Examen de Ampliación de Sistemas Operativos 18 de mayo de 2022

Nombre y Apellidos:_____

Cuestión 1. (2 Puntos)

El siguiente fragmento de código analiza el contenido de un directorio que se ha pasado como argumento al programa mostrando información sobre los ficheros y directorios incluidos en el mismo.

```
int main(int argc, char *argv[]) {
DIR *dd;
struct stat *estado;
struct dirent *entrada;
...
dd = opendir(argv[1]);
while ((entrada= readdir (dd)) != NULL){
      ...
      estado = malloc(sizeof(struct stat));
      ...
      stat(...,estado);
}
```

1.a. La llamada a **`opendir`** puede fallar y el programa se puede mejorar haciendo una comprobación previa de la ruta que nos han pasado en **`argv[1]`** . Qué comprobarías y cómo antes de llamar a **`opendir`**?

En el cuerpo del bucle **`while`** que recorre las entradas del directorio se utiliza la llamada **`stat`** para conocer información de estado de las entradas que se incluyen en el directorio.

1.b. Si la llamada a **`stat(…,estado)`** tiene éxito, qué representa **`estado->st_nlink`**

Nombre y Apellidos:_____

1.c. Qué ocurre con **`stat(…,estado)`** si no se tiene permiso de lectura en un fichero del directorio?

Cuestión 2. (2 Puntos)

El siguiente fragmento de código muestra un programa en el que se crean dos procesos hijos.

```
int main(int argc, char * argv[]){

	int fds[2];
	pid_t pid1,pid2;

	...

	if (pid1 = fork()){
		if (pid2 = fork()){



		} else {



		}

	} else {



	}

	...

}
```

2.a. ¿Sería posible comunicar el proceso hijo con el pid que se almacena en la variable `pid1` (que denotaremos como ***"hijo1"***) con el proceso hijo con el pid que se almacena en la variable `pid2` (que denotaremos como ***"hijo2"***) utilizando una tubería (*pipe*) sin nombre?

Nombre y Apellidos:_____

En caso afirmativo completar el código para que ***hijo2*** lea de la tuberia en la que escribe **hijo1**

2.b. ¿Ofrecerían alguna ventaja las tuberías con nombre frente a las tuberías sin nombre en este ejemplo? Razonar la respuesta incluyendo modificaciones en el código si fuese oportuno.

Nombre y Apellidos:_____

Cuestión 3. (2 Puntos)
Utilizando la llamada sistema **`waitpid`** podemos sincronizar la ejecución de determinados procesos forzando esperas hasta la terminación o cambio de estado de otros procesos o grupos de procesos.

3.a. ¿Es posible utilizar **`waitpid`** para que un proceso hijo espere la finalización de un proceso padre? En caso afirmativo esbozar un posible esquema de código a utilizar.

3.b. Otra forma de sincronizar procesos es con señales como en en el siguiente ejemplo:

```
#include <stdio.h>
...
int terminar = 0;
void shandler(int signo){
	char msj[]="signal capturada\n";terminar =1;
	if(signo == SIGUSR1) write(1,msj,strlen(msj));
	return;
}
int main(void){
	pid_t pid;
	printf("Soy el proceso pid = %d\n", getpid());
	signal(SIGUSR1,SIG_IGN);
	if((pid = fork()) != 0){
		printf("Enviando SIGUSR1 ...\n");
		kill(pid, SIGUSR1);
	} else {
		printf("Soy el proceso %d, mi padre es %d\n",getpid(), getppid());
		signal(SIGUSR1,shandler);
		while(!terminar);
	}
	return 0;
}
```

Nombre y Apellidos:_____

Bajo que circunstancias, si las hay, la llamada a **`kill(pid, SIGUSR1)`** en el proceso padre provoca la terminación posterior del proceso hijo. Razonar la respuesta.

3.c. Modificar el código anterior modificando mascaras de señales para que el proceso hijo siempre termine. Sugerencia: utilizar las llamadas **`sigfillset, sigemptyset`** y **`sigprocmask`**

Nombre y Apellidos:_____

Ejercicio 1. (2 Puntos)

`getopt` es una función de la librería estándar de Unix que permite analizar parámetros de tipo opción introducidos por la línea de comandos. Para analizar opciones largas se utiliza la extensión de GNU `getopt_long`. Completar la plantilla del programa **ejercicio1.c** para que acepte las siguientes cuatro opciones largas:

`--breve`	fijara *`breve`* a 1
`--verbose`	fijara *`breve`* a -1
`--entorno` ***`VAR`***	mostrara el contenido de la variable de entorno **`$VAR`** con el formato:

- **“**`La variable de entorno` ***`VAR`*** `tiene el valor` ***`VALOR`*** **”** Si `breve es -1`
- “***`VALOR`***” Si `breve es 1`
- “***`VAR=VALOR`***” Si `breve es 0`

`--fecha` mostrara por la salida la fecha actual en un formato legible del tipo:

- “`Jueves 18/5/2022, 17:00:59`”. Si `breve es -1`
- “`18/5/2022`”. Si `breve es 1`
- “*`segundos`***”** *(segundos desde 1 de Enero de 1970)* Si `breve es 0`

Además, se admitirán 2 opciones cortas:

`-e` ***`VAR`***	equivalente a **`--entorno`** ***`VAR`***
`-f`	equivalente a **`–fecha`**

Si se incluye alguna opción distinta a las anteriores el programa mostrará un mensaje de error y abortará su ejecución devolviendo el código de terminación 1.

Después de analizar estas opciones el programa deberá mostrar por la salida estándar el resto de argumentos que se hayan pasado al programa que no sean de tipo opción.

Ejercicio 2. (2 Puntos)

Completar la plantilla del programa **ejercicio2.c** para que todos los procesos que se lancen a ejecutar desde este shell simplificado se hagan por defecto en *background*. El shell incluirá dos ordenes especiales sin argumentos:

- **`asoexit`** que forzará al shell la terminación de todos los procesos que se hubieran lanzado previamente (si los hubiese) y esperará a su terminación antes de salir del shell.

- **`asowait`** que forzará al shell a ceder el terminal de control al último proceso que se hubiese lanzado y estuviera aún en ejecución (si lo hubiese) y esperará a su terminación o suspensión antes de recuperar de nuevo el terminal de control para solicitarse nuevas ordenes.