Nombre y Apellidos:_____

Cuestión 1. (2 Puntos)

En el siguiente fragmento de código, un proceso padre crea un proceso hijo y espera a su terminación para informar de su estado.

```
#include ...
int status;
...
int main(int argc, char** argv) {
	int pid;
	...
	pid = fork();
	if (pid==0) {
		status=do_something();
		exit(status);
	}
	// esperar a que el proceso hijo finalice o cambie de estado
	...
	printf ("Fin proceso hijo %d, codigo de terminación: %d\n", pid, status);
}
```

La forma tradicional de implementar esta funcionalidad sería usando la llamada **`waitpid`** o **`wait`** antes del **`printf`** que se indica en el código.

¿Sería posible una implmeentación alternativa para el código anterior sin necesidad de llamar explícitamente a **`waitpid`** o **`wait`** dentro del **`main`** (ni tampoco llamando explícitamente a otra función que invocase directamente a alguna de estas llamadas)?. Razonar la respuesta y en caso afirmativo indicar como completarías el código.

Nombre y Apellidos:_____

Cuestión 2. (1 Punto)
Cuando hablamos de señales en Linux o en Unix a veces indicamos que es necesario bloquear una determinada señal y otras veces hablamos de que es necesario ignorarla. ¿Es equivalente? Razonar la respuesta. Puede indicarse algún ejemplo para ilustrar la respuesta.

Cuestión 3. (1 Punto)

¿Puedes describir que es lo que hace el siguiente fragmento de código e indicar si es correcto? Razonar la respuesta y en caso de que se incluya algún error indicar las modificaciones que sería necesario incluir en el mismo.

```
#include …
int main(int argc, char** argv) {
	fd_set readfds;
	FD_ZERO(&readfds);
	FD_SET(STDIN_FILENO, &readfds);
	int ready = select(STDIN_FILENO, &readfds, NULL, NULL, NULL);
	if (FD_ISSET(STDIN_FILENO, &readfds)) {
		printf("Entrada disponible en la entrada estándar.\n");
	}
	...
	return 0;
}
```

Nombre y Apellidos:_____

Ejercicio 1. (2 Puntos)

El comando **grep** es una utilidad muy popular de Unix que sirve para buscar cadenas de texto. El comando **cut** es otro comando de uso frecuente que permite cortar secciones de lineas de texto. Ambos se suelen combinar con frecuencia para realizar filtros. Escribir un programa usando llamadas al sistema y funciones de la librería de C cuyo ejecutable llamaremos **calificacion** y que admita dos argumentos, el nombre de un fichero (por ejemplo examenmayo) y una cadena (por ejemplo manuel) de modo que ejecutar el comando:

```
$ ./calificacion examenmayo manuel
```

sea equivalente a ejecutar:

```
$ cat examenmayo | grep manuel | cut -d " " -f 2
```

El programa deberá crear 3 procesos hijos y esperar a su terminación. Cada uno de los hijos ejecutará el correspondiente comando: el hijo 1 ejecutará el comando cat, el hijo2 ejecutará el comando grep y el hijo 3 ejecutará el comando cut. Para la comunicación entre los procesos hijos se deberán utilizar pipes anónimos.

Nombre y Apellidos:_____

Ejercicio 2. (4 Puntos)
El siguiente fragmento muestra parte del código de un programa llamado **runp** en el que un proceso padre crea un proceso hijo que se encarga de ejecutar un comando con un único argumento que se pasa por línea de comandos al proceso padre.

```
#include ...

int main(int argc, char *argv[]) {
	pid_t childpid;
	if (argc != 3) {
		fprintf(stderr, "Uso: %s comando argumento\n", argv[0]);
		return 1;
	}
	childpid = fork();
	if (childpid == -1) {
		perror("fallo en fork");
		return 1;
	}
	if (childpid == 0) {
	// Ejectuar el comando con sus argumentos
	}
	...
}
```

Ej2.1 (2 puntos) Completar el programa de modo que:

a) El proceso padre termine su ejecución tras crear el proceso hijo sin esperar a su terminación.
b) El proceso hijo se ejecutará en "segundo plano" y finalizará cuando termine la ejecución del comando y no se verá afectado por la recepción de señales relacionadas con el terminal o por la desconexión del mismo.

Si el ejecutable de nuestro programa se llama **runp**, puede comprobarse el funcionamiento con:

```
$ ./runp sleep 10
```

Sugerencia: en la rama if del proceso hijo se puede utilizar el siguiente esquema de código:

```
	if (childpid == 0) {
	// 1- Hacer que el hijo se pueda ejecutar "en segundo plano"
	// 2- Ejectuar el comando con sus argumentos
	// 3 - Informar posible error
	}
```

Ej2.2 (2 puntos) Completar el programa de modo que el proceso hijo se ejecute en primer plano. En este caso, el programa padre deberá esperar a que el hijo termine o se suspenda y se deberá informar del resultado de su ejecución por la salida estándar. El proceso hijo en este caso si podrá recibir señales desde el terminal.