

Llamadas al sistema: Uso de C y las llamadas al sistema.

Laboratorio en clase

1. Crea un fichero ".c" con cada ejemplo a continuación.
2. Compila cada ejemplo y comprueba su salida.
3. Utiliza la terminal para visualizar la jerarquía de procesos creada en cada ejercicio (utiliza el comando **ps**tree e identifica los PID y PPID de cada proceso, crea una tabla donde señales el proceso de cada ejemplo).

PID	PPID	Ejemplo

4. Modifica la ejecución del comando exec y utiliza todas sus variantes posibles (en el Tema 3.3 tenéis un ejemplo de las diferentes llamadas que existen para esta función en C).

Prepara una memoria con todos los pasos anteriores, utiliza capturas de pantalla para mostrar los resultados.

Ejemplo 1

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

int main(void)
{
    printf("ID de proceso: %ld\n", (long) getpid());
    printf("ID de proceso padre: %ld\n", (long) getppid());
    printf("ID de usuario propietario: %ld\n", (long) getuid());
    return 0;
}
```

Explica qué hace el código anterior.

Ejemplo 2

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

/* This program demonstrates that the use of the getpid() function.
 *
 * When this was run 100 times on the computer the author is
 * on, only twice did the parent process execute before the
 * child process executed.
 * Note, if you juxtapose two strings, the compiler automatically
 * concatenates the two, e.g., "Hello " "world!"
 * The return value of fork() is actually pid_t ('pid' 't'ype).
 * When it is assigned to 'int pid', if the type is different, there
 * is an implicit cast; however, when we print the return value
```



```
* of getpid(), it is necessary to explicitly cast it as an
* integer.
*
* The type 'pid_t' is defined in the library header <sys/types.h>
*/
```

```
int main( void ) {
    printf( "PADRE: Soy el proceso Padre: mi (PID) es %d\n", (int) getpid() );

    int pid = fork();

    if ( pid == 0 ) {
        printf( "HIJO: después de fork()\n");
        printf( "HIJO: soy el proceso Hijo, mi (PID) es %d\n", (int) getpid() );
    } else {
        printf( "PADRE: después de fork()\n");
        printf( "PADRE: el (PID) del Padre sigue siendo %d - fork() retorna %d\n",
                (int) getpid(), pid );
    }

    return 0;
}
```

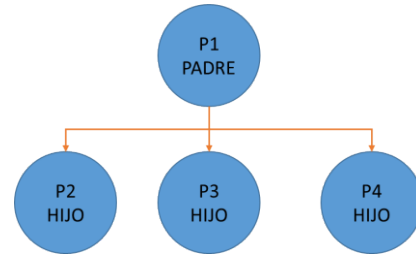
Ejemplo 3

```

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

int main (void) {
    int i;
    int padre;
    padre = 1;
    for (i=0; i < 3; i++) {
        if (padre == 1){
            if (fork() == 0) /* Proceso hijo */
            {
                fprintf(stdout, "Este es el proceso hijo con padre %ld\n",
                (long)getppid());
                padre = 0;
            } else /* Proceso padre */
            {
                fprintf(stdout, "Este es el proceso padre con ID %ld\n",
                (long)getpid());
                padre = 1;
            }
        }
    }
    return 0;
}

```



Explica qué hace el código anterior.

EJEMPLO 3.1

```

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

int main (int argc, char *argv[]) {
    int num;
    pid_t pid;
    srandom(getpid());

    for (num= 0; num< 3; num++) {
        pid= fork();
        printf ("Soy el proceso de PID %d y mi padre es %d (PID).\n",
        getpid(), getppid());

        if (pid== 0)
            break;
    }

    if (pid== 0)
        sleep(random() % 5);
    else
        for (num= 0; num< 3; num++)
            printf ("Fin del proceso de PID %d.\n", wait (NULL));

    return 0;
}

```



Ejemplo 4

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
int main (void) {
    pid_t childpid;
    int status=0;
    int result;
    if ((childpid = fork()) == -1) {
        perror("Error en llamada a fork\n");
        exit(1);
    }
    else if (childpid == 0) {
        result = getpid() < getppid() ;
        fprintf(stdout, "Soy el proceso hijo (%ld) y voy a devolver a mi padre
(%ld) el valor %d después de esperar 2 segundos\n", (long)getpid(),
(long)getppid(), result);
        sleep(2);
        exit (result);
    }
    else {
        while( childpid != wait(&status));
        fprintf(stdout, "Soy el proceso padre (%ld) y mi hijo (%ld) me ha
devuelto %d\n", (long)getpid(), (long)childpid, status);
    }
    return (0);
}
```

Explica qué hace el código anterior.



Ejemplo 5

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

int main() {
    int status;
    printf( "Soy el proceso Padre, mi (PID) es %d\n", (int) getpid() );
    printf ("Listar los procesos \n");
    if (execl ("/bin/ps", "ps", "-f", NULL) < 0) {
        fprintf(stderr, "Error en exec %d\n", errno);
        exit(1);
    }
    printf ("Fin de la lista de procesos\n");
    exit(0);
}
```

Explica qué hace el código anterior.

Confirma el path del comando ps utilizando la instrucción **whereis ps** o **which ps**.

Ejemplo 6 (example_01.3.C y example_01.4.C)

example_01.3.C

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

int main(int argc, char *argv[])
{
    printf("Soy example_01.3.C, mi PID es %d\n", getpid());
    char *args[] = {"example_01.4", "C", "Programming", NULL};
    execv("./example_01.4", args);
    printf("Volviendo a example_01.3.C \n");
    return 0;
}
```

example_01.4.C

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

int main(int argc, char *argv[])
{
    printf("Soy example_01.4.C\n");
    printf("Mi PID es %d\n", getpid());
    return 0;
}
```

Llamad a los ficheros así.
Compila ambos ficheros y luego
ejecuta el primero.

Explica qué hace el código anterior.

Ejemplo 7

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
int main() {
    pid_t childpid, waitreturn;
    int status;
    if ((childpid = fork()) == -1) {
        fprintf(stderr, "Error en fork %d\n", errno);
        exit(1);
    }
    else if (childpid == 0)
    { /* codigo del proceso hijo */
        if ( execl ("/bin/ps", "ps", "-fu", getenv ("USER"), 0) < 0) {
            fprintf(stderr, "Error en exec %d\n", errno);
            exit(1);
        }
    }
    else /* codigo del proceso padre */
        while(childpid != (waitreturn = wait(&status)))
            if ((waitreturn == -1) && (errno != EINTR))
                break;

    exit(0);
}
```

Explica qué hace el código anterior.

Avanzado: comunicación entre procesos con tuberías

Ejemplo 8

El siguiente programa ejecuta el comando **ls -l — sort -n +4** comunicando a los procesos **ls** y **sort** con tuberías:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(void) {
    int fd[2];
    pid_t childpid;
    pipe(fd);
    if ((childpid = fork()) == 0) { /* ls es el hijo */
        dup2(fd[1], STDOUT_FILENO);
        close(fd[0]);
        close(fd[1]);
        execl("/usr/bin/ls", "ls", "-l", NULL);
        perror("Exec fallo en ls");
    } else { /* sort es el padre */
        dup2(fd[0], STDIN_FILENO);
        close(fd[0]);
        close(fd[1]);
        execl("/usr/bin/sort", "sort", "-n", "+4", NULL);
        perror("Exec fallo en sort");
    }
    exit(0);
}
```

Este de momento no lo expliques, solo intenta comprender qué ocurre entre los procesos.