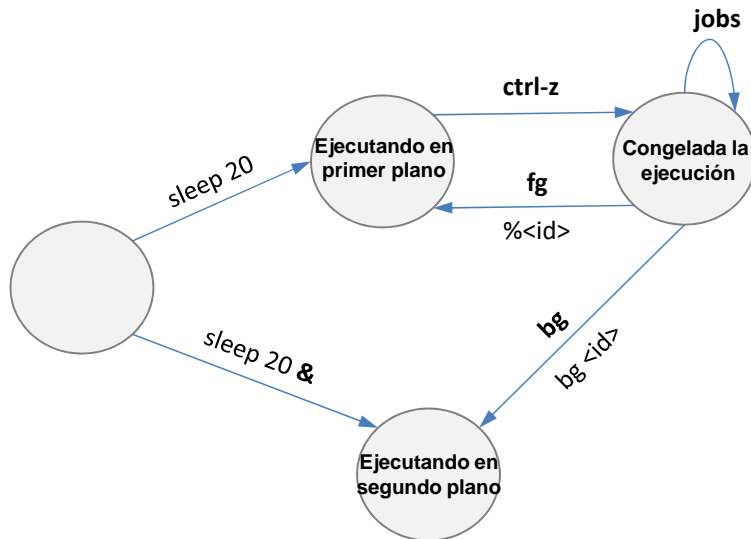
Control de trabajos

de congelado a ejecución en segundo plano



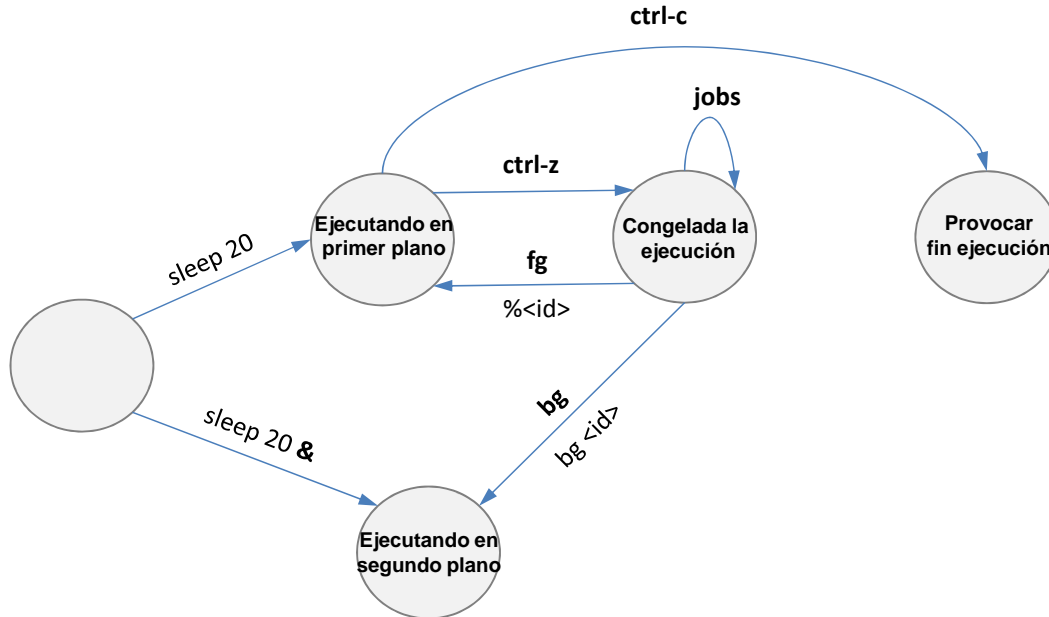
Control de trabajos

listado de trabajos



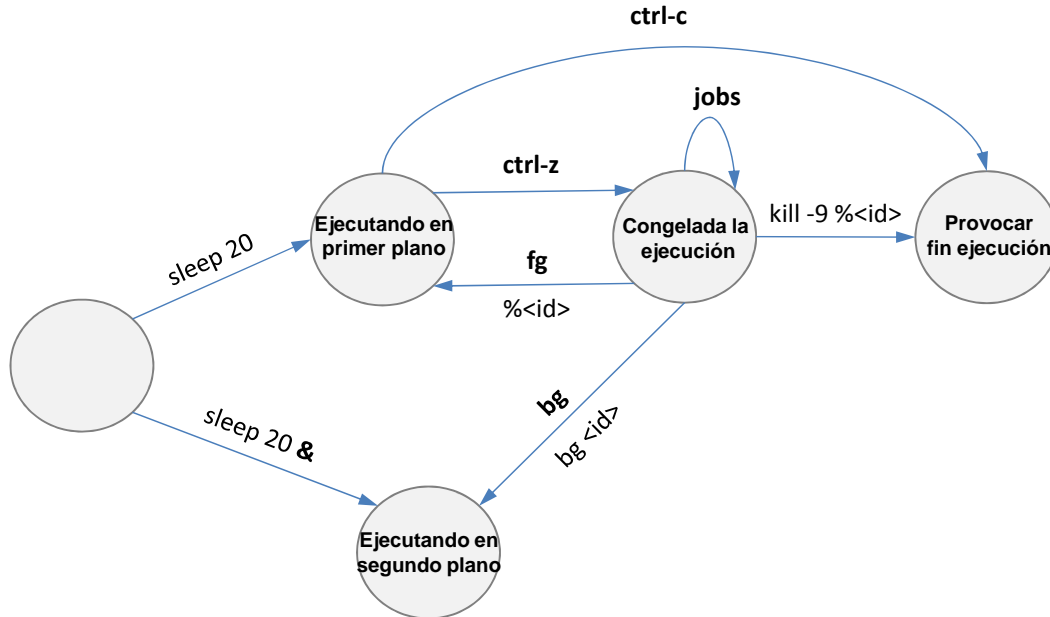
Control de trabajos

finalización ejecutando en primer plano



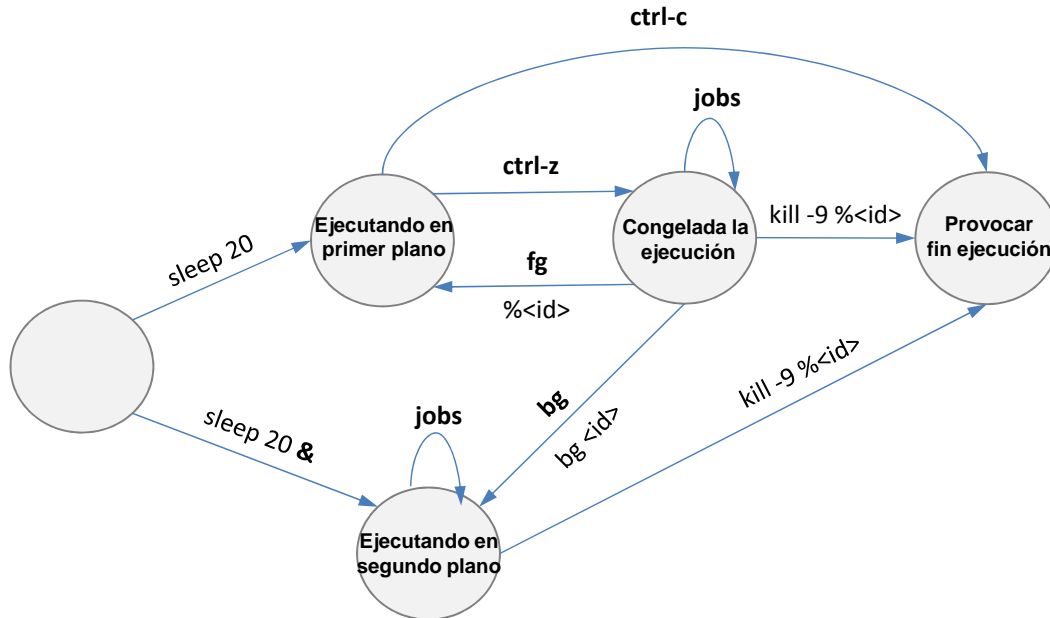
Control de trabajos

finalización trabajo congelado

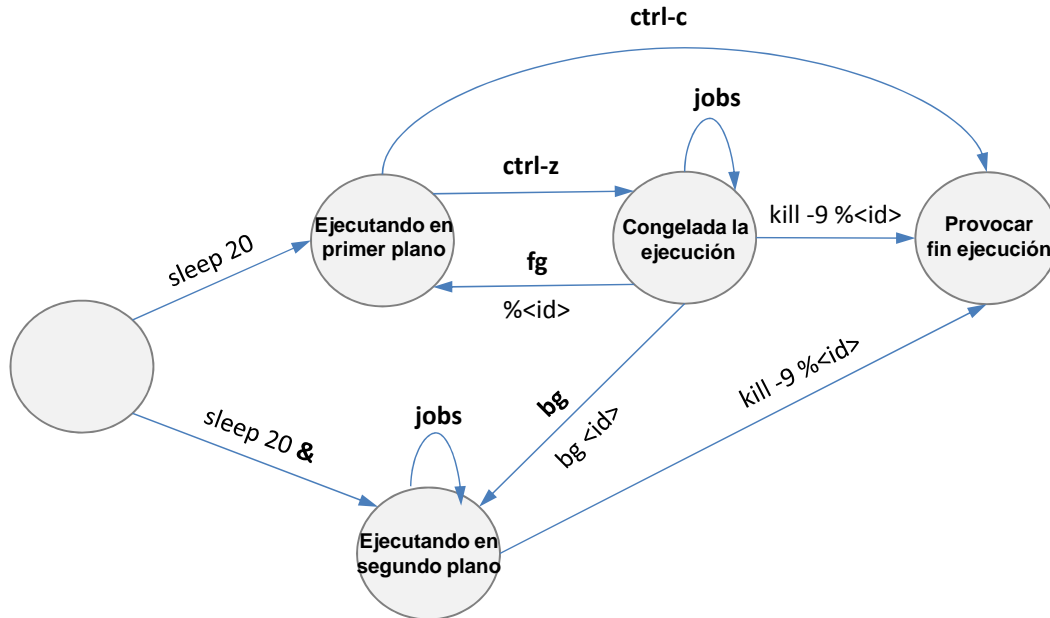


Control de trabajos

finalización ejecutando en segundo plano



Control de trabajos



Contenidos



- Control de trabajos
- **Control de procesos**
- Compresión y descompresión

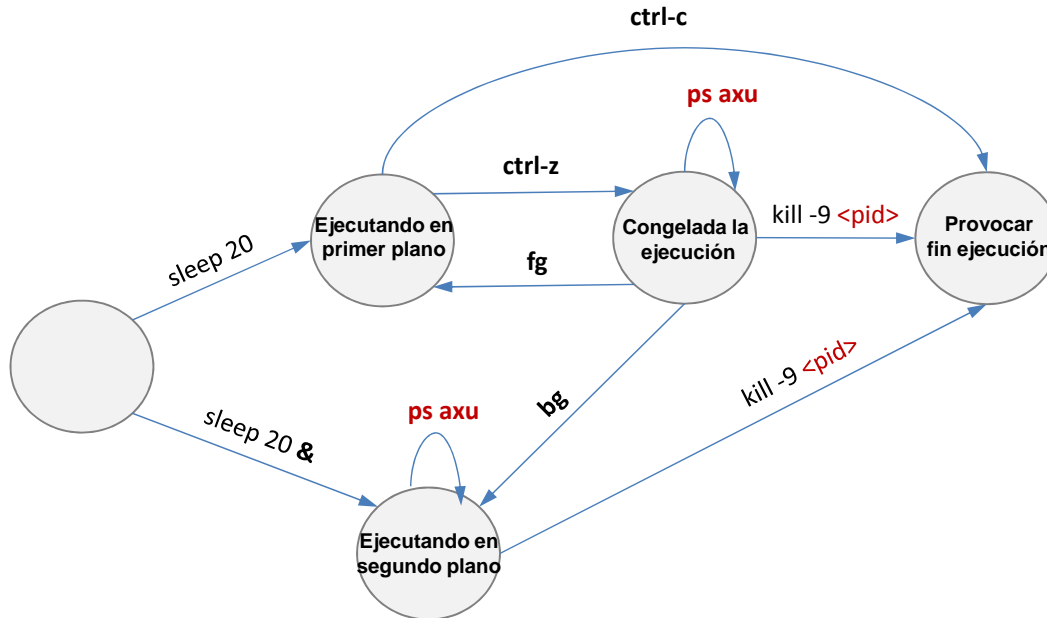
Motivación

- ¿Cómo puedo gestionar los programas que están siendo ejecutados desde otra sesión de intérprete de mandatos?

```
acaldero@phoenix:~/work$ ps axu  
acaldero@phoenix:~/work$ kill -1 1
```

...

Control de procesos



Contenidos



- Control de trabajos
- Control de procesos
- **Compresión y descompresión**

Comprimir, descomprimir y listar

	Extensión	Comprimir	Descomprimir	Listar
zip	.zip	zip -r f.zip <ficheros>	zip -a f.zip	unzip -l f.zip
Tar + gzip	.tgz .tar.gz	tar zcvf f.tgz <ficheros>	tar zxvf f.tgz	tar ztvf f.tgz
Tar + bzip2	.tbz2 .tar.bz2	tar jcf f.tbz2 <ficheros>	tar jxf f.tbz2	tar jtvf f.tbz2
rar	.rar	rar a f.rar <ficheros>	rar x f.rar	rar l f.rar

Agenda



Linux



**Servicios para
procesos**



Ejercicios



Contenidos



- fork+exec+exit simple
- fork+exec+exit múltiple

Contenidos



- **fork+exec+exit simple**
- fork+exec+exit múltiple

Repaso

fork() + exec()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

fork() + exec()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```


Repaso

fork() + exec()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

fork() + exec()

```
pid = fork();  
pid == 0)
```

padre

```
pid = fork();  
(pid == 0)  
execvp("./programa", NULL);  
exit(1);  
//  
while (pid != -1) { wait(&status); }
```

Hijo

Repaso

fork() + exec()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

fork() + exec()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

fork() + exec()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

fork() + exec()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```

Repaso

wait() + exit()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```

Repaso

wait() + exit()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```


Repaso

wait() + exit()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

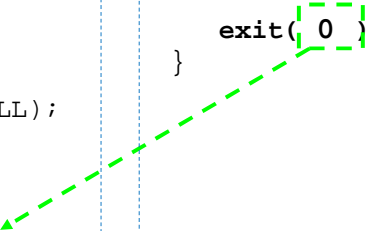
    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```



Repaso

wait() + exit()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

wait() + exit()

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

wait() + exit()

Contenidos



- fork+exec+exit simple
- **fork+exec+exit múltiple**

Repaso

múltiples procesos (bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else
    {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else
    {
        /* hacer algo */
        exit(3);
    }
}
```

Repaso

múltiples procesos (bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else
    {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else
    {
        /* hacer algo */
        exit(3);
    }
}
```

Repaso

múltiples procesos (bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else
    {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else
    {
        /* hacer algo */
        exit(3);
    }
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }
    else
    {
        while (pid != wait(&status));
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }
    else
    {
        while (pid != wait(&status));
    }
}
```


Repaso

múltiples procesos (bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else {
        /* hacer algo */
        exit(3);
    }
}
```

Repaso

múltiples procesos (bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else {
        /* hacer algo */
        exit(3);
    }
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }
    else {
        while (pid != wait(&status));
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }
    else {
        while (pid != wait(&status));
    }
}
```

Repaso

múltiples procesos (bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid != 0) {
        while (pid != wait(&status));
    }
    else {
        /* hacer algo */
        exit(3);
    }
}
```

Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```


Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

Repaso

múltiples procesos (no bloqueante)

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

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

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

main : {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != wait(&status));
}
```

Repaso

múltiples procesos (no bloqueante)

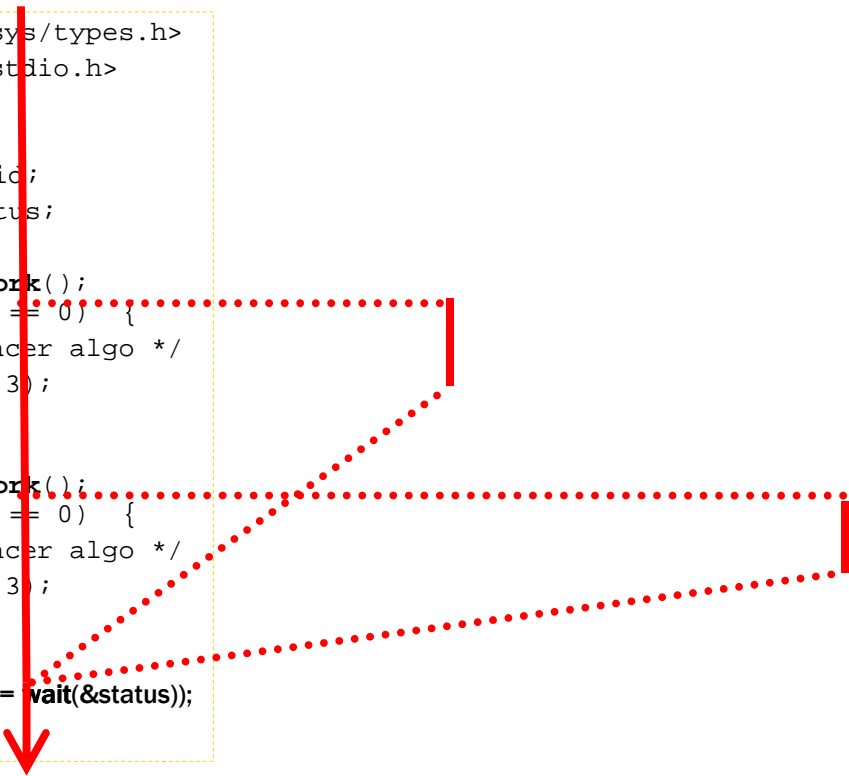
```
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    pid = fork();
    if (pid == 0) {
        /* hacer algo */
        exit(3);
    }

    while (pid != -1) wait(&status);
}
```



Agenda



Linux



Servicios para
procesos



Ejercicios





Parcial del curso 2010-2011

Ejercicio 2 (3 puntos)

- Se desea implementar un proceso que lea un número desde el teclado y comience a escribir una secuencia de números consecutivos a partir del número introducido. Simultáneamente deberá seguir leyendo del teclado y en el momento en que se produzca otra entrada cambiará la secuencia de números que aparecen por pantalla.



Parcial del curso 2010-2011

Ejercicio 2 (3 puntos)

- Un ejemplo de ejecución sería el siguiente:

<pre>\$> miprograma 27 28 29 30 31 32 5 6 7 8 3 4 5 0 Fin del programa \$> -</pre>	<p>← Número introducido por teclado</p> <p>← Número introducido por teclado</p> <p>← Número introducido por teclado</p> <p>← Número introducido por teclado</p>
------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------



Parcial del curso 2010-2011

Ejercicio 2 (3 puntos)

- Puesto que no se pueden simultanear en un mismo proceso la lectura teclado y escritura en la pantalla, se decide realizar un programa con:
 - un proceso padre que continuamente está leyendo por teclado, si la entrada es un número entero lanza un proceso hijo cuya misión consiste en escribir números consecutivos a partir del introducido.
 - Cuando el usuario introduzca otro número el padre mata al proceso que estaba escribiendo y lanza un nuevo hijo con la misma labor.
 - El programa termina cuando el usuario introduce un 0, en este caso el padre mata al proceso en ejecución y termina el programa sacando por pantalla el literal “Fin del programa”



Parcial del curso 2010-2011

Ejercicio 2 (3 puntos)

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

int main ( void )
{
    int  contador;
    pid_t pid;

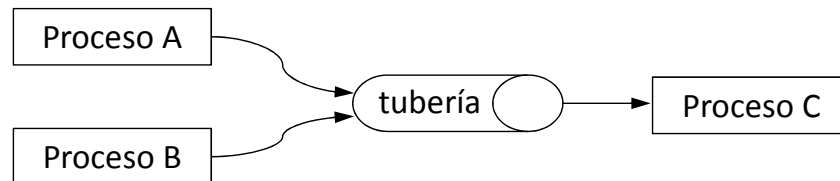
    scanf ("%d",&contador);
    while (contador)
    {
        pid = fork();
        switch (pid)
        {
            case 0:    /* estoy en el hijo */
                        while (1) { printf("%d",++contador ); sleep(1); }
            case -1:    perror("error al ejecutar el fork")
                        exit(-1) ;
            default:    scanf ("%d", &contador);
                        kill(pid,9);
        }
    }
    printf("Fin del programa");
}
```



Final del curso 2008-2009

Ejercicio 5 y 6 (3.5 puntos)

- Escribir una función en C sobre UNIX que cree tres procesos comunicados mediante una tubería, de manera que dos de ellos tengan la salida estándar asociada a la tubería y el otro la entrada estándar. *Argumentos: nombres de programa que deberán ejecutar los tres procesos hijos.*





Final del curso 2008-2009

Ejercicio 5 y 6 (3.5 puntos)

```
#include <stdio.h>

int main( void )
{
    int tuberia[2];
    int pid1, pid2;

    /* el proceso padre, que crea el pipe, será el proceso p1 */
    if (pipe(tuberia) < 0) {
        perror("No se puede crear la tubería") ;
        exit(0);
    }

    /* se crea el proceso p2 */
    switch ((pid1=fork())) {
        case -1: perror("Error al crear el proceso") ;
                /* se cierra el pipe */
                close(tuberia[0]) ;
                close(tuberia[1]) ;
                exit(0) ;
                break ;
    }
```



Final del curso 2008-2009

Ejercicio 5 y 6 (3.5 puntos)

```
case 0: /* proceso hijo, proceso p2 */
    /* se cierra el descriptor de lectura del pipe */
    close(tuberia[0]) ;
    /* aquí iría el código del proceso p2 */
    /* escribiría usando el descriptor tuberia[1] */
    break ;

default: /* el proceso padre crea ahora el proceso p3 */
    switch ((pid2 = fork()) {
        case -1: perror("Error al crear el proceso") ;
            close(tuberia[0]) ;
            close(tuberia[1]) ;
            /* se mata al proceso anterior */
            kill(pid1, SIGKILL) ;
            exit(0) ;

        case 0: /* proceso hijo (p3) lee de la tubería */
            close(tuberia[1]) ;
            /* código del proceso p3 que lee de la tubería */
            break ;

        default: /* el proceso padre (p2) escribe en la tubería */
            close(tuberia[0]) ;
            /* código del proceso p1 que escribe en la tubería */
            break ;
    }
}
```

Sistemas Operativos

sesión 10: procesos

Grado en Ingeniería Informática
Universidad Carlos III de Madrid