



# Tarea 2

Sistemas Operativos

Jorge Angel Juárez Vázquez  
2213026247

Profesor: Jose Netz Romero Duran  
12 August 2024

# Tarea 2 - Procesos



Negocio: WildFork

## Menú

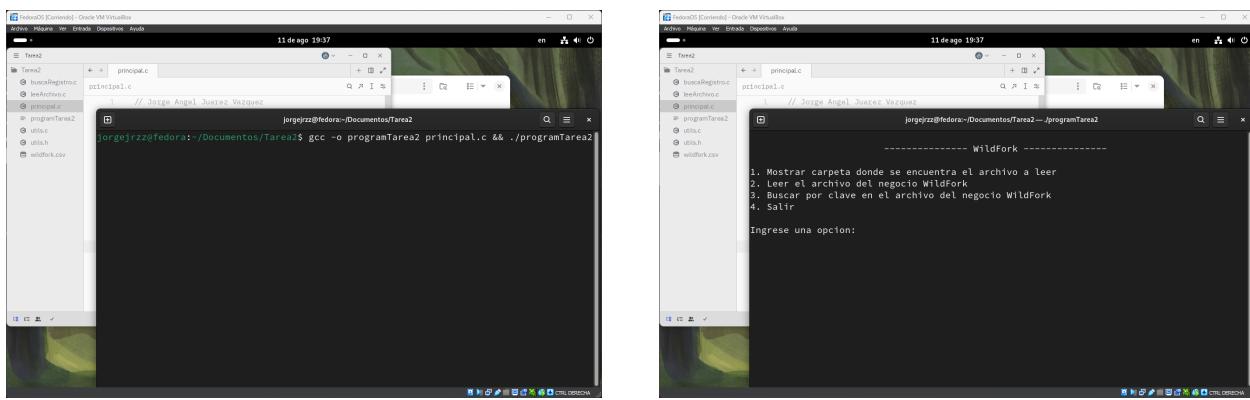
Para poder imprimir el menú con el que vamos a estar interactuando para esta tarea, se desarrolló una función de tipo `int` que hace las siguientes cuatro tareas:

1. Limpiar la pantalla
2. Imprimir el menú
3. recibir la opción ingresada por el usuario
4. devolver la opción elegida

El código que genera la impresión del menú, es el siguiente:

```
1 int print_menu() {
2     int option;
3     system("clear");
4     printf("\n\t\t\t----- WildFork -----\\n\\n");
5     printf("1. Mostrar carpeta donde se encuentra el archivo a leer\\n");
6     printf("2. Leer el archivo del negocio WildFork\\n");
7     printf("3. Buscar por clave en el archivo del negocio WildFork\\n");
8     printf("4. Salir\\n\\n");
9     printf("Ingrese una opcion: ");
10    scanf("%d", &option);
11    return option;
12 }
```

Y por terminal, la impresión se ve de la siguiente manera:



## Opción 1 - Mostrar la carpeta en donde se encuentra el archivo a leer

Para la primera opción de menú anterior, tenemos que hacer que nuestro programa ejecute por terminal el comando `ls -l`, esto se puede hacer con la instrucción `exec1` pero, una característica de este comando es que termina con el proceso actual para poder ejecutar la instrucción que nosotros le estemos pasando, pero podemos usar dos procesos, uno donde se tenga el proceso de nuestro programa, y otro donde se ejecute el comando con `exec1` proceso que va a terminar cuando se ejecute esta instrucción, entonces el código para poder hacer esto, es el siguiente:

```

1 pid1 = fork();
2 if (pid1 < 0) {
3     // Error al crear el proceso
4     perror("Fork failed");
5     exit(1);
6 } else if (pid1 == 0) {
7     // Código del proceso hijo.
8     execl("/bin/ls", "ls", "-l", NULL);
9 }
10 wait(NULL);

```

Este código proporciona la siguiente salida por terminal:

```

jorgejrzz@fedora-:~/Documentos/Tarea2--/programTarea2
----- WildFork -----
1. Mostrar carpeta donde se encuentra el archivo a leer
2. Leer el archivo del negocio WildFork
3. Buscar por clave en el archivo del negocio WildFork
4. Salir

Ingrese una opcion: 1

jorgejrzz@fedora-:~/Documentos/Tarea2--/programTarea2
----- WildFork -----
total 48
-rw-r--r--. 1 jorgejrzz jorgejrzz 2331 ago 11 19:14 buscaRegistro.c
-rw-r--r--. 1 jorgejrzz jorgejrzz 1094 ago 11 19:15 leeArchivo.c
-rw-r--r--. 1 jorgejrzz jorgejrzz 4537 ago 11 19:21 principal.c
-rwxr-xr-x. 1 jorgejrzz jorgejrzz 17160 ago 11 19:37 programTarea2
-rw-r--r--. 1 jorgejrzz jorgejrzz 2088 ago 11 19:15 utils.c
-rw-r--r--. 1 jorgejrzz jorgejrzz 308 ago 11 19:18 utils.h
-rw-r--r--. 1 jorgejrzz jorgejrzz 579 ago 11 19:16 wildfork.csv

Presione Enter para continuar...

```

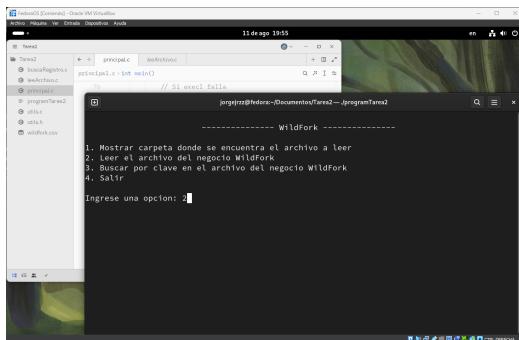
## Opción 2 - Leer el archivo del negocio WildFork

Para esta opción, se tenía que desarrollar en otro archivo la lógica para poder leer el archivo `csv` en dónde se encuentran los registros de diferentes productos del negocio, entonces, lo primero que se hace en este caso para poder ejecutar otro programa, es compilarlo, todo esto con la misma instrucción de `execl`, pero como en este caso se necesitan dos de estas instrucciones (una para compilarlo, y otra para ejecutar el

programa) no podemos solo utilizar dos procesos, si no que necesitamos crear 3 procesos. Con el siguiente código podemos lograr esto:

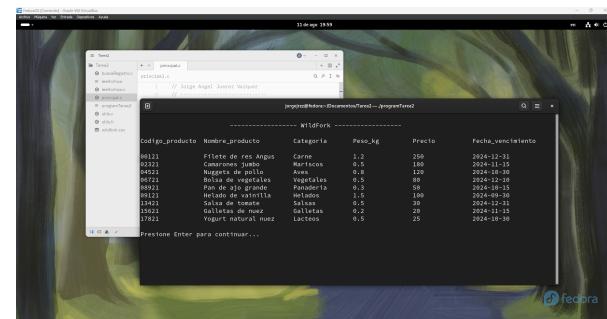
```
1 // Crear el primer proceso para la compilación
2 pid1 = fork();
3 if (pid1 < 0) {
4     // Error al crear el proceso
5     perror("Fork failed");
6     exit(1);
7 } else if (pid1 == 0) {
8     // Código del proceso hijo: compilar los archivos con gcc
9     execl("./usr/bin/gcc", "gcc", "utils.c", "leeArchivo.c", "-o", "leeArchivo", "-I", ".h", NULL);
10    // Si execl falla
11    perror("execl failed");
12    exit(1);
13 }
14
15 // Esperar a que la compilación termine
16 wait(NULL);
17
18 // Crear el segundo proceso para ejecutar el programa compilado
19 pid2 = fork();
20 if (pid2 < 0) {
21     // Error al crear el proceso
22     perror("Fork failed");
23     exit(1);
24 } else if (pid2 == 0) {
25     // Código del proceso hijo: ejecutar el programa compilado
26     execl("./leeArchivo", "./leeArchivo", NULL);
27     // Si execl falla
28     perror("execl failed");
29     exit(1);
30 }
31 // Esperar a que el programa compilado termine
32 wait(NULL);
```

Este código genera la siguiente salida por la línea de comandos:



A terminal window titled 'jorgeff@fedora-DOCUMENTACION:~/Tarea2--JprogramTarea2'. It shows the command 'java WildFork' being run. The output displays a menu with options 1 through 4, followed by a prompt 'Ingrese una opcion:'. The menu items are:

1. Mostrar carpeta donde se encuentra el archivo a leer
2. Leer el archivo del negocio WildFork
3. Buscar por clave en el archivo del negocio WildFork
4. Salir

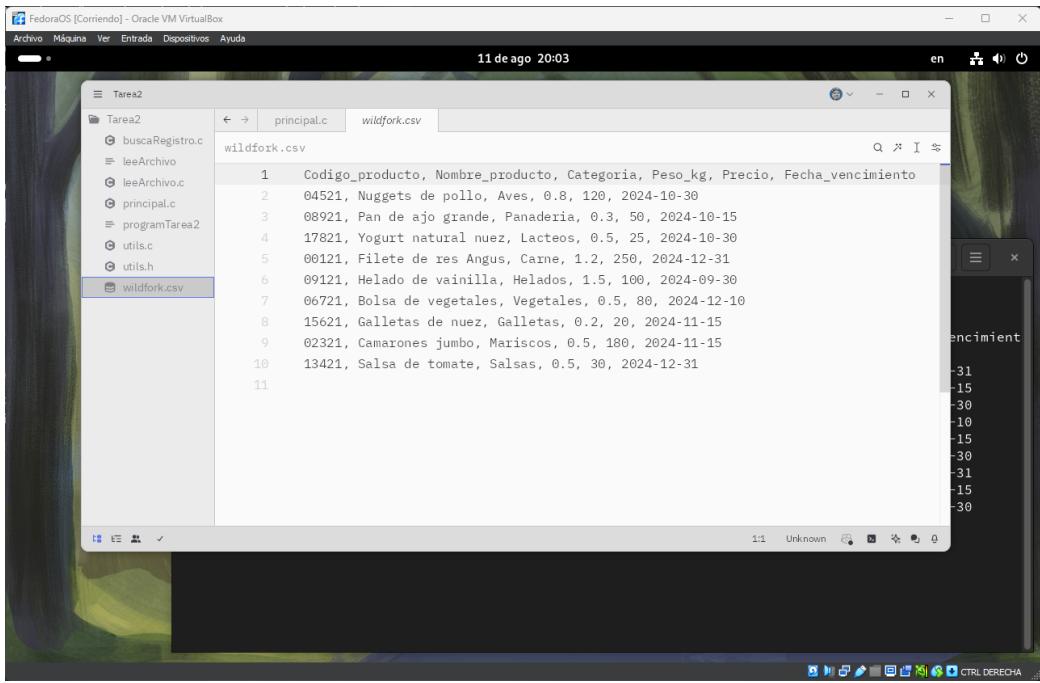


A screenshot of a Java application window titled 'jorgeff@fedora-DOCUMENTACION:~/Tarea2--JprogramTarea2'. The window displays a table of products with columns: Código\_producto, Nombre\_producto, Categoría, Peso\_kg, Precio, and Fecha\_vencimiento. The data is as follows:

Código_producto	Nombre_producto	Categoría	Peso_kg	Precio	Fecha_vencimiento
00121	Filette de res Angus	Carne	1.2	250	2024-12-31
00221	Comarcón Jumbo	Mariicon	0.5	180	2024-11-15
00321	Huevo de gallina	Carne	0.5	120	2024-12-10
00421	Bolsita de vegetales	Vegetales	0.5	80	2024-12-10
00521	Yuca	Almidones	0.5	50	2024-11-15
00621	Melado de vainilla	Melados	1.5	100	2024-09-30
12321	Sopa de tomate	Saladas	0.5	30	2024-11-15
15621	Galletas de nuez	Snacketas	0.2	20	2024-11-15
17821	Yogurt natural nuez	Lacteos	0.5	25	2024-10-15

## Archivo `leeArchivo.c`

Este archivo es el encargado de leer el archivo en donde se encuentran los registros del negocio, el cual tiene el siguiente contenido:



	Código_producto	Nombre_producto	Categoría	Peso_kg	Precio	Fecha_vencimiento
1	04521	Nuggets de pollo	Aves	0.8	120	2024-10-30
2	08921	Pan de ajo grande	Panadería	0.3	50	2024-10-15
3	17821	Yogurt natural nuez	Lacteos	0.5	25	2024-10-30
4	00121	Filete de res Angus	Carne	1.2	250	2024-12-31
5	09121	Helado de vainilla	Helados	1.5	100	2024-09-30
6	06721	Bolsa de vegetales	Vegetales	0.5	80	2024-12-10
7	15621	Galletas de nuez	Galletas	0.2	20	2024-11-15
8	02321	Camarones jumbo	Mariscos	0.5	180	2024-11-15
9	13421	Salsa de tomate	Salsas	0.5	30	2024-12-31
10						
11						

Para poder leerlo, también se optó por el uso de procesos, esto porque en el archivo están desordenados, y se requiere que al usuario se le muestren los registros de forma ordenada, entonces un proceso se va a utilizar para leer el archivo y almacenar el contenido en un arreglo, y otro proceso para ordenar el arreglo e imprimirlo por consola.

Por la naturaleza de los procesos y los arreglos en C esto puede ser una tarea difícil en cuestión de compartir memoria en procesos, y es justamente lo que se hizo con el siguiente archivo `leeArchivo.c`:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/ipc.h>
6 #include <sys/shm.h>
7 #include <sys/wait.h>
8
9 #include "utils.h"
10
11 #define MAX_LINE_LENGTH 1024
12 #define NUM_PRODUCTOS 10
13
14 int main() {
15     // Crear una clave para la memoria compartida
16     key_t key = ftok("shmfile", 65);
17
18     // Calcular el tamaño de la memoria compartida
19     int shmid = shmget(key, NUM_PRODUCTOS * MAX_LINE_LENGTH * sizeof(char), 0666|IPC_CREAT);
20
21     // Adjuntar la memoria compartida al espacio de direcciones del proceso
22     char (*arr)[MAX_LINE_LENGTH] = shmat(shmid, NULL, 0);
23
24     // Crear un proceso hijo
25     pid_t pid = fork();
26
27     if (pid < 0) {
28         // Error al crear el proceso
29         perror("Fork failed");
30         exit(1);
31     } else if (pid == 0) {
32         // Código del proceso hijo: Crear el arreglo leyendo el archivo
33         csv_to_array("wildfork.csv", arr);
34         // Desapegar la memoria compartida
35         shmdt(arr);
36
37     } else {
38         // Código del proceso padre: Esperar a que el hijo termine, ordenar el arreglo y finalmente mostrarlo en pantalla
39         wait(NULL);
40         int i = 0;
41         char *field;
42
43         order_by_codigo(arr, NUM_PRODUCTOS);
44
45         // Imprimir el arreglo
46         printf("\n\t\t\t----- WildFork -----\\n\\n");
47         for (int i = 0; i < NUM_PRODUCTOS; i++) {
48             // Separar los campos y formatear la salida con tabuladores
49             field = strtok(arr[i], ",");
50             while (field != NULL) {
51                 // Ajusta el ancho de los campos según sea necesario
52                 printf("%-15s\t", field);
53                 field = strtok(NULL, ",");
54             }
55             printf("\\n");
56         }
57
58         // Desapegar la memoria compartida
59         shmdt(arr);
60
61         // Eliminar el segmento de memoria compartida
62         shmctl(shmid, IPC_RMID, NULL);
63     }
64
65     return 0;
66 }

```

Como se puede observar las dos funciones principales son `order_by_codigo();` y `csv_to_array();`

Estas funciones, y sobre todo el procedimiento `order_by_codigo();` se planean usar mas de una ves, entonces pensando en la modularidad del proyecto, se realizo una biblioteca propia para el proyecto llamada `utils.h` :

```
1 // utils.h
2 #ifndef UTILS_H
3 #define UTILS_H
4
5 #define MAX_LINE_LENGTH 1024
6
7 void order_by_codigo(char arr[][MAX_LINE_LENGTH], int num_productos);
8 int csv_to_array(char *file_path, char arr[][MAX_LINE_LENGTH]);
9 int search_by_clave(char arr[][MAX_LINE_LENGTH], int num_productos, char *clave);
10
11 #endif
```

Estas funciones y procedimientos tienen las siguientes implementaciones en el archivo `utils.c`:

```

● ● ●

1 #include "utils.h"
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6
7 #define MAX_LINE_LENGTH 1024
8 #define NUM_PRODUCTOS 10
9
10 void order_by_codigo(char arr[][MAX_LINE_LENGTH], int num_productos) {
11     char temp[MAX_LINE_LENGTH];
12
13     for (int i = 1; i < num_productos - 1; i++) {
14         for (int j = i + 1; j < num_productos; j++) {
15             // Comparar los códigos de producto (los primeros campos de cada cadena)
16             if (strcmp(arr[i], arr[j]) > 0) {
17                 // Intercambiar las filas si están en el orden incorrecto
18                 strcpy(temp, arr[i]);
19                 strcpy(arr[i], arr[j]);
20                 strcpy(arr[j], temp);
21             }
22         }
23     }
24 }
25
26 int csv_to_array(char *file_path, char arr[][MAX_LINE_LENGTH]) {
27     FILE *fp;
28     char buffer[MAX_LINE_LENGTH];
29     char *field;
30     int i = 0;
31
32     // Abrir el archivo en modo lectura
33     fp = fopen(file_path, "r");
34     if (fp == NULL) {
35         perror("Error al abrir el archivo\n");
36         return 1;
37     }
38
39     // Leer el archivo línea por línea
40     while (fgets(buffer, MAX_LINE_LENGTH, fp) != NULL) {
41         // Eliminar el salto de línea al final de la línea
42         buffer[strcspn(buffer, "\n")] = 0;
43         // agregar los datos al arreglo
44         strcpy(arr[i], buffer);
45         i++;
46     }
47     // Cerrar el archivo
48     fclose(fp);
49     return 0;
50 }
51
52 int search_by_clave(char arr[][MAX_LINE_LENGTH], int num_productos, char *clave) {
53     int izquierda = 0;
54     int derecha = num_productos - 1;
55
56     // Crear una copia del arreglo para no modificar el original
57     char arr_copy[num_productos][MAX_LINE_LENGTH];
58     memcpy(arr_copy, arr, num_productos * MAX_LINE_LENGTH);
59
60     while (izquierda <= derecha) {
61         int medio = izquierda + (derecha - izquierda) / 2;
62
63         int comparacion = strcmp(clave, strtok(arr_copy[medio], ","));
64
65         if (comparacion == 0) {
66             return medio;
67         } else if (comparacion < 0) {
68             derecha = medio - 1;
69         } else {
70             izquierda = medio + 1;
71         }
72     }
73
74     return -1; // Clave no encontrada
75 }

```

## Opción 3 - Buscar por clave en el archivo del negocio WildFork

En esta opción se ocupa la misma lógica que en el punto anterior, y justamente se ocupan las funciones y procedimientos establecido en nuestra biblioteca `utils.h`

Ahora el programa que se tiene que ejecutar en esta opción es del archivo `buscaRegistro.c` el cual tiene el siguiente contenido:

```

● ● ●
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/ipc.h>
6 #include <sys/shm.h>
7 #include <sys/wait.h>
8
9 #include "utils.h"
10
11 #define MAX_LINE_LENGTH 1024
12 #define NUM_PRODUCTOS 10
13
14
15 int main() {
16     // Crear una clave para la memoria compartida
17     key_t key = ftok("shmfile", 65);
18
19     // Calcular el tamaño de la memoria compartida
20     int shmid = shmget(key, NUM_PRODUCTOS * MAX_LINE_LENGTH * sizeof(char), 0666|IPC_CREAT);
21
22     // Adjuntar la memoria compartida al espacio de direcciones del proceso
23     char (*arr)[MAX_LINE_LENGTH] = shmat(shmid, NULL, 0);
24
25     // Crear un proceso hijo
26     pid_t pid = fork();
27
28     if (pid < 0) {
29         // Error al crear el proceso
30         perror("Fork failed");
31         exit(1);
32
33     } else if (pid == 0) {
34         // Código del proceso hijo: Crear el arreglo leyendo el archivo
35         csv_to_array("wildfork.csv", arr);
36         // Desapegar la memoria compartida
37         shmdt(arr);
38
39     } else {
40         // Código del proceso padre
41         wait(NULL);
42         int i = 0;
43         char *field;
44         char input[5];
45
46         // Ordenar el arreglo por el campo "codigo"
47         order_by_codigo(arr, NUM_PRODUCTOS);
48
49         // El usuario ingresa una clave
50         printf("Ingrese la clave del producto (5 digitos): ");
51         scanf("%s", input);
52         int index = search_by_clave(arr, NUM_PRODUCTOS, input);
53
54         if (index != -1) {
55             printf("Producto encontrado:\n\n");
56             // Código_producto, Nombre_producto, Categoría, Peso_kg, Precio, Fecha_vencimiento
57             printf("%-15s\t%-15s\t%-15s\t%-15s\n", "Codigo_producto", "Nombre_producto", "Categoria", "Peso_kg", "Precio", "Fecha_vencimiento");
58             // Separar los campos y formatear la salida con tabuladores
59             field = strtok(arr[index], ",");
60             while (field != NULL) {
61                 // Ajusta el ancho de los campos según sea necesario
62                 printf("%-15s\t", field);
63                 field = strtok(NULL, ",");
64             }
65             printf("\n");
66         } else {
67             printf("Producto no encontrado\n");
68         }
69
70         // Desapegar la memoria compartida
71         shmdt(arr);
72
73         // Eliminar el segmento de memoria compartida
74         shmctl(shmid, IPC_RMID, NULL);
75     }
76
77     return 0;
78 }

```

Este código al compilarlo y ejecutarlo, genera la siguiente interacción por pantalla:

```

jorgejrz@fedora:~/Documentos/Tarea2--./programTarea2
ingrese la clave del producto (5 digitos): 17821
Producto encontrado:
Codigo_producto Nombre_producto Categoría Peso_kg Precio Fecha_vencimiento
17821 Yogurt natural nuez Lacteos 0.5 25 2024-10-30

```

```

jorgejrz@fedora:~/Documentos/Tarea2--./programTarea2
ingrese la clave del producto (5 digitos): 17821
Producto encontrado:
Codigo_producto Nombre_producto Categoría Peso_kg Precio Fecha_vencimiento
17821 Yogurt natural nuez Lacteos 0.5 25 2024-10-30

```

Como se esta haciendo la búsqueda sobre un arreglo en donde los registros están ordenado ascendentemente, la búsqueda es con el algoritmo de **búsqueda binaria**, por lo que la hace bastante eficiente, esto es lo que pasa cuando no se encuentra el registro:

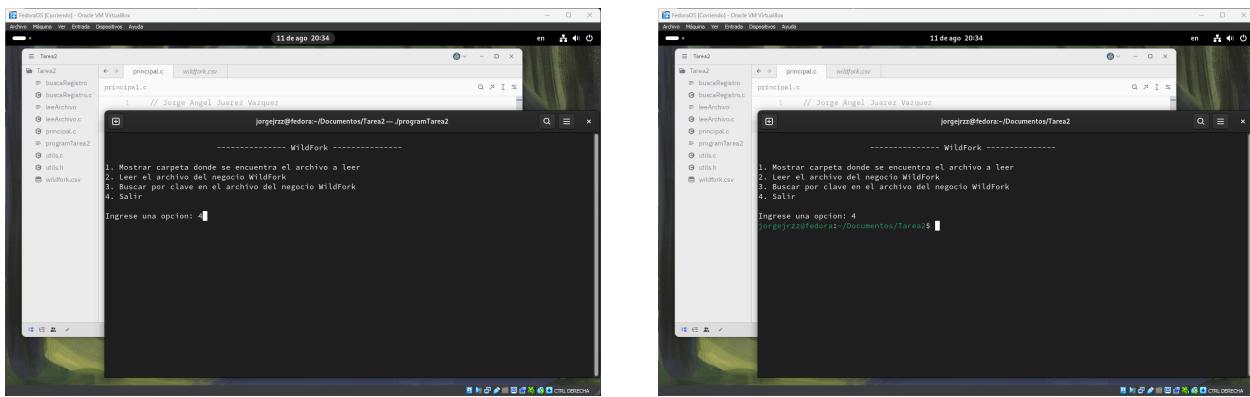
```

jorgejrz@fedora:~/Documentos/Tarea2--./programTarea2
ingrese la clave del producto (5 digitos): 1
Producto no encontrado

```

## Opcion 4 - Salir

Finalmente esta opción hace que termine el programa principal, de forma que no quede ningún proceso o uso de memoria no liberado, así es como se muestra la terminal cuando el usuario elige esta opción:



```
jorgejrz@fedora-:~/Documentos/Tarea2$ ./principal.c
principal.c
principal.c // Jorge Angel Juarez Vazquez
jorgejrz@fedora-:~/Documentos/Tarea2$ jorgejrz@fedora-:~/Documentos/Tarea2$ WildFork
WildFork
1. Mostrar carpeta donde se encuentra el archivo a leer
2. Leer el archivo del negocio WildFork
3. Buscar por clave en el archivo del negocio WildFork
4. Salir
Ingrese una opcion: 4
jorgejrz@fedora-:~/Documentos/Tarea2$
```

El siguiente es el contenido completo del archivo `principal.c`:

```

 000
 1 #include <csdio.h>
 2 #include <csdlib.h>
 3 #include <sys/types.h>
 4 #include <sys/stat.h>
 5 #include <sys/ipc.h>
 6 #include <sys/shm.h>
 7 #include <sys/wait.h>
 8
 9 int print_menu() {
10     int option;
11     system("clear");
12     printf("\n\n\n\n\n----- WildFork -----\\n\\n");
13     printf("1. Mostrar carpeta donde se encuentra el archivo a leer\\n");
14     printf("2. Leer el archivo del negocio WildFork\\n");
15     printf("3. Buscar por clave en el archivo del negocio WildFork\\n");
16     printf("4. Salir\\n");
17     printf("Ingres una opcion: ");
18     scanf("%d", &option);
19     return option;
20 }
21
22 int main() {
23     int option;
24     pid_t pid1, pid2;
25     option = print_menu();
26
27     while (option != 4) {
28         switch (option) {
29             case 1:
30                 system("clear");
31                 pid1 = fork();
32                 if (pid1 < 0) {
33                     // Error al crear el proceso
34                     perror("fork failed");
35                     exit(1);
36                 } else if (pid1 == 0) {
37                     // Código del proceso hijo.
38                     exec("./bin/ls", "ls", "-l", NULL);
39                 }
40                 wait(NULL);
41                 break;
42             case 2:
43                 // gco utils.c leeArchivo.c -o leeArchivo -I . h 06 ./leeArchivo
44                 system("clear");
45                 // Crear el primer proceso para la compilación
46                 pid1 = fork();
47                 if (pid1 < 0) {
48                     // Error al crear el proceso
49                     perror("fork failed");
50                     exit(1);
51                 } else if (pid1 == 0) {
52                     // Código del proceso hijo: compilar los archivos con gcc
53                     exec("./usr/bin/gcc", "gcc", "utils.c", "-o", "leeArchivo", "-I", ".h", NULL);
54                     // Si ejecuta
55                     perror("exec failed");
56                     exit(1);
57                 }
58                 // Esperar a que la compilación termine
59                 wait(NULL);
60
61                 // Crear el segundo proceso para ejecutar el programa compilado
62                 pid2 = fork();
63                 if (pid2 < 0) {
64                     // Error al crear el proceso
65                     perror("fork failed");
66                     exit(1);
67                 } else if (pid2 == 0) {
68                     // Código del proceso hijo: ejecutar el programa compilado
69                     exec("./leeArchivo", "./leeArchivo", NULL);
70                     // Si ejecuta
71                     perror("exec failed");
72                     exit(1);
73                 }
74                 // Esperar a que el programa compilado termine
75                 wait(NULL);
76                 break;
77             case 3:
78                 system("clear");
79                 // Crear el primer proceso para la compilación
80                 pid1 = fork();
81                 if (pid1 < 0) {
82                     // Error al crear el proceso
83                     perror("fork failed");
84                     exit(1);
85                 } else if (pid1 == 0) {
86                     // Código del proceso hijo: compilar los archivos con gcc
87                     exec("./usr/bin/gcc", "gcc", "buscaRegistro.c", "-o", "buscaRegistro", "-I", ".h", NULL);
88                     // Si ejecuta
89                     perror("exec failed");
90                     exit(1);
91                 }
92                 // Esperar a que la compilación termine
93                 wait(NULL);
94
95                 // Crear el segundo proceso para ejecutar el programa compilado
96                 pid2 = fork();
97                 if (pid2 < 0) {
98                     // Error al crear el proceso
99                     perror("fork failed");
100                    exit(1);
101                } else if (pid2 == 0) {
102                    // Código del proceso hijo: ejecutar el programa compilado
103                    exec("./buscaRegistro", "./buscaRegistro", NULL);
104                    // Si ejecuta
105                    perror("exec failed");
106                    exit(1);
107                }
108                // Esperar a que el programa compilado termine
109                wait(NULL);
110                break;
111            default:
112                printf("Opcion no valida\\n");
113                break;
114        }
115        printf("\\nPresione Enter para continuar ...");
116        getchar();
117        getchar();
118        option = print_menu();
119    }
120
121    return 0;
122 }

```