SISTEMAS OPERATIVOS - ANALISTA UNIVERSITARIO EN SISTEMAS

PRACTICA 1: Programación C y llamadas al sistema

Ejercicio 1:

a) Escribir una función que ordene un array de n números enteros (n es un unsigned ingresado por el usuario) mediante el algoritmo quicksort.

b) Crear un programa que utilice la función qsort definida en <stdlib.h>

c) Dotar a la función del apartado a) con la siguiente característica: "La función podrá recibir, mediante puntero a función, qué tipo de orden se llevará a cabo, por ejemplo: ascendente, descendente, valor absoluto ascendente, valor absoluto descendente, etc.

Ejercicio 2:

a) Escribir una función que busque un elemento en un array de n números enteros, sin signo, ordenados, donde n es un unsigned ingresado por el usuario, mediante el algoritmo binary search

b) Crear un programa que utilice la función bsearch definida en <stdlib.h>

Ejercicio 3 :

Analizar los siguientes fragmentos de código y explicar su comportamiento.

a)

```
int main() {
  int p = fork();
  if (p != 0) for(;;);
  if (p == 0) execl("/bin/ls", "ls", "-l", NULL);
  printf("Mi pid es %d", getpid());
  return 0;
}
```

b)

```
int main() {
 int i, j, k;
 for (i=0; i<3; i++) {
    j = getpid();
    k = fork();
    printf("%d - %d\n",j, k);
 }
 return 0;
}
```

c)

```
int main() {
  int p = fork();
  if (p == 0) for(;;);
  wait();
  printf("Mi pid es %d", getpid());
  return 0;
}
```

Ejercicio 4 :

Crear un programa que genere dos procesos hijos.

El padre debe esperar por la terminación de ambos procesos hijos antes de imprimir por pantalla **"Proceso padre esperando..."**, y posteriormente imprimirá **"Hijo PID_1er_hijo finalizó"** e **"Hijo PID_2do_hijo finalizó"**.

El primer proceso hijo debe realizar un exec imprimiendo en pantalla el calendario del mes y año (formato *mm aaaa*). Parámetros establecidos por el padre y recibidos por línea de comandos (el usuario lo introduce).

El segundo hijo deberá recibir una señal SIGUSR1 y mediante un handler emitirá por pantalla el mensaje **"Recibida la señal SIGUSR1"** y luego finalizará con **_exit()**.

Finalmente el proceso padre mostrará por pantalla su propio PID y finalizará.

Ejemplo de ejecución:

```
$ ./ejercicio3 10 2009
Proceso padre esperando por hijo 13480
    octubre 2009
 lu ma mi ju vi sá do
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
Hijo 13480 terminado
Proceso padre esperando por hijo 13770
(enviar señal desde otra terminal mediante comando kill)
Recibida la señal SIGUSR1
Hijo 13770 terminado
Proceso padre 13400 terminando...
```

Ejercicio 5 :

Crear un programa denominado ***my_cat*** que imite al comando UNIX **`cat`**. El mismo concatena el contenido de uno o más archivos en un único flujo de caractere (por defecto) dirigido a *stdout*.
Su uso será:

`my_cat <archivo1> <archivo2> ... [-o <archivo_salida>]`

- Al pasar la opción **`-o`** en lugar de escribir a la salida estándar (pantalla) escribirá a un archivo llamado **`<archivo_salida>`**.

Nota: Usar las llamadas al sistema: open, write, read, close y cualquier otra que sea necesaria.

Ejercicio 6 :

Crear un programa llamado **`my_ls`**.
For uso será:

`my_ls <opción> <directorio>`

donde **`<directorio>`** es el nombre de un directorio y según la opción que se le pase hará lo siguiente. **`<opción>`:**

`-e` : liste todos los archivos contenidos en **`<directorio>`**
`-d` : liste sólo los directorios contenidos en **`<directorio>`**
`-i` : liste el número de i-nodo y el nombre de los archivos contenidos en **`<directorio>`**.

Notas: Usar las llamadas al sistema: **`stat`** y cualquier otra que consideren necesaria.
Además usar las funciones **`opendir`**, **`readdir`**, **`closedir`**.

Ejercicio 7 :

Crear un comando denominado **my_chmod** que imite el comportamiento del comando UNIX **chmod**. El mismo sirve para cambiar los permisos de acceso a archivos y directorios.
Usar la llamada al sistema **chmod** y cualquier otra que sea de utilidad.

Ejercicio 8 :

El comando **pkill** en UNIX permite el envío de señales a los procesos discriminándolos por nombre (`man pkill`). Crear un programa denominado **my_pkill** que imite el comportamiento de dicho comando, pero restringido al envío de señales.

Es decir: **my_pkill [-signal] pattern**

Si no se especifica señal alguna, enviar SIGTERM.
pattern especifica una expresión regular para comparar con los nombres de los procesos.

my_pkill deberá presentar un mensaje de error en el caso de ser ejecutado por un usuario que, por ejemplo, trate de terminar un proceso sobre el que no tiene injerencia (uno lanzado por otro usuario).

NOTA FINAL: Para todos los ejercicios NO se permite el uso de la función system // man system(3)
#include <stdlib.h>
*int system(const char *command);*