Algunas señales admiten que se las ignore, lo cual ha de ser indicado por el proceso al sistema operativo. En este caso el sistema operativo simplemente desecha las señales ignoradas por ese proceso. Un proceso también puede enmascarar diversos tipos de señales. El efecto es que las señales enmascaradas quedan bloqueadas (no se desechan), a la espera de que el proceso las desenmascare.

Cuando un proceso recibe una señal sin haberla armado o enmascarado previamente, se produce la acción por defecto, que en la mayoría de los casos consiste en matar al proceso.

2.4. SERVICIOS DE PROCESOS

En esta sección se describen los servicios del sistema operativo necesarios para hacer los problemas que se proponen en las secciones siguientes. En el caso de los procesos, los servicios suelen ser restringidos, siendo más extensos para el caso de procesos ligeros.

Esta sección describe los principales servicios que ofrece POSIX para la gestión de procesos, procesos ligeros y planificación. También se presentan los servicios que permiten trabajar con señales y temporizadores

2.4.1. Servicios POSIX para la gestión de procesos

En esta sección se describen los principales servicios que ofrece POSIX para la gestión de procesos. Estos servicios se han agrupado según las siguientes categorías: identificación de procesos, el entorno de un proceso, creación de procesos y terminación de procesos.

Identificación de procesos

POSIX identifica cada proceso por medio de un entero único denominado *identificador de proceso* de tipo `pid_t`. Los servicios relativos a la identificación de los procesos son los siguientes:

Obtener el identificador de proceso

Este servicio devuelve el identificador del proceso que realiza la llamada. Su prototipo en lenguaje C es el siguiente:

```
pid_t getpid(void);
```

Obtener el identificador del proceso padre

Devuelve el identificador del proceso padre. Su prototipo es el que se muestra a continuación.

```
pid_t getppid(void);
```

Obtener el identificador de usuario real

Este servicio devuelve el identificador de usuario real del proceso que realiza la llamada. Su prototipo es:

```
uid_t getuid(void);
```


Obtener el identificador de usuario efectivo

Devuelve el identificador de usuario efectivo. Su prototipo es:

```
uid_t geteuid(void);
```

Obtener el identificador de grupo real

Este servicio permite obtener el identificador de grupo real. El prototipo que se utiliza para invocar este servicio es el siguiente:

```
gid_t getgid(void);
```

Obtener el identificador de grupo efectivo

Devuelve el identificador de grupo efectivo. Su prototipo es:

```
gid_t getegid(void);
```

El entorno de un proceso

El entorno de un proceso viene definido por una lista de variables que se pasan al mismo en el momento de comenzar su ejecución. Estas variables se denominan variables de entorno, y son accesibles a un proceso a través de la variable externa `environ`, declarada de la siguiente forma:

```
extern char **environ;
```

Obtener el valor de una variable de entorno

El servicio **`getenv`** permite buscar una determinada variable de entorno dentro de la lista de variables de entorno de un proceso. La sintaxis de esta función es:

```
char *getenv(const char *name);
```

Definir el entorno de un proceso

El servicio setenv permite fijar el entono de un proceso. La sintaxis de esta función es:

```
char *setenv(char **env);
```

Gestión de procesos

Crear un proceso

La forma de crear un proceso en un sistema operativo que ofrezca la interfaz POSIX es invocando el servicio `fork`. El sistema operativo trata este servicio realizando una clonación del proceso que

lo solicite. El proceso que solicita el servicio se convierte en el proceso padre del nuevo proceso, que es, a su vez, el proceso hijo.

El prototipo de esta función es el siguiente:

```
pid_t fork();
```

Ejecutar un programa

El servicio exec de POSIX tiene por objetivo cambiar el programa que está ejecutando un proceso. En POSIX existe una familia de funciones exec, cuyos prototipos se muestran a continuación:

```
int execl(char *path, char *arg, ...);
int execv(char *path, char *argv[]);
int execle(char *path, char *arg, ...);
int execve(char *path, char *argv[], char *envp[]);
int execlp(char *file, const char *arg, ...);
int execvp(char *file, char *argv[]);
```

Terminar la ejecución de un proceso

Cuando un programa ejecuta dentro de la función `main` la sentencia `return(valor)`, ésta es similar a `exit(valor)`. El prototipo de la función `exit` es:

```
void exit(int status);
```

Esperar por la finalización de un proceso hijo

Permite a un proceso padre esperar hasta que termine la ejecución de un proceso hijo (el proceso padre se queda bloqueado hasta que termina un proceso hijo). Existen dos formas de invocar este servicio:

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

Ambas llamadas esperan la finalización de un proceso hijo y permiten obtener información sobre el estado de terminación del mismo.

Suspender la ejecución de un proceso

Suspende al proceso durante un número determinado de segundos. Su prototipo es:

```
int sleep(unsigned int seconds)
```

El proceso se suspende durante un número de segundos pasado como parámetro. El proceso despierta cuando ha transcurrido el tiempo establecido o cuando se recibe una señal.

2.5. SERVICIOS POSIX DE GESTIÓN DE PROCESOS LIGEROS

En esta sección se describen los principales servicios POSIX relativos a la gestión de procesos ligeros. Estos servicios se han agrupado de acuerdo a las siguientes categorías:

- Atributos de un proceso ligero.
- Creación e identificación de procesos ligeros.
- Terminación de procesos ligeros.

Atributos de un proceso ligero

Cada proceso ligero en POSIX tiene asociado una serie de atributos que representan sus propiedades. Los valores de los diferentes atributos se almacenan en un objeto atributo de tipo `pthread_attr_t`. Existen una serie de servicios que se aplican sobre el tipo anterior y que permiten modificar los valores asociados a un objeto de tipo atributo. A continuación se describen las principales funciones relacionadas con los atributos de los procesos ligeros.

Crear atributos de un proceso ligero

Este servicio permite iniciar un objeto atributo que se puede utilizar para crear nuevos procesos ligeros. El prototipo de esta función es:

```
int pthread_attr_init(pthread_attr_t *attr);
```

Destruir atributos

Destruye el objeto de tipo atributo pasado como argumento a la misma. Su prototipo es:

```
int pthread_attr_destroy(pthread_attr_t*attr);
```

Asignar el tamaño de la pila

Cada proceso ligero tiene una pila cuyo tamaño se puede establecer mediante esta función cuyo prototipo es el siguiente:

```
int pthread_attr_setstacksize(pthread_attr_t*attr, int stacksize);
```

Obtener el tamaño de la pila

El prototipo del servicio que permite obtener el tamaño de la pila de un proceso es:

```
int pthread_attr_getstacksize(pthread_attr_t*attr, int stacksize);
```

Establecer el estado de terminación

El prototipo de este servicio es:

```
int pthread_attr_setdetachstate(pthread_attr_t*attr,
int detachstate);
```