



UNS



DCIC

SISTEMAS OPERATIVOS

Proyecto 2024



Comisión 34

-Brenda Belen Martinez Ocampo

Índice

1. Experimentación de Procesos y Threads con los Sistemas Operativos	3
1.1. Procesos, threads y Comunicación	3
1. Pumper Nic.	3
PIPE	3
Cola de mensajes	5
2. Mini Shell.	10
1.2. Sincronización	22
1. Taller de Motos.	22
2. Santa Claus.	24
2. Problemas	27
2.1. Lectura	27
2.2. Problemas Conceptuales	27
1. Paginación y Segmentación en Memoria	27
2. Tabla de páginas	30

1. Experimentación de Procesos y Threads con los Sistemas Operativos

1.1. Procesos, threads y Comunicación

Para la ejecución de cada uno de los archivos se creo un script ejecutar.sh debe ingresar

cd [ruta del directorio]

y posicionado en dicho directorio en caso de no poseer los permisos de ejecucion ingrese

chmod +x ejecutar.sh

luego para ejecutar el script ingrese

./ejecutar.sh

1. Pumper Nic.

PIPE

a) Archivo con la resolución adjunta.

Para implementar el problema con pipes se hizo de la siguiente manera:

Pipes: Se definen 9 pipes que permiten la comunicación entre procesos:

- **pipe_clienteCOMUN y pipe_clienteVIP:** Para enviar los pedidos de clientes comunes y VIP. Cliente escribe y recibirPedido lee.
- **pipe_cola:** Indica la llegada de un nuevo pedido.El cliente escribe y recibirPedido lee.
- **pipe_orden_hamburguesa, pipe_orden_vegano, y pipe_orden_papas:** Para enviar órdenes de preparación a los empleados específicos. RecibirPedido escribe y el empleado a cargo lee.
- **pipe_entrega_hamburguesa, pipe_entrega_vegano, y pipe_entrega_papas:** Para recibir confirmaciones de que un pedido está listo.El cliente lee y el empleado a cargo lee.

Procesos: se definen 5 procesos:

EmpleadoHamburguesa, EmpleadoVegano y EmpleadoPapa: Estas funciones simulan a los empleados responsables de preparar cada tipo de pedido. Cada función cierra los pipes que no usa y escucha en su pipe de orden correspondiente. Cuando recibe un pedido, muestra un mensaje indicando que está preparando el tipo de pedido

correspondiente y, al finalizar, envía un mensaje de "pedido listo" a través de su pipe de entrega.

recibirPedido: Es el proceso encargado de recibir los pedidos de clientes y asignarlos a los empleados correspondientes. Cierra los pipes no necesarios y configura pipe_clienteVIP en modo no bloqueante para dar prioridad a los clientes VIP. Lee los pedidos desde pipe_cola para saber cuándo hay un nuevo cliente. Prioriza a los clientes VIP. Si no hay pedidos VIP, atiende a los pedidos comunes. Dependiendo del tipo de pedido (hamburguesa, vegano o papas), envía el pedido al pipe de orden correspondiente.

cliente Simula la actividad de cada cliente, quienes pueden ser comunes o VIP. Cada cliente decide aleatoriamente si entra a la cola o se va (simulado con $\text{rand}() \% 10 + 1$). Si el valor es 1, el cliente decide no hacer el pedido y se retira.

Si el cliente decide hacer el pedido: Escribe su pedido en el pipe correspondiente (pipe_clienteCOMUN o pipe_clienteVIP) y notifica la llegada mediante pipe_cola. Espera la confirmación de su pedido en el pipe de entrega correspondiente. Una vez recibido su pedido, muestra un mensaje indicando que se va después de recibirlo.

main Inicializa todos los pipes y crea un proceso hijo para cada empleado y el proceso de recepción de pedidos. Crea 10 procesos de clientes (según CANT_CLIENTES). Al finalizar, espera que todos los clientes y procesos de empleados terminen antes de cerrar el programa.

Para la comunicación de esta solución se utilizaron pipes para transmitir datos entre procesos, permitiendo que los clientes y el proceso receptor de pedidos se comuniquen de manera eficiente. Cada tipo de pedido tiene su propio pipe para recibir órdenes y otro para enviar confirmaciones, lo cual simplifica la gestión de la información y evita problemas de sincronización en procesos compartidos.

La solución da prioridad a los clientes VIP mediante un pipe no bloqueante (pipe_clienteVIP), verificando primero si hay un pedido VIP antes de procesar a los clientes comunes.

También la solución proporciona un NO acoplamiento de responsabilidades, teniendo cada empleado su propio proceso. Esto reduce el tiempo de espera para cada cliente y aumenta la eficiencia en el procesamiento de pedidos.

La implementación usa una espera activa en el proceso receptor de pedidos y en cada empleado, leyendo continuamente de los pipes de órdenes.

La solución obtenida es modular y con comunicación directa entre los distintos procesos por medio de pipes.

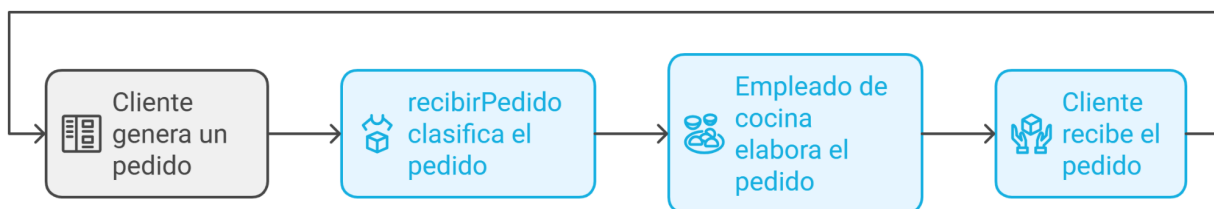
Cola de mensajes

b) Archivo con la resolución adjunta.

Este código simula un sistema de restaurante Pumper Nic, en el que los clientes realizan pedidos de comida, y los empleados preparan y entregan estos pedidos. Cada cliente y empleado opera en un proceso independiente, y la comunicación entre ellos se realiza mediante colas de mensajes.

El flujo de esta solución está dado por

- 1- Cliente genera un pedido y lo envía (cliente → recibirPedido).
- 2- recibirPedido clasifica el pedido y lo redirige al empleado de cocina correspondiente (recibirPedido → empleado).
- 3- El empleado procesa el pedido y lo devuelve al cliente (empleado → cliente).
- 4- El cliente recibe el pedido y continúa el ciclo.



Para esta solución:

Se definen constantes y la estructura mensaje para la comunicación entre los procesos.

KEY: La clave para la cola de mensajes.

PRIORIDAD_VIP y PRIORIDAD_COMUN: Establecen las prioridades de los clientes. Los clientes VIP tienen prioridad (valor menor).

TIPO_HAMBURGUESA, TIPO_VEGANO, TIPO_PAPAS: que representan los tipos de pedido que los clientes pueden hacer.

CANT_CLIENTES: Número total de clientes que se generarán.

PRIMER_CLIENTE: Identificador base para los clientes (comienza en 100).

La estructura mensaje tiene los siguientes campos:

tipo: El tipo de mensaje.

es_vip: Indica si el cliente es VIP (1) o normal (0).

tipo_pedido: Tipo de comida solicitada (hamburguesa, menú vegano o papas fritas).

id_client: ID único del cliente.

Cada tipo de pedido tiene un proceso de empleado específico que se dedica a cocinar y a enviar el menú de vuelta cuando está listo:

EmpleadoHamburguesa(), EmpleadoVegano(), EmpleadoPapa()

Estas funciones simulan los empleados que preparan los pedidos. Cada empleado está esperando un tipo específico de pedido (hamburguesa, menú vegano o papas fritas).

El proceso de cada empleado está dado por el ciclo donde recibe el pedido usando `msgrcv`, simula el tiempo de preparación con un `sleep(1)`, marca el pedido como listo, cambiando el valor de tipo y lo envía de vuelta al cliente usando `msgsnd`.

recibirPedido()

Esta función simula un empleado encargado de recibir los pedidos de los clientes. Los clientes VIP tienen prioridad sobre los normales (esto se hace con el valor negativo en el tipo de mensaje).

El proceso de atención de este empleado es esperar el primer mensaje con prioridad para clientes VIP, imprime el mensaje indicando que está atendiendo al cliente con su id y su tipo de pedido, cambia el tipo de mensaje para marcar el pedido como "preparado" y por último envía el pedido al cliente y espera a que termine.

cliente()

Esta función simula el comportamiento de un cliente realizando un pedido. Cada cliente sigue estos pasos:

El cliente llega y genera un número aleatorio que decide si es VIP o no, y un tipo de pedido aleatorio (hamburguesa, menú vegano, o papas fritas), asigna su id, y el tipo del mensaje está dado por si es vip o no. El cliente envía su pedido a la cola de mensajes. Después de enviar el pedido, espera recibir una confirmación de que su comida está lista. Cuando el pedido es entregado, el cliente se va.

y por último el **main()**

La función **main()** se encarga de inicializar la cola de mensajes y crear los procesos:

Cola de mensajes: Se crea y configura la cola de mensajes con **msgget**.

Creación de empleados: Se crean cinco procesos hijo para los empleados: uno para hamburgues, otro para menú vegano, dos para papas fritas y uno para recibir los pedidos.

Creación de clientes: Se crean 20 procesos de clientes.

Espera de procesos: El proceso principal espera que todos los clientes y empleados terminen antes de proceder.

Eliminación de la cola de mensajes: Finalmente, se elimina la cola de mensajes con **msgctl**

La cola permite asignar prioridades mediante el tipo de mensaje.

Los VIP tienen prioridad (con **PRIORIDAD_VIP = 1**), y los clientes comunes tienen una prioridad inferior. Por otro lado los pedidos se manejan de forma centralizada y son dirigidos a los empleados correspondientes según el tipo de pedido mediante la función **recibirPedido**, esto da flexibilidad a la cola de mensajes para seleccionar mensajes de clientes VIP.

Implementar Cola de mensajes tiene ciertas ventajas, una de ellas es la simplicidad de gestión en la prioridad : La cola de mensajes permite manejar la prioridad de los VIP de manera más limpia, seleccionando directamente los mensajes en orden de prioridad. Otra ventaja que podemos encontrar es la flexibilidad en la solución, las colas de mensajes facilitan la comunicación asíncrona y hay menos probabilidad de bloqueo. Por último destacar que la solución con cola de mensajes es más eficiente en términos de uso de recursos del sistema.

Explicación de cambios realizados

Realice varias modificaciones al código, la principal de la lógica de que el proceso **recibirPedido** se bloqueaba esperando que el pedido esté listo y no había concurrencia fue solucionado, y ahora el modelo presentado sigue el ciclo anteriormente explicado, en donde los empleados de cocina envían el pedido al cliente una vez terminado.

- Elimine **#define MENULISTO 1000**

-Defini la constante `#define PRIMER_CLIENTE 100` para no usarlo en cada llamada, sino que se usa en la creación de los clientes.

-A la hora de definir el struct me encontré con el campo `long mtype` que genero confusión en la implementación del problema con `long tipo`(identificador del tipo de mensaje de la cola) este campo hacía referencia al tipo de pedido por lo que lo renombre `int tipo_pedido` (0 = hamburguesa, 1 = vegano, 2 = papas)

-EMPLEADOS HAMBURGUESA,VEGANO,PAPAS

ya no utiliza el mensaje `pedido.id_client-100` sino que utilizo `pedido.id_client` ya que está inicializado en 100.

ahora asigno como identificador del tipo de mensaje de la cola `-> pedido.tipo = pedido.id_client;` para que el cocinero pueda enviar el pedido al cliente.

-RECIBIR PEDIDO

saque la parte del código de `//Swap` para prioridades

elimine esta parte del código

`msgrcv(queueID, &pedido, msgSize, MENULISTO, 0);` //1000 corresponde a pedido finalizado y listo para entregar

`printf("Pedido entregado a cliente %d.\n", pedido.id_client-100);`

`pedido.tipo = pedido.id_client;`

`msgsnd(queueID, &pedido,msgSize, 0);`

que hacía una espera innecesaria de pedido finalizado y se bloqueaba.

Cliente

disminuí el tiempo de espera ya que me parecía mucho `rand()%10`

modifique `pedido.id_client = id+100;` ya que ahora los creo directamente desde el main a partir de 100.

elimine el `//Swap` para prioridades

modifique `pedido.mtype = pedido.es_vip ? PRIORIDAD_VIP : PRIORIDAD_COMUN;`

esta línea en realidad se confundió con pedido.tipo, en lugar de mtype, y lo que hago es $\text{pedido.tipo} = \text{PRIORIDAD_COMUN} - \text{pedido.es_vip}$; es decir si pedido.es_vip es 0 significa que es cliente común por lo que

$\text{PRIORIDAD_COMUN} - 0 = 2 = \text{PRIORIDAD_COMUN}$, y si es 1, es decir es vip me queda $\text{PRIORIDAD_COMUN} - 1 = 1 = \text{PRIORIDAD_VIP}$

2. Mini Shell.

Esta Mini Shell es un programa de línea de comandos simple que permite al usuario ejecutar un conjunto limitado de comandos relacionados con la gestión de directorios y archivos en un sistema.

- ayuda: Muestra una lista de comandos disponibles y una breve descripción de cada uno cuando se llama a "help" proporcionando información sobre cómo utilizar la Mini Shell.

-crearArchivo: Crea un archivo en la ubicación actual utilizando el nombre que se le pasa como argumento.

-crearDirectorio: se copia el nombre del directorio de argv[1] a la variable dir, utilizando strncpy. Se declara un entero resultado y se intenta crear el directorio con la función mkdir. El segundo parámetro (0777) establece los permisos para el nuevo directorio: rwxrwxrwx: permisos de lectura, escritura y ejecución.

-eliminarDirectorio: se copia el nombre del directorio desde argv[1] a la variable dir, utilizando strncpy. Se intenta eliminar el directorio utilizando la función rmdir.

-listarContenidoDirectorio: Se declara un puntero dire de tipo struct dirent, que se utilizará para almacenar la información de cada entrada de directorio leída. se copia el nombre del directorio desde argv[1] a la variable dir, utilizando strncpy. Si el directorio se abre correctamente, se entra en un bucle que utiliza readdir para leer cada entrada del directorio. Readdir devuelve un puntero a la estructura dirent que contiene información sobre la entrada, o NULL si no hay más entradas.

Una vez que se han leído todas las entradas, se cierra el directorio con closedir(d).

-modificarPermisos: verifica que se hayan pasado al menos tres argumentos. Usa la función stat para obtener los permisos actuales del archivo o directorio en argv[1]. Si falla, muestra un mensaje de error. Modifica los permisos y por último los aplica

-mostrarContenidoArchivo: se copia el nombre del archivo desde argv[1] a la variable dir, utilizando strncpy. Se intenta abrir el archivo especificado por dir en modo lectura ("r"). Se utiliza un bucle while junto con fgets para leer el contenido del archivo línea por línea. Una vez que se han leído todas las líneas o si el archivo no se abre, se cierra el archivo con fclose(f).

-minishell: char comando[MAXIMO]: Almacena el comando ingresado por el usuario.
char d[MAXIMO]: Almacena argumentos adicionales para los comandos (como nombres de directorios o archivos).

char val[100]: Almacena parámetros de permisos para el comando chmod

Un bucle while que continuará ejecutándose hasta que el usuario ingrese "exit". Se solicita al usuario que ingrese un comando y se almacena en la variable comando. Se comparan los comandos ingresados usando strcmp y se ejecutan acciones correspondientes para cada uno. Para cada comando, se utiliza fork() para crear un nuevo proceso hijo. En el proceso hijo, se llama a execv() para ejecutar el programa correspondiente. El proceso padre espera a que el proceso hijo termine su ejecución.

Capturas de casos prueba de la minishell:

- 1) Primero nos posicionamos en donde está ubicada la minishell.
- 2) Cambiamos el permiso a ejecutable del script generado.
- 3) Ejecutamos el script
- 4) La minishell nos muestra el mensaje “Bienvenido a la Mini Shell: Para ver los comandos utilice ‘help’.
- 5) ingresamos help.
- 6) la minishell nos muestra el mensaje de que comandos podemos usar y una breve descripción de para qué sirve cada uno.



```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help
brendam@raspberrypi:~ $ cd /home/brendam/Desktop/Minishell
brendam@raspberrypi:~/Desktop/Minishell $ chmod +x ejecutar.sh
brendam@raspberrypi:~/Desktop/Minishell $ ./ejecutar.sh
Bienvenido a la Minishell: Para ver los comandos utilice 'help'.

-> help
-mkdir nombre: Crea un directorio con el nombre indicado en la ubicacion actual.
-rmdir nombre : Elimina el directorio con ese nombre indicado.
-touch nombre: Crea un archivo con el nombre indicado en la dirección actual.
-o puede indicar el directorio donde crear el archivo de la forma [DIRECTORIO]/[NOMBRE].
-ls nombre: Lista el contenido del directorio con el nombre indicado.
-cat nombre: Muestra el contenido del archivo con el nombre indicado.
-chmod nombre permisos : Cambia los permisos indicados de un archivo en el directorio con ese nombre.
Los comandos de los permisos son:
    Para escritura: '+escritura'0 '-escritura'
    Para lectura: '+lectura' 0 '-lectura'
    Para ejecucion: '+ejecutar'0'-ejecutar'
-exit : Sale de la Minishell.

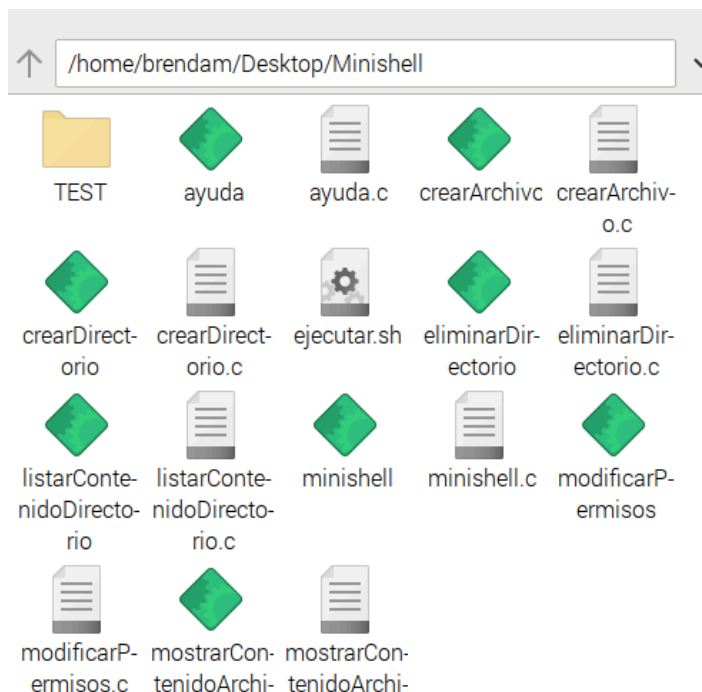
-> █
```

- 7) ingresamos el comando mkdir TEST y nos muestra el mensaje de que se creó correctamente el directorio test (se puede ver en la captura siguiente al comando)
Al querer crear el mismo directorio nos muestra el mensaje de error ya que el directorio ya existe.

```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help
brendam@raspberrypi:~/Desktop/Minishell $ ./ejecutar.sh
Bienvenido a la Minishell: Para ver los comandos utilice 'help'.

-> help
-mkdir nombre: Crea un directorio con el nombre indicado en la ubicacion actual.
-rmdir nombre : Elimina el directorio con ese nombre indicado.
-touch nombre: Crea un archivo con el nombre indicado en la direccion actual.
-o puede indicar el directorio donde crear el archivo de la forma [DIRECTORIO]/[NOMBRE].
-ls nombre: Lista el contenido del directorio con el nombre indicado.
-cat nombre: Muestra el contenido del archivo con el nombre indicado.
-chmod nombre permisos : Cambia los permisos indicados de un archivo en el directorio con ese nombre.
Los comandos de los permisos son:
    Para escritura: '+escritura' o '-escritura'
    Para lectura: '+lectura' o '-lectura'
    Para ejecucion: '+ejecutar' o '-ejecutar'
-exit : Sale de la Minishell.

-> mkdir TEST
Directorio creado correctamente
-> mkdir TEST
Error al crear el directorio: File exists
-> 
```



- 8) Ingresamos `rmdir TEST` y vemos el mensaje de que se ha eliminado correctamente y efectivamente se elimina.
- Al querer hacer nuevamente `rmdir TEST` nos muestra el error de que el directorio no existe.

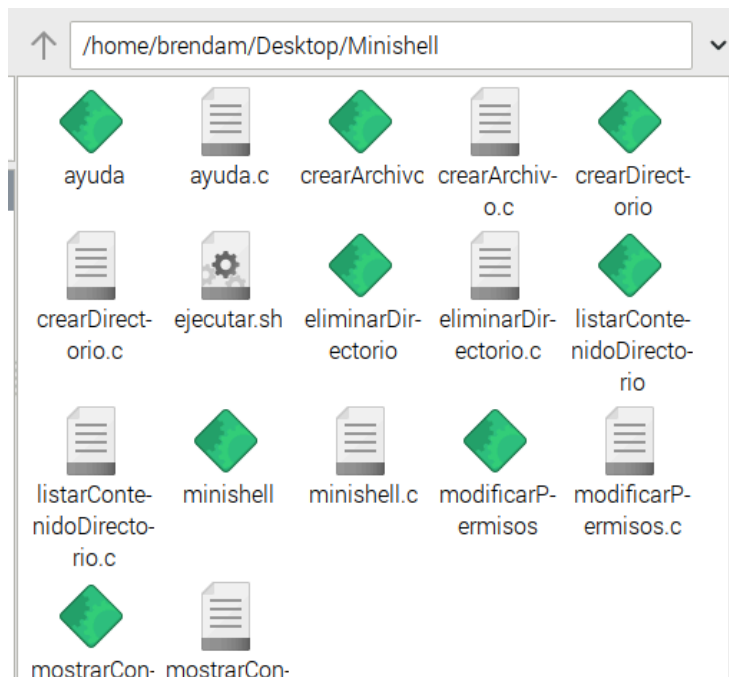
```
brendam@raspberry: ~/Desktop/Minishell
File Edit Tabs Help

-o puede indicar el directorio donde crear el archivo de la forma [DIRECTORIO]/[NOMBRE].
-ls nombre: Lista el contenido del directorio con el nombre indicado.
-cat nombre: Muestra el contenido del archivo con el nombre indicado.
-chmod nombre permisos : Cambia los permisos indicados de un archivo en el directorio con ese nombre.
Los comandos de los permisos son:
    Para escritura: '+escritura' o '-escritura'
    Para lectura: '+lectura' o '-lectura'
    Para ejecucion: '+ejecutar' o '-ejecutar'
-exit : Sale de la Minishell.

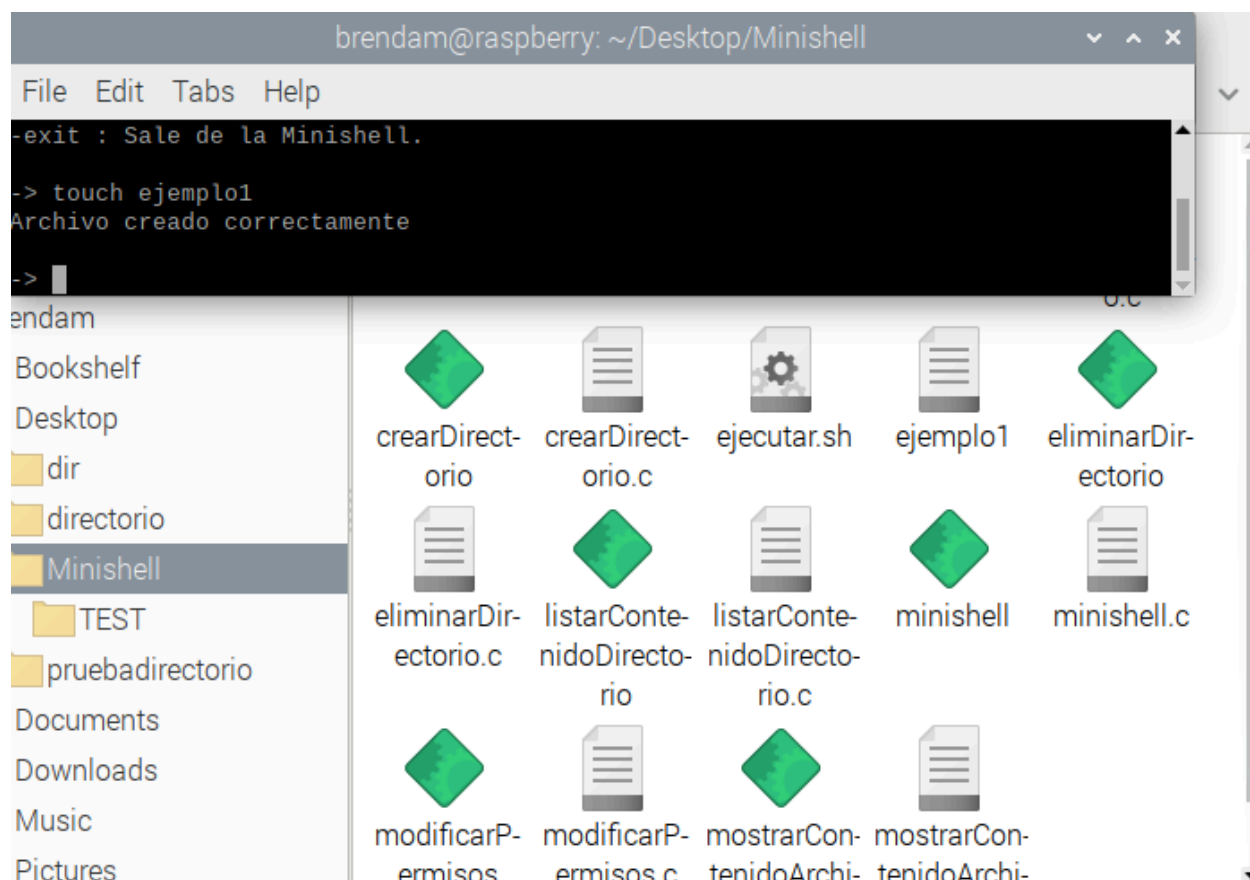
-> mkdir TEST
Directorio creado correctamente
-> mkdir TEST
Error al crear el directorio: File exists

-> rmdir TEST
Directorio eliminado correctamente
-> rmdir TEST
Error al eliminar el directorio

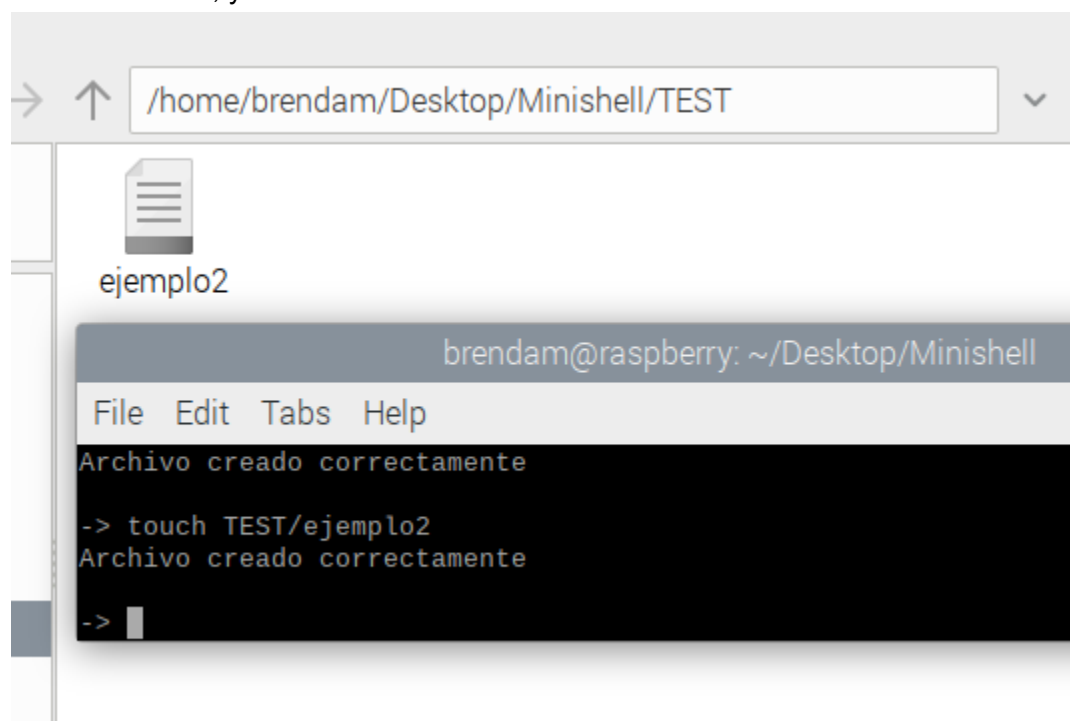
: No such file or directory
->
```



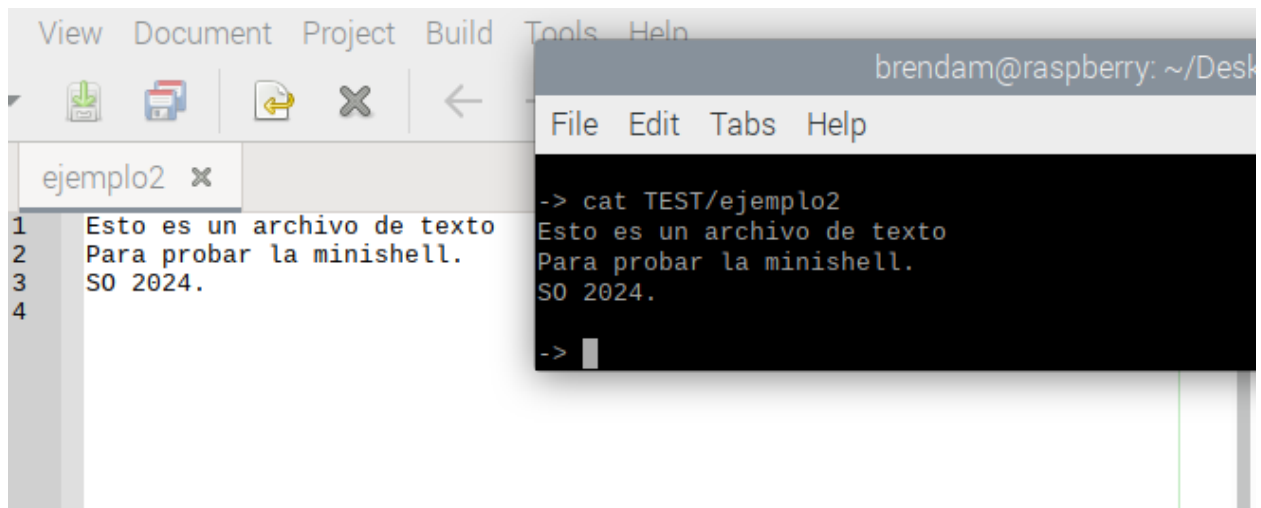
9) Al ingresar el comando touch ejemplo 1 vemos como se crea un archivo en la dirección actual.



Podemos hacer `touch TEST/ejemplo2` para crear un archivo en una dirección especificada, en este caso `TEST`, y vemos cómo se crea correctamente.

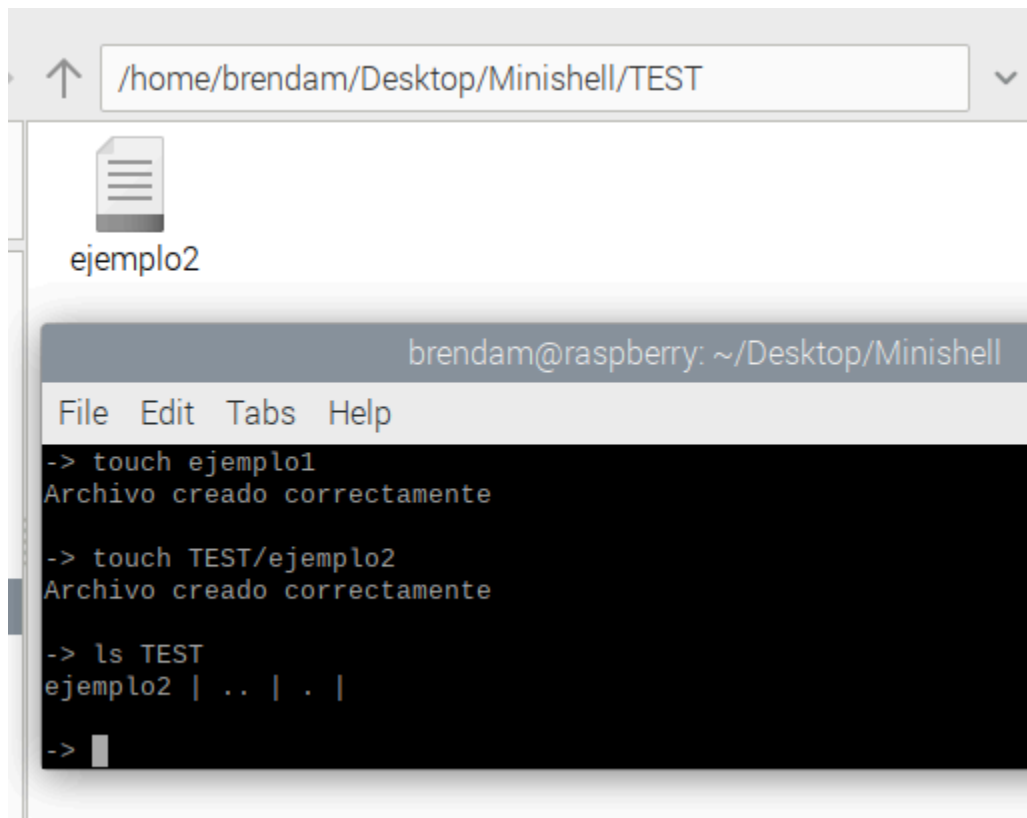


10) Al ingresar `cat TEST/ejemplo2` vemos que nos muestra el contenido del archivo `ejemplo2`.



```
brendam@raspberry: ~/Desk
File Edit Tabs Help
ejemplo2 x
1 Esto es un archivo de texto
2 Para probar la minishell.
3 SO 2024.
4
-> cat TEST/ejemplo2
Esto es un archivo de texto
Para probar la minishell.
SO 2024.
-> |
```

11) Ingresamos el comando `ls TEST` y vemos que lista el contenido del directorio `TEST`



```
↑ /home/brendam/Desktop/Minishell/TEST
ejemplo2
brendam@raspberry: ~/Desktop/Minishell
File Edit Tabs Help
-> touch ejemplo1
Archivo creado correctamente
-> touch TEST/ejemplo2
Archivo creado correctamente
-> ls TEST
ejemplo2 | .. | . |
-> |
```

12) Para verificar el comando de cambio de permisos abrimos otra consola, nos dirigimos a la ubicación de la minishell y mediante el comando `ls -l` podemos ver todos los permisos de los archivos del directorio. Miramos el último... `TEST` que es el que vamos a modificarle los permisos durante esta prueba, inicialmente tiene los 3 permisos (`rwX`)

```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help
brendam@raspberrypi:~$ cd /home/brendam/Desktop/Minishell
brendam@raspberrypi:~/Desktop/Minishell$ ls -l
total 168
-rwxr-xr-x 1 brendam brendam 15520 Nov  4 11:05 ayuda
-rw-rw-rw- 1 brendam brendam 1021 Nov  4 09:50 ayuda.c
-rwxr-xr-x 1 brendam brendam 15640 Nov  4 11:05 crearArchivo
-rw-rw-rw- 1 brendam brendam 378 Nov  3 22:05 crearArchivo.c
-rwxr-xr-x 1 brendam brendam 15644 Nov  4 11:05 crearDirectorio
-rw-rw-rw- 1 brendam brendam 411 Nov  4 09:59 crearDirectorio.c
-rwxrwxrwx 1 brendam brendam 360 Nov  3 14:26 ejecutar.sh
-rw-r--r-- 1 brendam brendam 0 Nov  4 10:48 ejemplo1
-rwxr-xr-x 1 brendam brendam 15644 Nov  4 11:05 eliminarDirectorio
-rw-rw-rw- 1 brendam brendam 482 Nov  4 10:33 eliminarDirectorio.c
-rwxr-xr-x 1 brendam brendam 15752 Nov  4 11:05 listarContenidoDirectorio
-rw-rw-rw- 1 brendam brendam 544 Nov  3 22:19 listarContenidoDirectorio.c
-rwxr-xr-x 1 brendam brendam 15940 Nov  4 11:05 minishell
-rw-rw-rw- 1 brendam brendam 2333 Nov  3 14:32 minishell.c
-rwxr-xr-x 1 brendam brendam 15684 Nov  4 11:05 modificarPermisos
-rw-rw-rw- 1 brendam brendam 1394 Nov  4 09:52 modificarPermisos.c
-rwxr-xr-x 1 brendam brendam 15688 Nov  4 11:05 mostrarContenidoArchivo
-rw-rw-rw- 1 brendam brendam 537 Nov  3 22:38 mostrarContenidoArchivo.c
drwxr-xr-x 2 brendam brendam 4096 Nov  4 11:03 TEST
brendam@raspberrypi:~/Desktop/Minishell$
```

cambiamos el permiso de TEST/ejemplo2, sacando el de lectura, nos notifica que el permiso se modificó correctamente.

en la otra consola podemos ver como cambia el permiso del archivo...siendo este wx.

```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help
e indicado.
-chmod nombre permisos : Cambia los permisos indicados de
un archivo en el directorio con ese nombre.
Los comandos de los permisos son:
    Para escritura: '+escritura' 0 '-escritura'
    Para lectura: '+lectura' 0 '-lectura'
    Para ejecucion: '+ejecutar' 0 '-ejecutar'
-exit : Sale de la Minishell.

-> chmod TEST -lectura
permiso correctamente modificado
->

brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help
-rwxr-xr-x 1 brendam brendam 15684 Nov  4 11:05 modificarPermisos
-rw-rw-rw- 1 brendam brendam 1394 Nov  4 09:52 modificarPermisos
-rwxr-xr-x 1 brendam brendam 15688 Nov  4 11:05 mostrarContenidoA
-rw-rw-rw- 1 brendam brendam 537 Nov  3 22:38 mostrarContenidoA
d-wxr-xr-x 2 brendam brendam 4096 Nov  4 11:03 TEST
brendam@raspberrypi:~/Desktop/Minishell$
```


ingresando `chmod TEST/ejemplo2 +lectura` le damos permiso de lectura al archivo, nos avisa que se cambiaron correctamente los permisos y ahora podemos hacer un `cat` de `TEST/ejemplo2`.

```
-> chmod TEST/ejemplo2 +lectura
permiso correctamente modificado
-> cat TEST/ejemplo2
Esto es un archivo de texto
Para probar la minishell.
SO 2024.
-> 
```

En la otra consola podemos visualizar como se cambió efectivamente el permiso siendo este `rwX`

```
-> chmod TEST +lectura
permiso correctamente modificado
-> 
```

brendam@raspberrypi: ~/Desktop/Minishell

File Edit Tabs Help

-rwxr-xr-x	1	brendam	brendam	15684	Nov	4	11:05	modificarPermisos
-rw-rw-rw-	1	brendam	brendam	1394	Nov	4	09:52	modificarPermisos.c
-rwxr-xr-x	1	brendam	brendam	15688	Nov	4	11:05	mostrarContenidoArchivo
-rw-rw-rw-	1	brendam	brendam	537	Nov	3	22:38	mostrarContenidoArchivo.c
drwxr-xr-x	2	brendam	brendam	4096	Nov	4	11:03	TEST

brendam@raspberrypi:~/Desktop/Minishell \$

ingresando `chmod TEST -escritura`, sacamos el permiso de escritura al directorio `TEST` y vemos que al querer crear un archivo nos notifica que no tiene permiso

```
-> chmod TEST -escritura
permiso correctamente modificado
-> touch TEST/ejemploNuevo
Error al crear el Archivo: Permission denied
-> 
```

en la otra consola podemos ver cómo se quitó el permiso de escritura, quedando así

```
brendam@raspberry: ~/Desktop/Minishell
File Edit Tabs Help

Para escritura: '+escritura' 0 '-escritura'
Para lectura: '+lectura' 0 '-lectura'
Para ejecucion: '+ejecutar' 0 '-ejecutar'
-exit : Sale de la Minishell.

-> chmod TEST -lectura
permiso correctamente modificado
-> chmod TEST +lectura
permiso correctamente modificado
-> chmod TEST -escritura
permiso correctamente modificado
->

brendam@raspberry: ~/Desktop/Minishell
File Edit Tabs Help

-rwxr-xr-x 1 brendam brendam 15684 Nov  4 11:05 modificarPermisos
-rw-rw-rw- 1 brendam brendam  1394 Nov  4 09:52 modificarPermisos.c
-rwxr-xr-x 1 brendam brendam 15688 Nov  4 11:05 mostrarContenidoArchivo
-rw-rw-rw- 1 brendam brendam   537 Nov  3 22:38 mostrarContenidoArchivo.c
-r-xr-xr-x 2 brendam brendam  4096 Nov  4 11:03 TEST
brendam@raspberry:~/Desktop/Minishell $
```

Ingresando `chmod TEST +escritura` le damos permiso de escritura al directorio TEST, probamos de crear de un archivo y se crea correctamente.

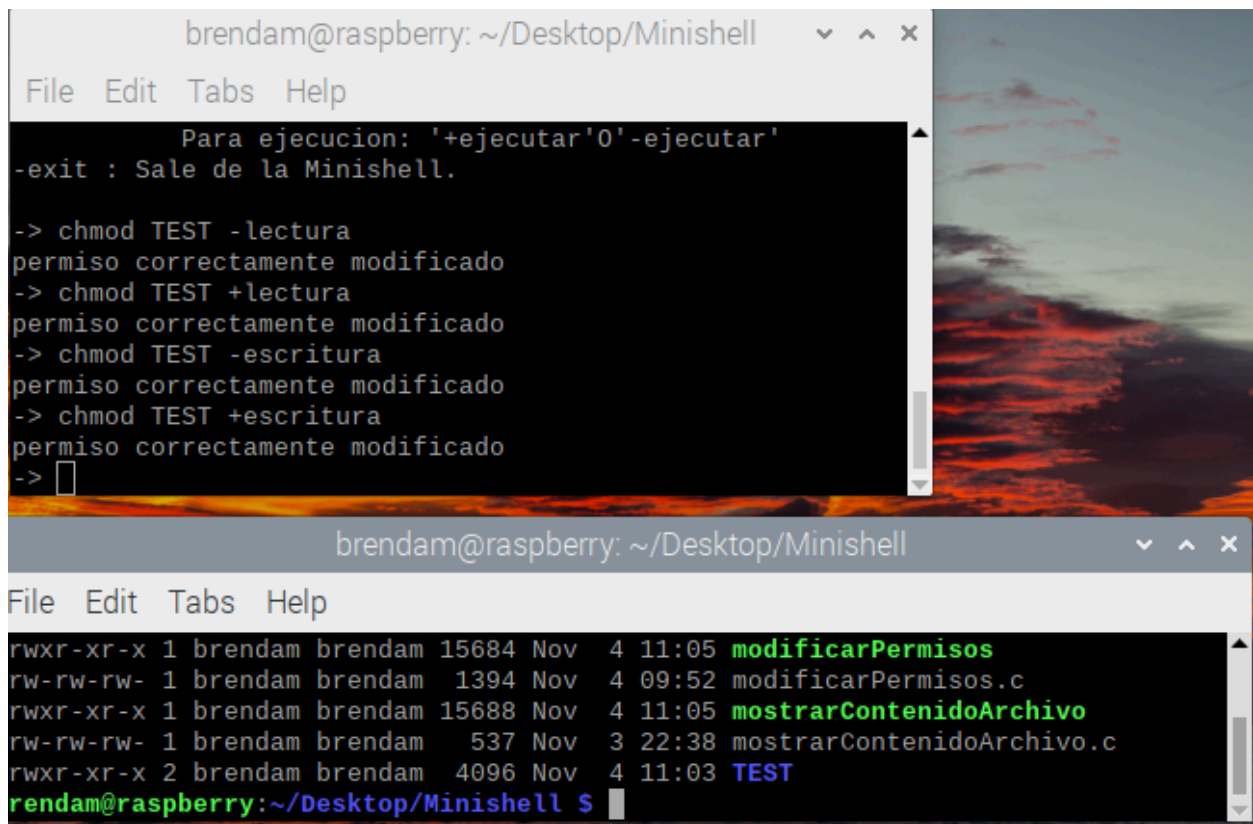
```
brendam@raspberry: ~/Desktop/Minishell
File Edit Tabs Help

Error al crear el Archivo: Permission denied

-> chmod TEST +escritura
permiso correctamente modificado
-> touch TEST/ejemploNuevo
Archivo creado correctamente

->
```

En la otra consola podemos ver el cambio de permisos



```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help

Para ejecucion: '+ejecutar'0'-ejecutar'
-exit : Sale de la Minishell.

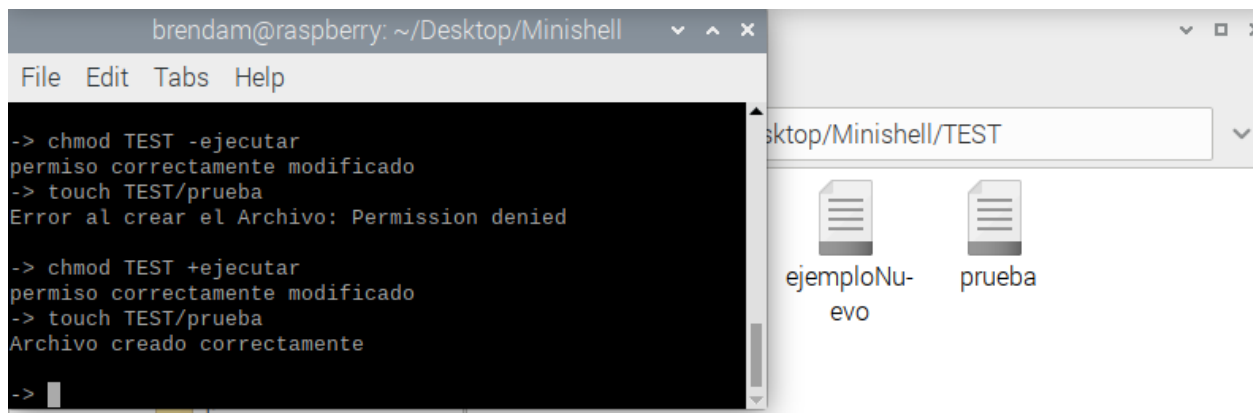
-> chmod TEST -lectura
permiso correctamente modificado
-> chmod TEST +lectura
permiso correctamente modificado
-> chmod TEST -escritura
permiso correctamente modificado
-> chmod TEST +escritura
permiso correctamente modificado
->

brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help

-rwxr-xr-x 1 brendam brendam 15684 Nov  4 11:05 modificarPermisos
-rw-rw-rw- 1 brendam brendam  1394 Nov  4 09:52 modificarPermisos.c
-rwxr-xr-x 1 brendam brendam 15688 Nov  4 11:05 mostrarContenidoArchivo
-rw-rw-rw- 1 brendam brendam   537 Nov  3 22:38 mostrarContenidoArchivo.c
-rwxr-xr-x 2 brendam brendam  4096 Nov  4 11:03 TEST
brendam@raspberrypi:~/Desktop/Minishell $
```

Al ingresar el comando `chmod TEST -ejecutar` le cambiamos el permiso al directorio TEST y vemos que no se puede crear un archivo.

Al ingresar el comando `chmod TEST +ejecutar` le cambiamos el permiso al directorio TEST y podemos crear un archivo.

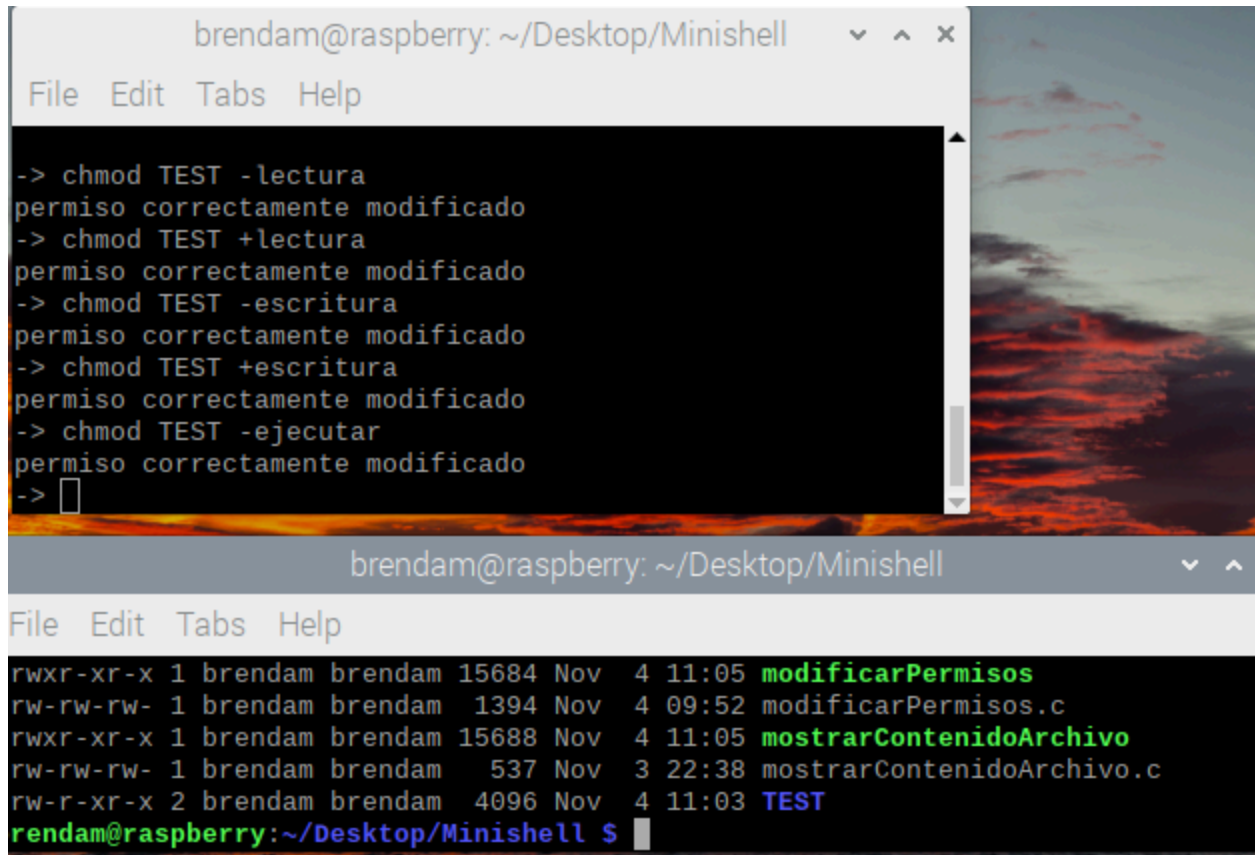


```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help

-> chmod TEST -ejecutar
permiso correctamente modificado
-> touch TEST/prueba
Error al crear el Archivo: Permission denied

-> chmod TEST +ejecutar
permiso correctamente modificado
-> touch TEST/prueba
Archivo creado correctamente
->
```

En la siguiente captura se ve como cambia el permiso al hacer `-ejecutar`



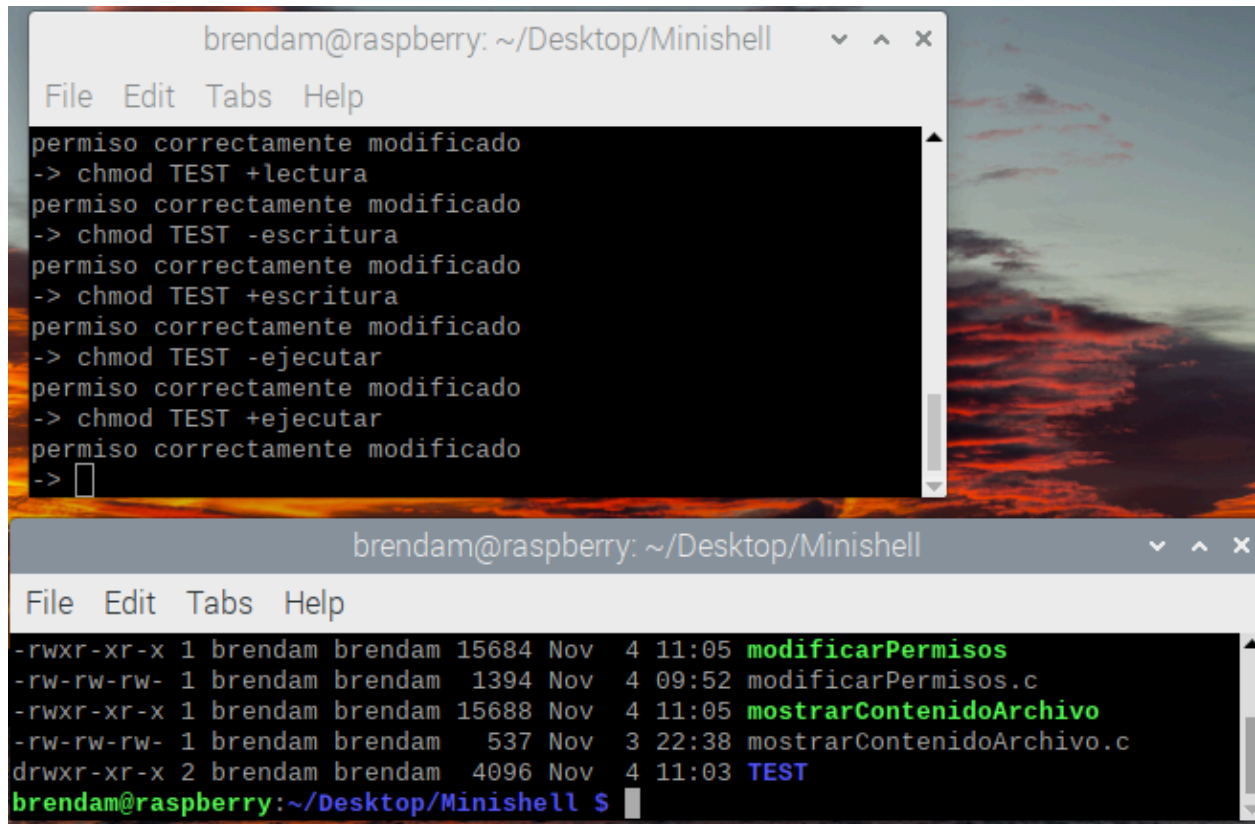
```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help

-> chmod TEST -lectura
permiso correctamente modificado
-> chmod TEST +lectura
permiso correctamente modificado
-> chmod TEST -escritura
permiso correctamente modificado
-> chmod TEST +escritura
permiso correctamente modificado
-> chmod TEST -ejecutar
permiso correctamente modificado
-> █

brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help

-rwxr-xr-x 1 brendan brendan 15684 Nov  4 11:05 modificarPermisos
-rw-rw-rw- 1 brendan brendan  1394 Nov  4 09:52 modificarPermisos.c
-rwxr-xr-x 1 brendan brendan 15688 Nov  4 11:05 mostrarContenidoArchivo
-rw-rw-rw- 1 brendan brendan   537 Nov  3 22:38 mostrarContenidoArchivo.c
-rw-r-xr-x 2 brendan brendan  4096 Nov  4 11:03 TEST
brendam@raspberrypi:~/Desktop/Minishell $ █
```

en la siguiente captura se ve como cambia el permiso al hacer +ejecutar

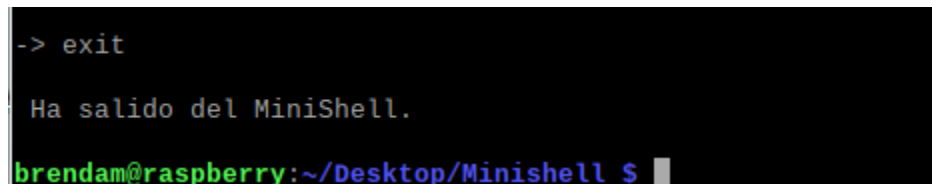


The image shows two screenshots of a terminal window on a Raspberry Pi. The window title is 'brendam@raspberrypi: ~/Desktop/Minishell'. The top screenshot shows a series of 'chmod' commands being executed on a file named 'TEST', with each command followed by the message 'permiso correctamente modificado'. The bottom screenshot shows the output of a 'ls -l' command, displaying a directory listing with permissions, owner, group, size, date, and filename. The files listed are 'modificarPermisos', 'modificarPermisos.c', 'mostrarContenidoArchivo', 'mostrarContenidoArchivo.c', and 'TEST'.

```
brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help
permiso correctamente modificado
-> chmod TEST +lectura
permiso correctamente modificado
-> chmod TEST -escritura
permiso correctamente modificado
-> chmod TEST +escritura
permiso correctamente modificado
-> chmod TEST -ejecutar
permiso correctamente modificado
-> chmod TEST +ejecutar
permiso correctamente modificado
-> 

brendam@raspberrypi: ~/Desktop/Minishell
File Edit Tabs Help
-rwxr-xr-x 1 brendam brendam 15684 Nov  4 11:05 modificarPermisos
-rw-rw-rw- 1 brendam brendam  1394 Nov  4 09:52 modificarPermisos.c
-rwxr-xr-x 1 brendam brendam 15688 Nov  4 11:05 mostrarContenidoArchivo
-rw-rw-rw- 1 brendam brendam   537 Nov  3 22:38 mostrarContenidoArchivo.c
drwxr-xr-x 2 brendam brendam  4096 Nov  4 11:03 TEST
brendam@raspberrypi:~/Desktop/Minishell $
```

13) Por último ingresamos exit para salir de la mini shell y nos muestra el mensaje “ Ha salido del Minishell.



The image shows a terminal window with the 'exit' command being entered and executed. The output is 'Ha salido del MiniShell.' followed by the prompt 'brendam@raspberrypi:~/Desktop/Minishell \$'.

```
-> exit
Ha salido del MiniShell.
brendam@raspberrypi:~/Desktop/Minishell $
```

1.2. Sincronización

Para la ejecución de cada uno de los archivos se creo un script ejecutar.sh debe ingresar

cd [ruta del directorio]

y posicionado en dicho directorio en caso de no poseer los permisos de ejecucion ingrese

chmod +x ejecutar.sh

luego para ejecutar el script ingrese

./ejecutar.sh

1. Taller de Motos.

Archivo con la resolución adjunta.

-Para resolver este problema se utilizaron 6 semaforos

- **ruedas_armadas:** controla la cantidad de ruedas listas
- **chasis_listo:** indica que el chasis está listo
- **motor_listo:** indica que el motor se agregó.
- **pintura_hecha:** indica que la moto ha sido pintada.
- **equipamiento_listo:** indica que la moto está lista para agregar equipamiento extra.
- **inicio:** permite que la construcción del ciclo de motos inicie (cada dos motos) .

- 6 hilos, uno para cada operario pthread_t operarios[6]

- La inicializacion de los semaforos es

ruedas_armadas en 2

chasis_listo en 0

motor_listo en 0

pintura_hecha 0

equipamiento_listo 0

inicio 4

-Cada operario realiza una tarea específica en un ciclo infinito , esperando a que se cumplan las condiciones necesarias mediante la sincronización con semáforos.

- **operario_armar_ruedas():** Para cada una de las ruedas decrementa una unidad del semáforo inicio y una unidad de ruedas_armadas.
Arma rueda e incrementa el semáforo chasis_listo para permitir que se arme el chasis.
- **operario_armar_chasis() :** Espera dos ruedas listas (consume dos veces el semáforo chasis_listo). Arma el chasis e incrementa el semáforo motor_listo para permitir que se arme el motor

- **operario_agregar_motor()** : Espera a que el chasis esté listo (decrementa el semáforo motor_listo). Agrega el motor e incrementa el semáforo pintura_hecha permitiendo que se pinte la moto.
- hay dos operarios a cargo de la pintura **operario_pintar_rojo()** y **operario_pintar_verde()** Ambos comparten el semáforo pintura_hecha. Esperan a que el motor esté listo (decrementan el semaforo pintura_hecha). Pintan la moto (roja o verde), e incrementan una unidad equipamiento_listo (la moto pasa a la etapa final) y dos unidades ruedas_armadas (libera ruedas para la siguiente moto).
- **operario_agregar equipamiento()** : Espera dos motos pintadas listas (equipamiento_listo consume dos veces, para que en la primer moto no se agregue el equipamiento extra, en la segunda si, en la tercera no y asi sucesivamente). Agrega equipamiento extra, e incrementa 4 unidades del semáforo inicio para reiniciar el proceso (2 ruedas y 2 motos).

Esta solucion la pense como una secuencia de letra representada por :

AABC(D o E)AABC(D o E)FAABC(D o E)AABC(D o E)F...

donde

A: armar rueda

B: armar chasis

C: agregar motor

D: pintar moto de rojo

E: pintar moto de verde

F: agregar equipamiento extra.

Explicación de cambios realizados

En la primer entrega del proyecto este ejercicio estaba implementado de una manera incorrecta, contenia un mutex innecesario para pintar la moto, el operario de armar ruedas hacia 3 print para simular agregar rueda 1, rueda 2 e imprimir 2 ruedas listas, pero esto no es correcto ya que no hacia un uso correcto del semáforo. Ademas agregar_equipamiento_extra no estaba correctamente sincronizado mediante el uso de semáforos, en su lugar usaba `int agregar = rand() % 2;` para determinar si se agregaba o no, lo cual es incorrecto.

A la hora de reentregar pensé la solución como un ejercicio típico de sincronización de secuencia de letras, pensándolo de esta manera, saque el mutex e hice que la sincronización sea exclusivamente mediante el uso de semáforos (y no utilizando print o función random) Agregue un semáforo extra inicio para que cada dos motos se repita el ciclo y para que la primer moto no tenga equipamiento extra, la segunda si, la tercera no y así sucesivamente.

2. Santa Claus.

a) Archivo adjunto con la resolución.

b) La resolución de este problema mediante el uso de hilos y semáforos está dada por:

-hilos:

threadRenos[CANT_RENOS]

threadElfos[CANT_ELFOS]

threadSanta

conde CANT_RENOS es 9 y CANT_ELFOS es 9.

-6 semáforos en donde:

- **sem_santa**: Santa Claus se duerme hasta que un evento lo despierte.(semáforo binario).
- **sem_nueveRenos**: Indica a los renos cuándo pueden irse después de ser atendidos(semáforo contador). Se incrementa (sem_post) cuando Santa termina de enganchar a todos los renos.
- **sem_tresElfos**:Indica a los tres elfos cuándo pueden regresar al trabajo después de ser ayudados(semáforo contador). Se incrementa (sem_post) cuando Santa resuelve los problemas.
- **sem_renos** : Cuenta cuántos renos llegaron al Polo Norte(semáforo contador). Se decrementa cada vez que un reno llega (sem_wait) y se incrementa nuevamente tras ser atendido (sem_post).
- **sem_elfos**:Controla el número de elfos con problemas.(semáforo contador).Se decrementa cada vez que un elfo con problemas entra en el grupo (sem_wait) y se incrementa después de que Santa los ayuda (sem_post).
- **sem_grupoElfos**: Controla cuántos elfos pueden formar un grupo para pedir ayuda.(semáforo contador)

Inicialización

sem_santa = 0 (Santa dormido)

sem_nueveRenos = 0 (ningún grupo de 9 renos listos)

sem_tresElfos = 0 (ningún grupo de 3 elfos esperando ayuda)

sem_renos = 9 (porque todos los renos comienzan fuera del Polo.)

sem_elfos = 3 (capacidad inicial para 3 elfos)

sem_grupoElfos=3 (limitando el tamaño de los grupos de elfos con problemas)

-Dos mutex:

- **mutexRenos** : Mutex para asegurar que solo un hilo reno acceda a la sección crítica del código.
- **mutexElfos**: Mutex para asegurar que solo un solo hilo elfo acceda a su sección crítica del código.

Santa Claus:

Santa Claus permanece dormido (`sem_wait(&sem_santa)`) hasta que un evento lo despierte. Se despierta si: Llegan 9 renos que necesitan ser enganchados al trineo o 3 elfos tienen problemas y piden ayuda. Atiende primero a los renos (si todos están presentes) y luego a los elfos (si están en grupos de 3).

Usa `sem_trywait` para verificar si han llegado los 9 renos. Si es así, engancha los renos al trineo y los renos quedan libres y regresan.

Usa `sem_trywait` para verificar si hay un grupo de 3 elfos esperando ayuda. Si es así, ayuda a los elfos (`sem_post` en `sem_tresElfos`). Los elfos quedan libres para regresar al trabajo.

Santa vuelve a dormir después de atender a los renos o elfos.

Renos:

Los renos llegan al Polo Norte. Cuando llegan los 9 renos, despiertan a Santa Claus para que los enganche al trineo. Usa un mutex para actualizar el estado de los renos. Si es el último reno (noveno), despierta a Santa (`sem_post(&sem_santa)`). Espera hasta que Santa termine de enganchar a todos los renos (`sem_wait(&sem_nueveRenos)`).

Elfos:

Los elfos trabajan de manera independiente y, ocasionalmente, encuentra un problema. Si un elfo encuentra un problema, usa un mutex para agruparse con otros elfos. Cuando 3 elfos tienen problemas, despiertan a Santa Claus para que los ayude. Espera a que Santa resuelva el problema (`sem_wait(&sem_tresElfos)`). Una vez ayudados, los elfos regresan al trabajo.

Explicación de cambios realizados

-Modifique la cantidad de elfos creados y además agregue la cantidad de elfos que necesitan ser ayudados para utilizar los semáforos:

```
#define CANT_RENOS 9
#define CANT_ELFOS 9
#define CANT_ELFO_AYUDA 3
```

-Agregue un nuevo semáforo `sem_grupoElfos` para controlar cuántos elfos pueden intentar formar un grupo para pedir ayuda a Santa.

Esto garantiza que el grupo de elfos que necesita ayuda esté bien sincronizado antes de despertar a Santa. Si no hay suficientes elfos para formar un grupo de 3, el semáforo impide que pidan ayuda. Este semáforo es manejado exclusivamente por los elfos. Lo incrementan una vez que recibieron ayuda de Santa, liberándolos.

-En lugar de crear 9 hilos renos y hacer que cada uno consuma los 9 semáforos, ahora los renos comparten el semáforo `sem_renos` para controlar la llegada de los renos. El último reno en llegar (cuando `sem_trywait(&sem_renos)` devuelve -1) despierta a Santa.

Esto asegura que solo el último reno despierte a Santa.

-En lugar de esperar a que los elfos se sincronicen de manera individual, ahora los elfos comparten el semáforo `sem_elfos`, y si se forman 3 elfos con problemas, estos despiertan a Santa.

Esta implementación agrupa a los elfos que tienen problemas, reduciendo el número de interacciones con Santa y manteniendo una sincronización eficiente entre ellos. Al usar `sem_trywait`, los elfos esperan hasta que puedan formar un grupo de tres.

-Antes, Santa se despertaba cada vez que un solo reno o elfo llegaba, lo cual era incorrecto. Ahora Santa solo se despierta cuando es necesario (cuando hay 9 renos o 3 elfos con problemas).

-Agregue identificador de hilo para poder ver con claridad qué hilo está en ejecución en las distintas partes del código

-Se asegura que los renos y elfos liberen semáforos apropiadamente después de ser atendidos o haber completado su tarea. El semáforo `sem_nueveRenos` se utiliza para asegurar que todos los renos han sido atendidos antes de que continúen, y el semáforo `sem_tresElfos` garantiza que Santa solo ayude a grupos de 3 elfos a la vez.

2. Problemas

2.1. Lectura

Seguridad: Fallo de integridad Presentación adjunta.

2.2. Problemas Conceptuales

1. Paginación y Segmentación en Memoria

Dirección lógica (16 bits) 0011000000110011

en decimal : 12339

a) Número de página y desplazamiento:

- **Tamaño de página:** 512 direcciones.

Se divide a la dirección lógica según el número de bits correspondientes al

Número de página(P) y al Desplazamiento (D).

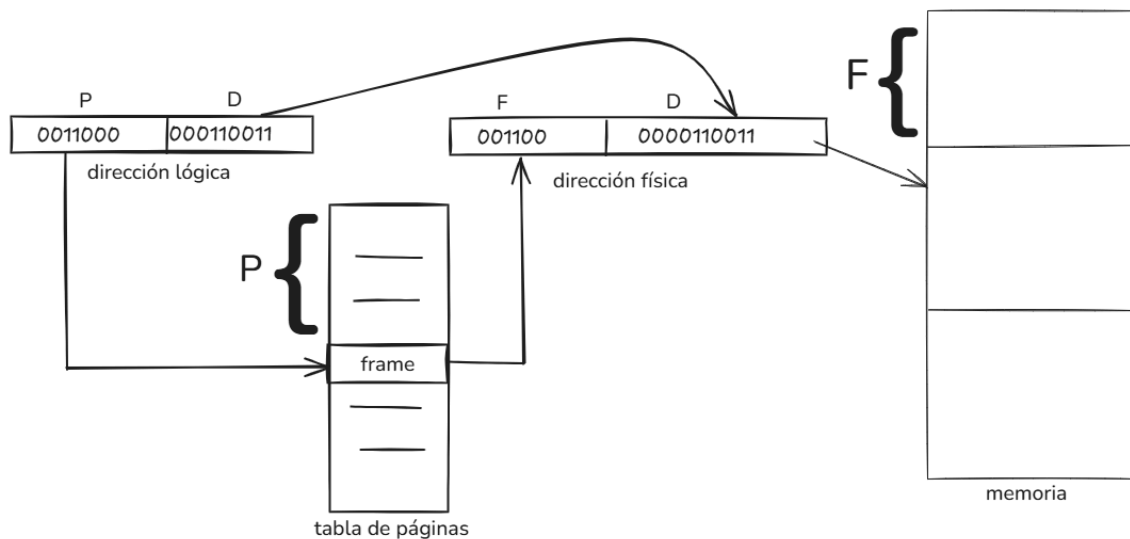
- Como $512 = 2^9 \rightarrow$ **Desplazamiento (D)= 9 \rightarrow 000110011** (en binario), que equivale a **51** en decimal.
- $16 \text{ bits} - 9 \text{ bits} = 7 \text{ bits}$, por lo que **Número de página(P)= 7 bits \rightarrow 0011000** (en binario), que equivale a **24** en decimal.

Número de marco:

- Por enunciado el número de marco(M) es la mitad del número de página(P) \rightarrow **$M = P/2$**
- Como $P=24$, el número de marco será **$M=24/2=12$**

Dirección física:

- Número de marco en binario (12 en decimal) es **0001100**
- El desplazamiento era **000110011** (9 bits).
- Luego la dirección física es: **0001100000110011** (en binario)



b)

Tamaño máximo del segmento: 2K direcciones.

- $2K=2048 \rightarrow 2^{11}$, lo que significa que **11 bits** están destinados al desplazamiento.
- Como la dirección lógica tiene 16 bits. $16-11 = \mathbf{5 \text{ bits}}$ que se utilizan para el número de segmento.

Número de segmento y desplazamiento:

- Para la dirección lógica **0011000000110011**:

Número de segmento (5 bits): **00110** (en binario), que sería segmento **6** en decimal.

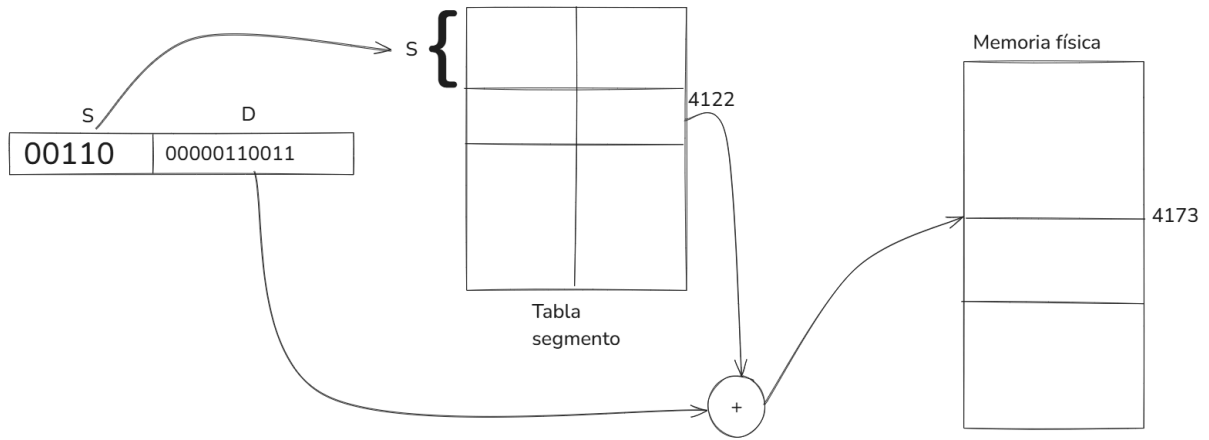
Desplazamiento (11 bits): **00000110011** (en binario), que sería desplazamiento **51** en decimal.

Dirección base del segmento:

- La base para cada segmento se coloca regularmente en direcciones reales, calculadas como: $20 + 4,096 + \text{Nro Segmento}$
- Para el segmento 6: Base = $20+4096+6=4122$. y en binario: 1000000011010

Dirección física:

- La dirección física se obtiene sumando la base del segmento y el desplazamiento.
- Dirección física = $4122 + 51 = 4173$. y en binario: 1000001001101



2. Tabla de páginas

Datos:

- Direcciones virtuales de 16 bits
- Direcciones físicas de 16 bits
- Páginas de 4096 bytes (2^{12} , por lo tanto, 12 bits para el offset).
- Algoritmo de reemplazo de páginas LRU

a)

0x621C

en binario : 0110001000011100

donde los primeros 4 bits corresponden al numero de pagina y los restantes 12 bits al desplazamiento teniendo así:

- Página: 0110 -> 6 en decimal.
- Desplazamiento: 001000011100 -> 540 en decimal.
- Para la página 6 el marco es 8 (1000) en memoria.
- luego la dirección física es el marco+desplazamiento, quedando así:
Dirección física: 1000001000011100 en binario y 0x821C en hexadecimal.
- Actualizar el bit de referencia a 1 a la página 6.

0xF0A3

en binario : 1111000010100011

donde los primeros 4 bits corresponden al numero de pagina y los restantes 12 bits al desplazamiento teniendo así:

- Página: 1111 -> 15 en decimal.
- Desplazamiento : 000010100011 -> 163 en decimal.
- Para la página 15 el marco es 2 (0010) en memoria.
- Luego la dirección física es el marco+desplazamiento, quedando así:
Dirección física: 0010000010100011 en binario y 0x20A3 en hexadecimal.
- Actualizar el bit de referencia a 1 a la página 15.

0xBC1A

en binario : 1011110000011010

donde los primeros 4 bits corresponden al numero de pagina y los restantes 12 bits al desplazamiento teniendo así:

- Página: 1011 -> 11 en decimal.

- Desplazamiento: 110000011010 -> 3098 en decimal.
- Para la página 11 el marco es 4 (0100) en memoria.
- Luego la dirección física es el marco+desplazamiento, quedando así:
Dirección física: 0100110000011010 en binario y 0x4C1A en hexadecimal.
- Actualizar el bit de referencia a 1 a la página 11.

0x5BAA

en binario : 0101101110101010

donde los primeros 4 bits corresponden al numero de pagina y los restantes 12 bits al desplazamiento teniendo así:

- Página: 0101 -> 5 en decimal.
- Desplazamiento: 101110101010 -> 2986 en decimal.
- Para la página 5 el marco es 13 (1101) en memoria.
- Luego la dirección física es el marco+desplazamiento, quedando así:
Dirección física: 1101101110101010 en binario y 0xDBAA en hexadecimal.
- Actualizar el bit de referencia a 1 a la página 5.

0x0BA1

en binario : 0000101110100001

donde los primeros 4 bits corresponden al numero de pagina y los restantes 12 bits al desplazamiento teniendo así:

- Página: 0000-> 0 en decimal
- Desplazamiento: 101110100001 -> 2977 en decimal.
- Para la página 0 el marco es 9 (1001) en memoria.
- Luego la dirección física es el marco+desplazamiento, quedando así:
Dirección física: 1001101110100001 en binario y 0x9BA1 en hexadecimal.
- Actualizar el bit de referencia a 1 a la página 0.

La tabla quedaría de la siguiente manera:

Página	Marco	Bit de referencia
0	9	1
1	-	0
2	10	0
3	15	0
4	6	0
5	13	1
6	8	1
7	12	0
8	7	0
9	-	0
10	5	0
11	4	1
12	1	0
13	0	0
14	-	0
15	2	1

b) Dirección lógica : 0x9001

en binario es : 1001000000000001

y su página es 9 (1001) Como la página 9 no tiene un marco asignado en memoria hay un fallo de página.

c) El algoritmo de reemplazo de páginas menos usadas recientemente (LRU) : El algoritmo LRU selecciona el marco que no ha sido utilizado durante el período más largo. Cuando se produce un fallo de página (es decir, cuando se intenta acceder a una página que no está en memoria), se revisan todos los marcos ocupados y se determina cuál fue el menos utilizado recientemente. Para implementar el LRU por completo, es necesario mantener una lista enlazada de todas las páginas en memoria, con la página de uso más reciente en la parte frontal y la de uso menos reciente en la parte final.

entonces tenemos que los marcos con bit de referencia 0, es decir los que no han sido referenciados recientemente : Marco 10 (Página 2), Marco 15 (Página 3), Marco 6 (Página 4), Marco 12 (Página 7), Marco 7 (Página 8), Marco 5 (Página 10), Marco 1 (Página 12), Marco 0 (Página 13)

y los marcos con bit 1 tenemos

Marco 9 (Página 0), Marco 13 (Página 5), Marco 8 (Página 6), Marco 4 (Página 11), Marco 2 (Página 15)