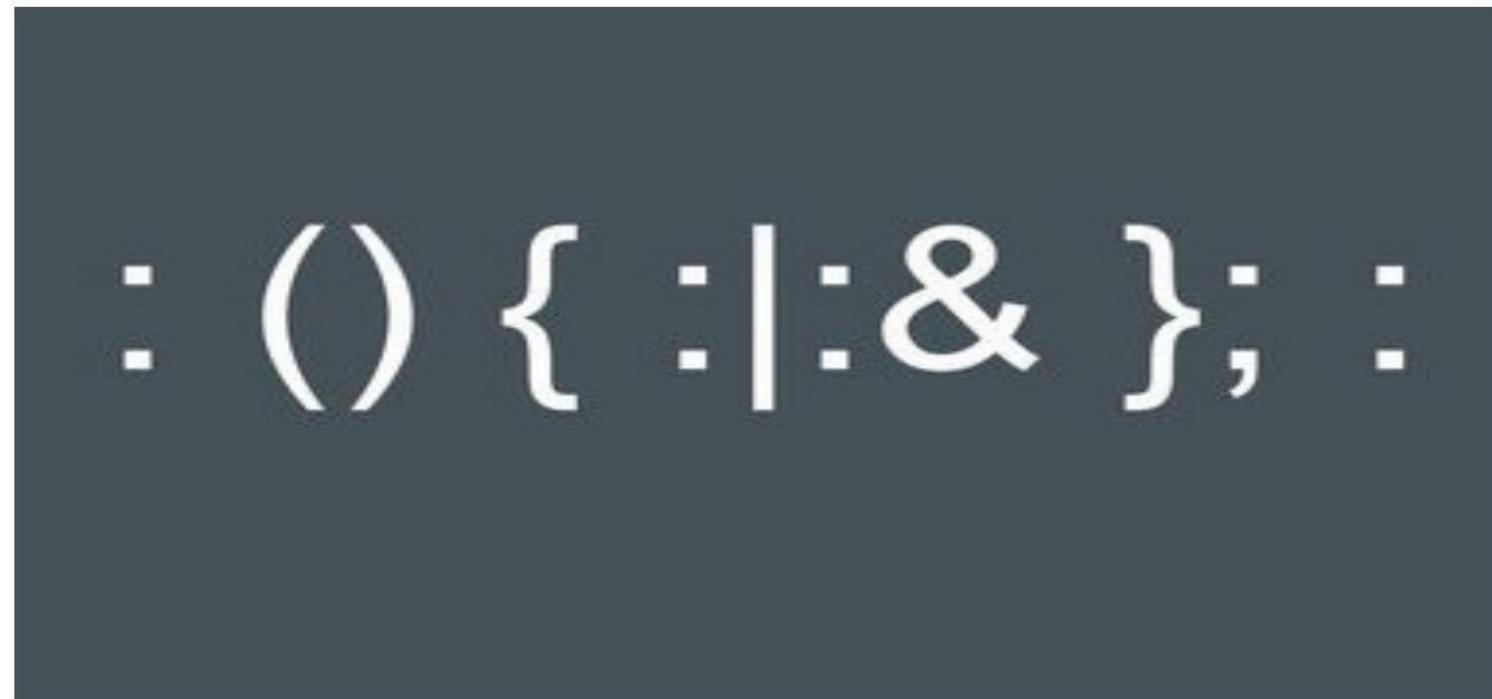


UD01: Programació de processos



0. ÍNDEX

1. Control de processos en Linux
2. Creació i execució de processos
3. Comunicació entre processos (pipes)
4. Sincronització entre processos
5. Creació de processos a Java
6. Redirecció d'entrada i eixida a Java



1. CONTROL DE PROCESSOS EN LINUX

Introducció

Tots els ordinadors actuals fan diverses tasques alhora, per exemple, executar un processador de textos, imprimir un document, visualitzar determinada informació per pantalla, etc... Quan un programa es carrega en memòria per a la seua execució es converteix en un **procés**.

Procés = Programa en execució

En un sistema operatiu multiprocés es pot executar més d'un procés alhora, **donant la sensació** a l'usuari que cada procés és l'únic que s'està executant.

En els sistemes amb 1 única CPU es va alternant l'execució dels processos, és a dir, es lleva un procés de la CPU, s'executa un altre i es torna a col·locar el primer sense que s'assabente de res; aquesta operació es realitza tan ràpid que **sempbla** que cada procés té una dedicació exclusiva.



1. CONTROL DE PROCESSOS EN LINUX

Introducció

Quan un programa usa la CPU i ix, cal fer una espècie de “foto” per a guardar tota la informació referent a aqueix procés amb l'objectiu de rearrancar-lo posteriorment en el mateix estat en el qual trobava quan va eixir de la CPU.

Això es coneix com el **BCP** (bloc de control de processos), que és una estructura de dades que conté informació com **l'identificador del procés**, el seu **estat** i més informació com, per exemple, l'estat dels **registres de la CPU**.

En Linux podem veure informació associada a cada procés teclejant el comando “ps” (process status).

```
Archivo Editar Ver Buscar Terminal
fpv@fpv-VirtualBox:/$ ps
 PID TTY      TIME CMD
 7852 pts/0    00:00:00 bash
 9687 pts/0    00:00:00 ps
fpv@fpv-VirtualBox:/$
```

PID	Identificador del procés
TTY	Terminal associat del qual liig i al qual escriu
TIME	Temps total de CPU usat
CMD	Nom del procés

1. CONTROL DE PROCESSOS EN LINUX

Introducció

També si teclegem “ps –f” també apareixerà el PPID, l'identificador del procés pare.

```
fpv@fpv-VirtualBox:/$ ps
    PID  TTY      TIME CMD
  7852  pts/0    00:00:00 bash
  9687  pts/0    00:00:00 ps
fpv@fpv-VirtualBox:/$ ps -f
UID      PID  PPID  C STIME TTY          TIME CMD
fpv      7852  7842  0 21:51 pts/0    00:00:00 bash
fpv      9727  7852  0 22:49 pts/0    00:00:00 ps -f
fpv@fpv-VirtualBox:/$ █
```

Fixa't com el procés que llança la instrucció “ps –f”, en crear-se dins del bash, es converteix en el seu “procés fill” i així es reflecteix en el resultat de la instrucció. Dit d'una altra forma, el procés “ps –f” té com **PPID** el 7852, que és el **PID** del bash.



1. CONTROL DE PROCESSOS EN LINUX

Introducció

En Windows podem usar des de la línia de comandos l'ordre “tasklist” per a veure els processos que s'estan executant:

```
 Símbolo del sistema
Microsoft Windows [Versión 10.0.19041.1165]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\franp>tasklist

Nombre de imagen          PID Nombre de sesión Núm. de ses Uso de memor
=====
System Idle Process          0 Services                  0      8 KB
System                         4 Services                  0    4.200 KB
Registry                      124 Services                 0   26.196 KB
smss.exe                       520 Services                 0     404 KB
csrss.exe                      764 Services                 0   2.096 KB
wininit.exe                     852 Services                 0   1.464 KB
csrss.exe                      860 Console                  1   3.204 KB
services.exe                    924 Services                 0   8.080 KB
lsass.exe                       944 Services                 0  13.824 KB
svchost.exe                     592 Services                 0  19.152 KB
Fontdrvhost.exe                  768 Services                 0  10.660 KB
WUDFHost.exe                     956 Services                 0   5.552 KB
svchost.exe                     1072 Services                0  12.372 KB
svchost.exe                     1120 Services                0   4.108 KB
winlogon.exe                     1204 Console                  1   2.912 KB
Fontdrvhost.exe                  1264 Console                  1  16.852 KB
dwm.exe                          1336 Console                  1  94.884 KB
```

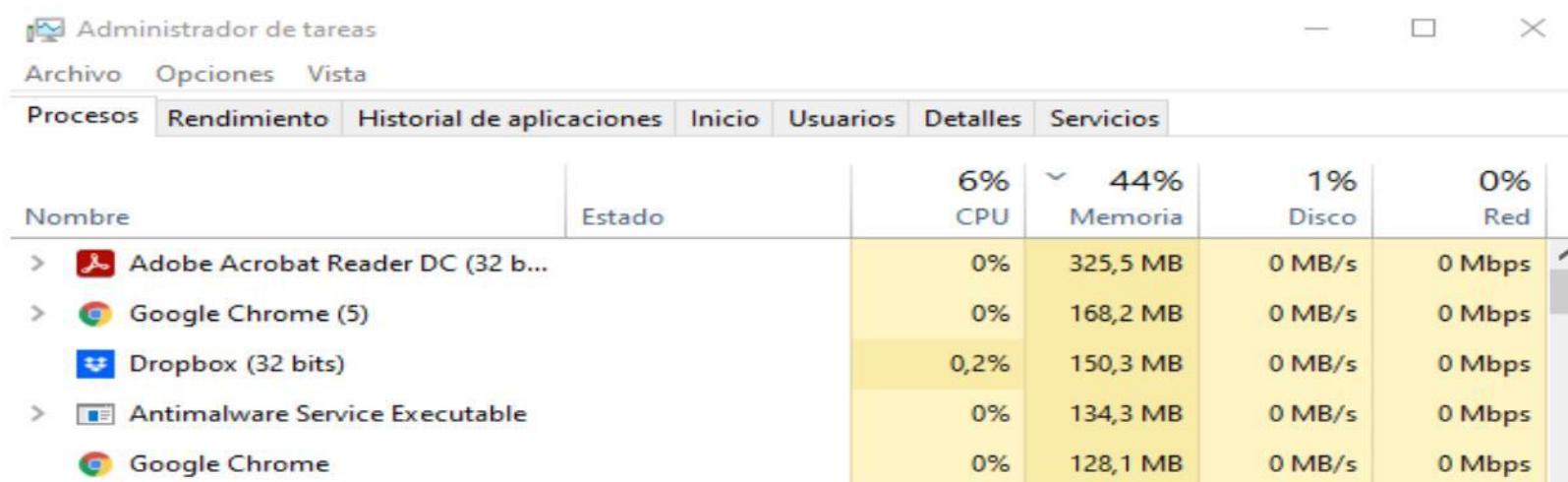


1. CONTROL DE PROCESSOS EN LINUX

Introducció

Encara que en Windows també es pot teclejar **[CTRL+Alt+Supr]** perquè es mostre per pantalla de l'Administrador de tasques.

Una vegada obert l'Administrador de tasques, podem consultar els programes en execució fent clic en la pestanya “Processos”:



The screenshot shows the Windows Task Manager window with the title "Administrador de tareas". The menu bar includes "Archivo", "Opciones", and "Vista". The "Procesos" tab is selected, displaying a list of running processes. The columns show the process name, state, CPU usage, memory usage, disk usage, and network usage. The processes listed are Adobe Acrobat Reader DC, Google Chrome (5 instances), Dropbox (32 bits), Antimalware Service Executable, and another Google Chrome instance.

Nombre	Estado	6% CPU	44% Memoria	1% Disco	0% Red
> Adobe Acrobat Reader DC (32 b...)		0%	325,5 MB	0 MB/s	0 Mbps
> Google Chrome (5)		0%	168,2 MB	0 MB/s	0 Mbps
Dropbox (32 bits)		0,2%	150,3 MB	0 MB/s	0 Mbps
> Antimalware Service Executable		0%	134,3 MB	0 MB/s	0 Mbps
Google Chrome		0%	128,1 MB	0 MB/s	0 Mbps

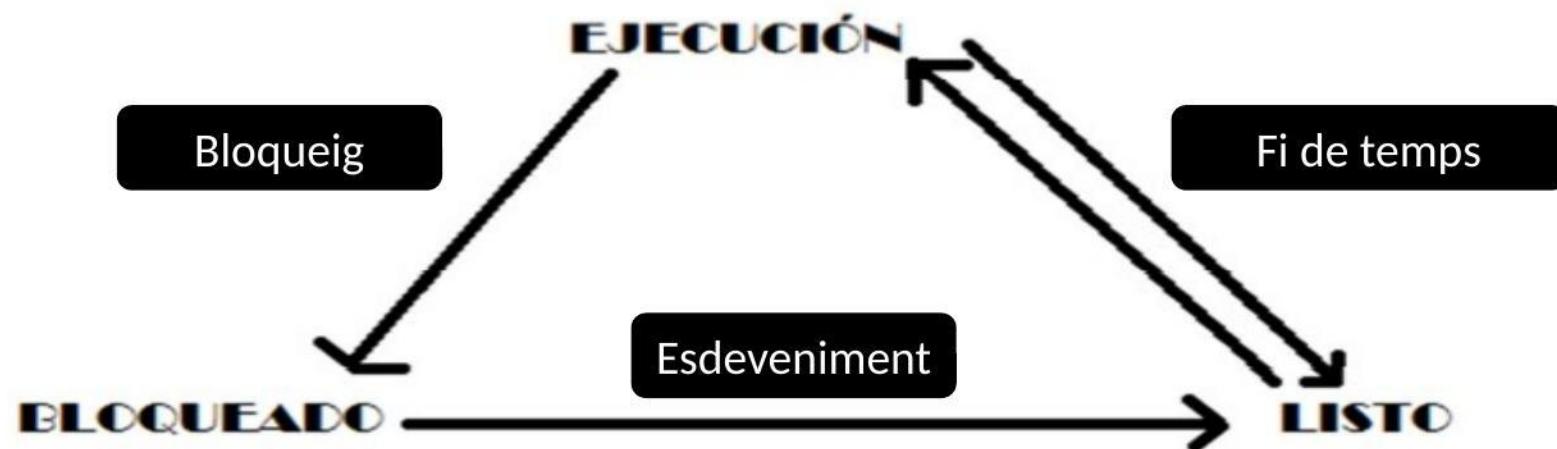
1. CONTROL DE PROCESSOS EN LINUX

Estats d'un procés

Un procés pot estar en la CPU, és a dir, estarà **EN EXECUCIÓ**.

Pot ser que estiga carregat en la RAM, preparat perquè el SO li done pas a la CPU, per la qual cosa estarà **LLEST**.

O pot ser que estiga esperant a utilitzar els recursos que un altre procés ha agafat i, fins que no finalitze el sistema operatiu, no li deixarà continuar. Llavors estarà en estat **BLOQUEJAT**



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Entorn de treball

Per a programar la creació i execució de processos utilitzarem un entorn Linux, **podent treballar amb qualsevol distribució**. Este any treballarem amb Fedora que podeu descarregar des de:

<https://getfedora.org/es/>

Fedora utilitza el paquet "dnf" en lloc de "apt" per a instal·lar programes. La sintaxi i opcions són similars:

\$ sudo dnf upgrade → Actualitza tots els paquets (**NO ÉS NECESSARI FER UN "update" ABANS**)

\$ sudo dnf list geany* → Llista els paquets del repositori que comencen per "geany".

\$ sudo dnf install geany → Instal·la el paquet "geany" des de'l repositori



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Entorn de treball

Una vegada instal·lades les dependències del llenguatge C, instal·la un IDE senzill per a programar, com per exemple, el geany.

Quan s'haja instal·lat l'IDE, es pot llançar teclejant “geany” directament en el terminal, o bé fent clic en la icona de l'aplicació.



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Entorn de treball

Per a provar que tenim l'IDE i les llibreries de C correctament instal·lades farem el típic programa “HolaMon.c”



The image shows a screenshot of a code editor window titled "HolaMundo.c". The code editor displays the following C program:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hola mundo\n");
6     return 0;
7 }
8
```

The code is color-coded: comments are in grey, keywords like `#include`, `int`, `main`, `printf`, and `return` are in blue, and strings are in orange. Line numbers are shown on the left side of the editor.

2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Entorn de treball

Compilarem des del terminal amb la següent instrucció, que utilitza el comando “gcc” i que genera, a partir del codi “HolaMon.c” l’objecte “HolaMon” utilitzant l’opció “-o”:

```
fpv@fpv-VirtualBox:~$ gcc HolaMundo.c -o HolaMundo
```

I executarem l’objecte “HolaMon” que hem generat després de la compilació amb “./”, retornant per pantalla el missatge “Hola món” de la següent manera:

```
fpv@fpv-VirtualBox:~$ ./HolaMundo
Hola mundo
```

2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Aproximació al Llenguatge C

Com en aquest tema treballarem una mica amb el Llenguatge C, us he deixat a l'Aula Virtual en pdf el llibre de “**Llenguatge C**” d'Enrique Bonet, professor d'informàtica de la Universitat de València. Aquest llibre us pot ajudar en cas que vulgueu aprofundir en l'aprenentatge d'aquest llenguatge que, segons l'Índex TIOBE, és el més utilitzat a tot el món.

D'altra banda, us deixo un breu butlletí d'activitats del llenguatge C perquè us familiaritzeu amb la seua sintaxi abans de començar a treballar amb processos. Si teniu dubtes, podeu anar consultant el llibre de “**Llenguatge C**”.



UD01_01_Primers_exercicis_en_C



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Aproximació al Llenguatge C (Solució magic.c)

```
#include <stdio.h>

void main(){
    int num = -1;
    int intento = 0;

    //1A PARTE - EL JUGADOR 1 INTRODUCE UN NUMERO PARA ADIVINAR.
    while ((num < 1) || (num > 100)){
        printf("JUGADOR1: Introduzca un numero (1-100) para adivinarlo:\n");
        scanf("%d",&num);
        if ((num < 1) || (num > 100)){
            printf("El numero %d no está incluido en el rango 1-100\n",num);
        }
    }

    //2A PARTE - EL JUGADOR 2 INTENTARÁ EN BUCLE ADIVINAR EL NÚMERO.
    while (intento != num){
        printf("JUGADOR2: Introduzca un numero (1-100):\n");
        scanf("%d",&intento);
        if (intento == num){
            printf("¡Enhorabuena, ya has adivinado el numero!\n");
        }else{
            if (intento > num){
                printf ("El numero es inferior. Sigue intentándolo...\n");
            }else{
                printf ("El numero es superior. Sigue intentándolo...\n");
            }
        }
    }
}
```



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Aproximació al Llenguatge C (Solució vector.c)

```
#include <stdio.h>

void main(){
    int vector[5];
    int ant = -1;

    //Bucle donde el usuario va introduciendo numeros
    for (int i=0; i<5; i++){
        do{
            printf("Introduzca elemento de la posición %d:",(i+1));
            scanf("%d",&vector[i]);
            if (ant >= vector[i]){
                printf("El numero %d no es mayor que %d!\n",vector[i],ant);
            }
        }
        while (ant >= vector[i]);
        ant = vector[i];
    }

    //Finalizamos imprimiendo el vector con el formato solicitado
    printf("El vector introducido es el siguiente: [ ");
    for (int i=0; i<5; i++){
        if (i != 4){
            printf("%d, ",vector[i]);
        }else{
            printf("%d ]\n",vector[i]);
        }
    }
}
```



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

exec()

La funció “exec()” realitza l'execució i terminació d'un procés, per al que necessita la llibreria **<unistd.h>**. En el moment que cridem a “exec()”, **substituirem** el procés que crida per un nou. Tindrà la següent sintaxi:

```
exec(ruta i nom del programa, arguments, (char *)NULL)
```

Per exemple, si volguérem executar un “ls –l” per a llistar tots els arxius d'un directori, el faríem de la següent manera:

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

void main(){
    printf("Esto es un ejemplo de exec():");
    printf("Los archivos en el directorio son:\n");
    execl("/bin/ls","ls","-l",(char *)NULL);
}
```



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

system

El comando anterior execl està una mica limitat per a executar accions del sistema, per la qual cosa el comú en cas de voler executar accions una mica més avançades és llançar el comando “system”, que necessitarà `<stdio.h>` i `<stdlib.h>` per a funcionar.

```
system(comando de consola a executar)
```

És bastant comú executar-ho en un “printf” per a recollir el resultat de l'anomenada a la funció, que és de tipus sencer, d'ací el “%d”:

```
printf("%d",system(comando de consola a executar))
```

Retorna un 0 si tot ha funcionat bé i el valor -1 si ha existit algun error en l'execució del comando de consola.



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

system

Realitzarem una activitat molt senzilla llançant “system” de la següent manera: es pretén crear un arxiu “llistat.txt” amb el llistat de fitxers i directoris que té el directori actual.

Aquest llistat ha d'aparéixer formatat amb 2 columnes:

- En la primera anirà l’“índex” del fitxer en disc (nombre enter de 6 xifres)
- En la segona el nom del fitxer o directori.

Una vegada generat el llistat, el mateix programa llançarà l'aplicació “geany” per a poder visualitzar-ho.



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

system (Solució)

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

void main(){
    printf("Ejemplo de uso de system");
    printf("%d",system("ls --format=single-column -i > listado.txt"));
    printf("%d",system("geany listado.txt"));
    printf("Fin de programa...");
}
```

listado.txt	
1	274437 Descargas
2	274443 Documentos
3	274436 Escritorio
4	142521 examples.desktop
5	152993 fichero.txt
6	146469 HolaMundo
7	147508 HolaMundo.c
8	274445 Imágenes
9	152994 listado.txt
10	147509 magico
11	147512 magico.c
12	274444 Música
13	274438 Plantillas
14	167748 prueba.txt
15	274442 Público

2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Obtenció de PID

Per a obtindre l'identificador d'un procés o PID usarem 2 funcions que retornen un tipus de dada **pid_t**:

getpid()	Retorna l'identificador del procés que realitza la crida.
getppid()	Retorna l'identificador del procés pare del procés actual.

Crea un programa que obtinga el pid del procés actual i del procés pare. Una vegada ho tingues, executa el comando “ps” en el terminal per a veure els processos que s'estan executant, quina coincidència veus?



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Obtenció de PID (Solució)

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

void main(){
    pid_t id_actual, id_padre;

    id_actual = getpid();
    id_padre = getppid();

    printf("PID de este proceso es: %d\n", id_actual);
    printf("PID del proceso padre es: %d\n", id_padre);
}
```

La “coincidència” és que el procés pare del procés que estem executant és el “bash” de Linux, ja que ha sigut ell qui ha creat el procés:

```
fpv@fpv-VirtualBox:~$ gcc uso_pid.c -o uso_pid
fpv@fpv-VirtualBox:~$ ./uso_pid
PID de este proceso es: 1617
PID del proceso padre es: 1592
fpv@fpv-VirtualBox:~$ ps
  PID TTY          TIME CMD
 1592 pts/1    00:00:00 bash
 1618 pts/1    00:00:00 ps
```

2. CREACIÓ I EXECUCIÓ DE PROCESSOS

fork

El comando fork s'encarrega de crear un nou procés. En cridar a aquesta funció, es crea un procés fill, que és una còpia exacta en codi i dades del procés que ha realitzat la crida (el procés pare), excepte el PID i la memòria que ocupen.

Les variables del procés fill són una còpia de les variables del procés pare, per la qual cosa modificar una variable en un dels processos no es reflecteix en l'altre. L'anomenada a fork es realitzarà de la següent manera, retornant un tipus de dada **pid_t**:

```
pid_t resultat = fork()
```

Els valors retornats són els següents:

-1	Si s'ha produït error
0	Si estem al procés fill
PID del fill	Si estem al procés pare



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

wait

Si estem en un procés pare, creem un fill amb fork() i si volem esperar que el fill finalitze per a continuar, hem de bloquejar al procés pare.

Justament d'això s'encarrega la instrucció “wait”, de fer que el procés pare espere a la finalització del procés fill. Si ho trobes una cosa complicada, pensa en el **pare que està esperant a la porta del col·legi al fet que isca el seu fill**, de manera que el pare espera al fill i mai a l'inrevés.

La instrucció retornarà l'identificador del procés fill (de tipus pid_t) l'execució de la qual ha finalitzat i es codificarà la majoria de vegades de la següent manera:

```
pid_t id_fill = wait(NULL)
```



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Ús de fork i wait

Després de veure tots els comandos i funcions relacionats amb la creació i execució de processos, és moment de dur a terme 2 senzills programes per a demostrar tot l'aprés.



UD01_02_Ús de fork i wait



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Ús de fork i wait (Solució exemple1Fork.c)

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

void main(){
    pid_t pid, hijo_pid;
    pid = fork();

    if (pid == -1){      //Ha ocurrido un error
        printf("No se ha podido crear el proceso hijo...\n");
        exit(-1);
    }

    if (pid == 0){      //Nos encontramos en el proceso hijo
        printf("Soy el proceso hijo, mi PID es %d y el PID"
               " de mi padre es %d\n",getpid(),getppid());
    }
    else{      //Nos encontramos en el proceso padre
        hijo_pid = wait(NULL);
        printf("Soy el proceso padre, mi PID es %d,"
               "el PID de mi padre es %d y mi"
               " hijo %d terminó\n",getpid(),getppid(),hijo_pid);
    }
}
```



2. CREACIÓ I EXECUCIÓ DE PROCESSOS

Ús de fork i wait (Solució activitat1.c)

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

void main(){
    pid_t pid, hijo_pid;
    int var = 6;
    pid = fork();

    if (pid == -1){      //Ha ocurrido un error
        printf("No se ha podido crear el proceso hijo...\n");
        exit(-1);
    }

    if (pid == 0){      //Nos encontramos en el proceso hijo
        var = var - 5;
        printf("Variable en Proceso Hijo: %d\n",var);
    }
    else{               //Nos encontramos en el proceso padre
        printf("El valor inicial de la variable es: %d\n",var);
        hijo_pid = wait(NULL);
        var = var + 5;
        printf("Variable en Proceso Padre: %d\n",var);
    }
}
```



3. COMUNICACIÓ ENTRE PROCESSOS

Pipes i FIFOs

Els pipes amb nom o FIFOs (First IN First OUT) permeten comunicar processos que no tenen perquè estar emparentats (pare-fill)

Un FIFO és com un fitxer amb nom que existeix en el sistema de fitxers i que poden obrir, llegir i escriure múltiples processos. Les dades escriptes es lligen com en una cua, primer a entrar (FIRST IN), primer a eixir (FIRST OUT) i a més:

Una operació de lectura en un FIFO queda en espera fins que el procés pertinent òrbita el FIFO per a iniciar l'escriptura.

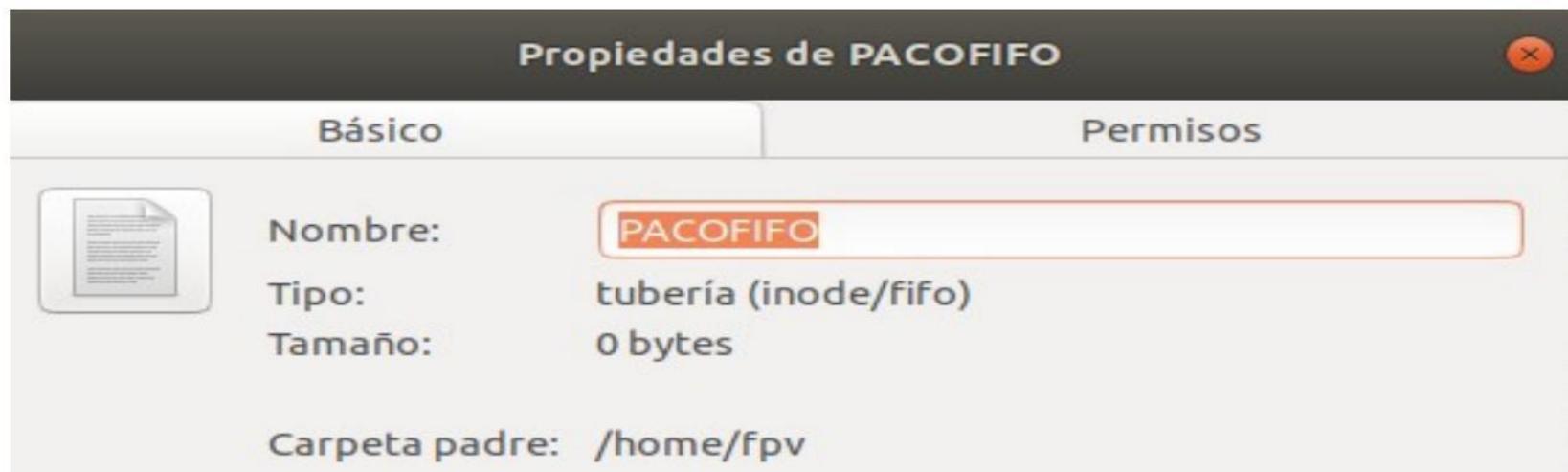


3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

Realitzarem de manera senzilla una FIFO en el terminal executant "mkfifo" així:

```
fpv@fpv-VirtualBox:~$ mkfifo PACOFIFO  
fpv@fpv-VirtualBox:~$ l -l PACOFIFO  
prw-rw-r-- 1 fpv fpv 0 sep 6 22:56 PACOFIFO |
```



3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

Vegem com funciona el FIFO. Execute des de línia de comandos l'ordre "cat" així:

```
fpv@fpv-VirtualBox: ~
Archivo Editar Ver Buscar Terminal Ayuda
fpv@fpv-VirtualBox:~$ cat PACOFIFO
```

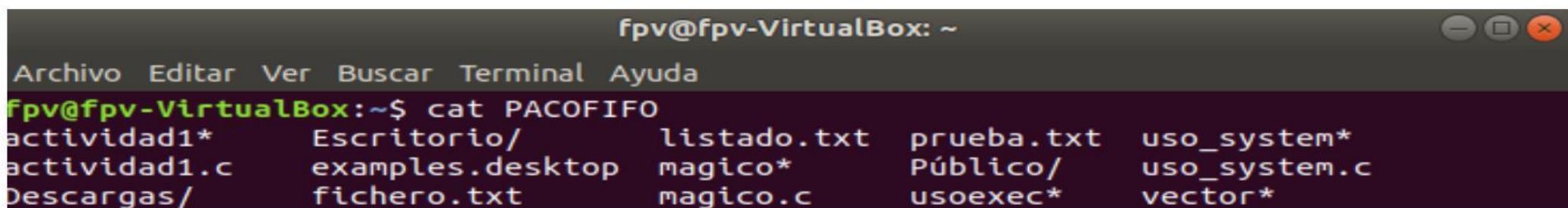
Veurem que es queda a l'espera. Òbric una nova terminal i execute "ls > PACOFIFO":

```
fpv@fpv-VirtualBox: ~
Archivo Editar Ver Buscar Terminal Ayuda
fpv@fpv-VirtualBox:~$ cat PACOFIFO
fpv@fpv-VirtualBox: ~
Archivo Editar Ver Buscar Terminal Ayuda
fpv@fpv-VirtualBox:~$ ls > PACOFIFO
```

3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

Veurem que el cat que anteriorment estava a l'espera, s'executa ja que ha rebut la informació que necessitava:



```
fpv@fpv-VirtualBox: ~
Archivo Editar Ver Buscar Terminal Ayuda
fpv@fpv-VirtualBox:~$ cat PACOFIFO
actividad1*      Escritorio/      listado.txt  prueba.txt  uso_system*
actividad1.c      examples.desktop  magico*       Público/    uso_system.c
Descargas/        fichero.txt     magico.c     usoexec*   vector*
```

Per a crear un FIFO en C utilitzem la instrucció mkfifo(). El seu format és el següent:



3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

Per a **obrir** un FIFO farem:

Identificador del FIFO

```
int open(nom_fifo, mode)
```

0: L
1: E
2: L/E

Per a **llegir** d'un FIFO farem la següent instrucció, que intenta llegir “numbytes” de l'identificador “id_fifo” per a guardar-los en el “buffer” (que moltes vegades serà un array de caràcters, p ex: char buffer[10]. Retornarà el nombre de bytes llegits, per la qual cosa, si comparem aquesta variable amb “numbytes”, podem saber si s'han aconseguit llegir tants bytes com es demanaven.

Nombre de bytes llegits

```
int read(id_fifo, buffer, numbytes)
```



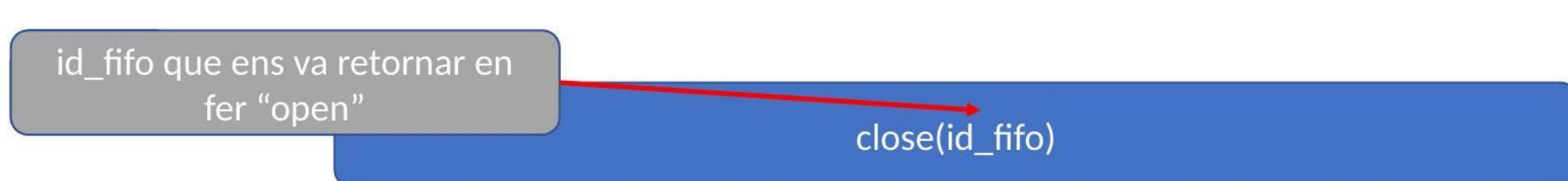
3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

Per a **escriure** és molt similar a llegir. Al “buffer” li donem el valor del que vulguem escriure, definim la seu grandària en “numbytes” i especificarem el FIFO on escriurem en “nom_fifo”.



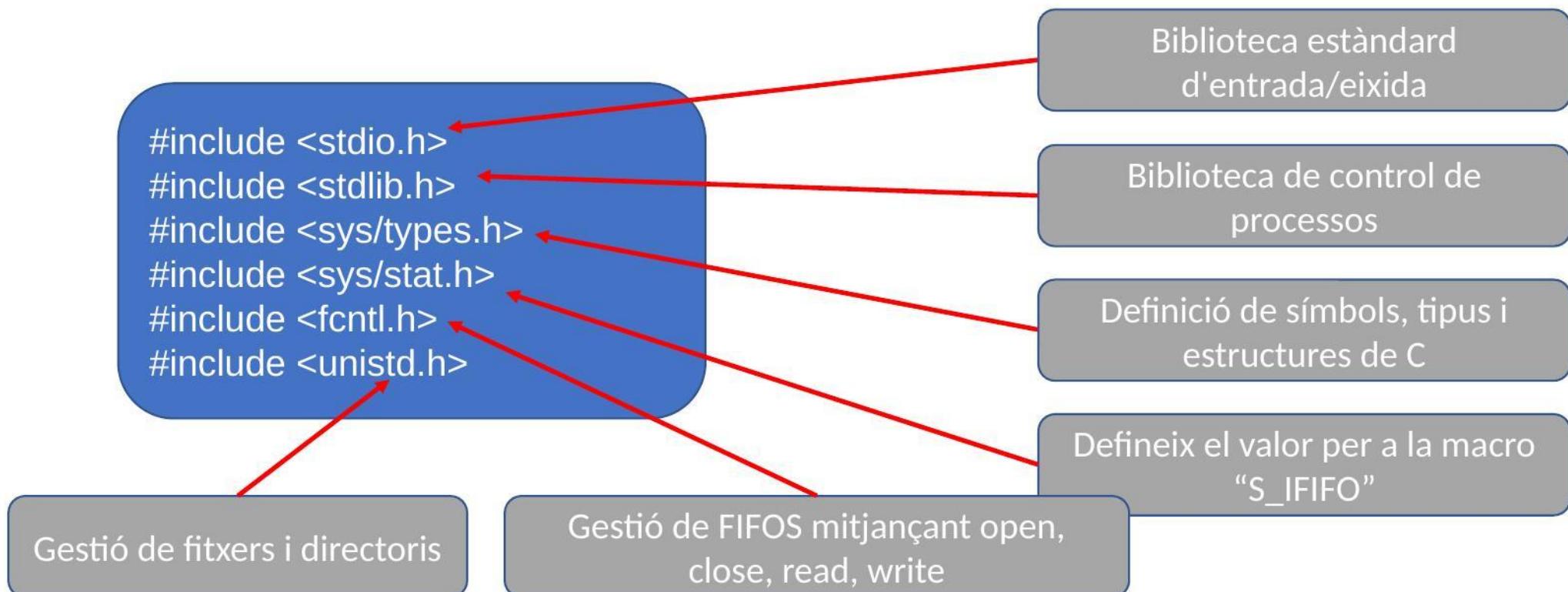
Per a **tancar** un FIFO, executarem la següent instrucció:



3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

En qualsevol dels casos, un programa que gestione pipes o FIFOs hauria de fer referència a les següents llibreries:



3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

Realitzarem a continuació una “pràctica guiada” en la qual posarem en comunicació 2 processos utilitzant les instruccions que hem estudiat en les diapositives anteriors.



UD01_03_Pràctica guiada comunicació



3. COMUNICACIÓ ENTRE PROCESSOS

Pipes o FIFOs

A continuació, realitzarem un breu butlletí d'exercicis que consta de 2 activitats en les quals haurem de posar en comunicació 2 processos a través dels mecanismes que ofereixen les FIFOs



UD01_04_FIFOs



4. SINCRONITZACIÓ ENTRE PROCESSOS

Ús de signal

Perquè els processos interactuen els uns amb els altres, necessiten cert nivell de sincronització, és a dir, és necessari que hi haja un funcionament coordinat entre els processos a l'hora d'executar alguna tasca. Per a poder completar correctament aquesta labor de sincronització utilitzarem **senyals**.

Un senyal és com un avís que un procés mana a un altre procés. En llenguatge C utilitzarem **signal()** per a especificar l'acció que ha de realitzar-se quan un procés rep un senyal. Haurem d'incloure la llibreria **<signal.h>**

Com existeixen molts tipus de senyals, ho simplificarem perquè sempre el tipus de senyal siga “SIGUSR1”, que C interpreta com a “senyal definit per l'usuari”:

Quan el procés reba un senyal SIGUSR1, es realitza una anomenada a la funció gestio_pare()

```
signal(SIGUSR1,gestio_pare);
```

4. SINCRONITZACIÓ ENTRE PROCESSOS

Ús de kill

La funció signal() **NO ENVIA UNA SENYAL**, simplement indica el comportament a seguir en cas de rebre-la. Per a enviar el senyal, utilitzem kill().

Envia un senyal SIGUSR1 al procés identificat amb el pid especificat

```
kill(pid,SIGUSR1);
```

Ús de sleep

La funció sleep() suspén el procés que realitza la crida la quantitat de segons indicada abans de continuar i ens ajuda a veure com s'estan sincronitzant els processos. Per exemple, si volem fer que el procés s'espere 1 segon:

```
sleep(1)
```

4. SINCRONITZACIÓ ENTRE PROCESSOS

Ús de pause

Dins de la sincronització, és interessant provocar que un procés es pause, és a dir, que s'espere a rebre un senyal. És freqüent fer que, per exemple, el pare s'espere a rebre un senyal del fill i a l'inrevés. La sintaxi és molt senzilla:

```
pause()
```

No confondre amb wait(NULL) del principi de la UD1 en el qual el pare esperava al fill:

- En el cas del wait, s'està esperant que **el fill finalitze la seu execució.**
- En el cas del pause, no estem dient que el procés finalitze, simplement ens quedem **esperant** “que ens avisen” amb un senyal que podem continuar amb l'execució.



4. SINCRONITZACIÓ ENTRE PROCESSOS

Ús de signal, kill, sleep i pause

A continuació, realitzarem una pràctica guiada que consistirà en 2 exercicis dissenyats per a posar a posar en pràctica els conceptes apresos anteriorment relacionats amb la sincronització de processos. Aquests conceptes són les funcions **signal** (que diu què fer en cas de rebre un senyal), **kill** (que envia el senyal), **sleep** (que suspén al procés que realitza la crida) i **pause** (que fa esperar a un procés fins que arriba un senyal).



UD01_05_Pràctica guiada sincronització



5. CREACIÓ DE PROCESSOS A JAVA

Entorn de treball

Per a treballar amb Java es recomana instal·lar la opendjk versió 11 i el IDE Geany.

En qualsevol cas, podem instal·lar tant Java com Geany des de la terminal de Linux amb aquests 2 comandos:

```
$ sudo dnf install geany  
$ sudo dnf install java-11-openjdk java-11-openjdk-devel
```

Per a comprovar si està tot correctament instal·lat podem fer-ho amb:

```
$ java --version  
$ javac -version
```

5. CREACIÓ DE PROCESSOS A JAVA

Entorn de treball

Per a escriure el programa, podem utilitzar Geany, però per a compilar utilitzarem el terminal de la següent manera:

```
$ javac -encoding UTF-8 servidor.java
```

I per a executar el programa:

```
$ java Servidor
```

5. CREACIÓ DE PROCESSOS A JAVA

Classes ProcessBuilder i Process

Java disposa de diverses classes per a la gestió de processos:

- **ProcessBuilder**: que defineix el procés que es vol llançar.
- **Process**: en invocar al mètode “start” de ProcessBuilder, es genera un objecte de tipus Process, que es correspon amb un procés en execució.

Per exemple, el següent codi llançaria l'aplicació “geany”:

```
Principal.java
1 import java.io.*;
2
3 public class Principal {
4
5     public static void main(String[] args) throws IOException {
6         ProcessBuilder pb = new ProcessBuilder("geany");
7         Process p = pb.start();
8
9     }
10
11 }
```

Versió
“Compactada”

Process p = new ProcessBuilder("geany").start();



5. CREACIÓ DE PROCESSOS A JAVA

Classes ProcessBuilder i Process

De fet, i d'una manera genèrica, l'ús del comando ProcessBuilder admet tots els arguments que necessitem separats per comes.

```
Process p = new ProcessBuilder("Comando", "Arg1", "Arg2"...."Argn").start();
```

Com per exemple:

```
public class Principal {  
  
    public static void main(String[] args) throws IOException {  
        ProcessBuilder pb = new ProcessBuilder("/usr/bin/firefox", "www.superdeporte.es");  
        Process p = pb.start();  
    }  
}
```



6. REDIRECCIÓ DE E/S A JAVA

Redirigint l'eixida amb getInputStream

El següent programa executa el comando “ls” dins d'un directori. Usarem el mètode “getInputStream” per a llegir l'stream d'eixida del procés, és a dir, per a llegir el que el comando “ls” envia a la consola.

```
public static void main(String[] args) throws IOException {
    ProcessBuilder pb = new ProcessBuilder("bash", "-c", "ls /home/fpv");
    Process p = pb.start();

    try {
        InputStream is = p.getInputStream();
        int c;
        while ((c = is.read()) != -1)
            System.out.print((char)c);
        is.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

D'aquest procés, queda't amb el que s'envia a la consola en l'stream “ls”

El mètode “read” retorna el següent byte d'informació que, si fem un “casting” a un char, ens retorna el caràcter que s'està enviant.



6. REDIRECCIÓ DE E/S A JAVA

Redirigint l'eixida amb getInputStream

Quan executem el programa anterior, el resultat d'executar el comando “ls” es converteix en l'entrada del nostre procés, per la qual cosa som capaços d'escriure'l en la consola:



The screenshot shows an IDE interface with a "Console" tab selected. The output of the "ls" command is displayed, listing files such as sincro1.c, sincro2, sincro2.c, snap, usoexec, usoexec.c, uso_pid, uso_pid.c, and uso_system.

Opcionalment, podríem demanar-li al procés actual que espere que finalitze el que hem creat (procés ls en aquest cas), per a **comprovar** que ha acabat bé.

```
int exitVal;
try {
    exitVal = p.waitFor();
    System.out.println("Valor de salida: " + exitVal);
}
catch (InterruptedException e) {
    e.printStackTrace();
}
```

0: bé
1: mal

Valor de salida: 0

6. REDIRECCIÓ DE E/S A JAVA

Escrivint en l'entrada amb getOutputStream

Suposem que tenim un programa Java que llig una cadena des de l'entrada estàndard i la visualitza:

```
public class EjemploLectura {  
  
    public static void main(String[] args) {  
        InputStreamReader in = new InputStreamReader(System.in);  
        BufferedReader br = new BufferedReader(in);  
        String texto;  
        try {  
            System.out.println("Introduce una cadena...");  
            texto = br.readLine();  
            System.out.println("Cadena escrita: "+texto);  
            in.close();  
        }catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
<terminated> EjemploLectura [Java Application]  
Introduce una cadena...  
Hola mundo  
Cadena escrita: Hola mundo
```



6. REDIRECCIÓ DE E/S A JAVA

Escrivint en l'entrada amb getOutputStream

Amb el mètode `getOutputStream` podem enviar dades a l'entrada estàndard del programa “EjemploLectura.java” anterior. Per exemple, “Hola soy Paco Pérez\n”

```
public static void main(String[] args) throws IOException {
    File directorio = new File("/home/fpv/eclipse-workspace/Practica4/bin");
    String cadena = "Hola soy Paco Perez\n";
    ProcessBuilder pb = new ProcessBuilder("java", "EjemploLectura");
    pb.directory(directorio);
    Process p = pb.start();

    OutputStream os = p.getOutputStream();
    os.write(cadena.getBytes());
    os.flush();

    try {
        InputStream is = p.getInputStream();
        int c;
        while ((c = is.read()) != -1)
            System.out.print((char)c);
        is.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

Executem “*EjemploLectura” amb
“cadena” en l'entrada

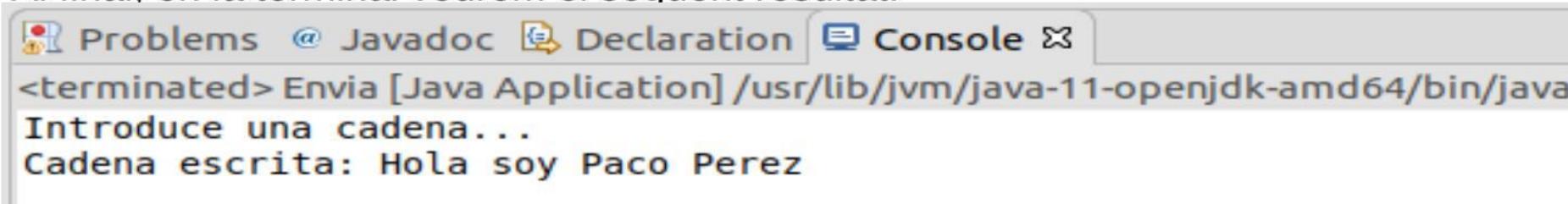
Tornem a fer ús de `getInputStream` per a
recollir el resultat del programa i poder
visualitzar-lo.

6. REDIRECCIÓ DE E/S A JAVA

Escrivint en l'entrada amb getOutputStream

El resultat amb OutputStream és que som capaços d'escriure en l'entrada de “EjemploLectura” amb l'stream que deixa el programa anterior.

Al final, en la terminal veurem el seqüent resultat:



A screenshot of an IDE's Console tab. The tab bar includes 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active. The output window shows the following text:
<terminated> Envia [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java
Introduce una cadena...
Cadena escrita: Hola soy Paco Perez

Si només utilitzem “getOutputStream” i després no executem la instrucció “getInputStream”, el programa finalitzarà correctament **però no serem capaços de veure per pantalla el resultat de l'execució**, d'aquí ve que siga necessari fer ús dels 2 mètodes de redirecció.

6. REDIRECCIÓN DE E/S A JAVA

Processos a Java

A continuació, desenvoluparem uns senzills programes relacionats amb la creació de processos en Java i amb la redirecció de la E/S, per a posar en pràctica els mètodes `getInputStream()` i `getOutputStream()` vistos anteriorment.



UD01_06_Processos a Java

