Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Sistemas Operativos

# Trabajo Práctico Minix

Ejercicios Optativos

Fecha de Entrega: 14/7/05



| Grupo 2 | LU | Correo electrónico |
|---|---|---|
| Valdez Miguel Angel | 498/02 | crack57ar@gmail.com |
| Di Vincenzo | 284/02 | jdivincenzo@gmail.com |



**RESUMEN**

En el presente trabajo resolvemos puntualmente tres problemas, dos de ellos, clásicos dentro del mundo de los sistemas operativos. Primeramente, resolvemos el problema de Lectores/Escritores con prioridad Lectores; luego el de los Filósofos Chinos; y por último, presentamos un programa que muestra por pantalla la información que posee el inodo del archivo pasado como parámetro por el usuario.

**PALABRAS CLAVE**

Minix - Lectores/Escritores - Filósofos Chinos - Inodo

# Índice

# Introducción

## Enunciado

**- Sistemas Operativos -**
**Trabajo Práctico Minix**

---

**Objetivos del práctico**

Al terminar este trabajo Ud. habrá aprendido a:

- Instalar MINIX 2.0 en su versión DOSMINIX o sobre un simulador.
- Utilizar convenientemente los principales comandos de MINIX.
- Compilar y ejecutar programas escritos en lenguaje C.
- Aplicar en forma práctica algunos de los conocimientos brindados en la materia.
- Realizar modificaciones al Sistema Operativo MINIX.

**Herramientas necesarias**

Para resolver los ejercicios propuestos necesitará:

- Una PC con Windows 95/98 con, al menos, 80 Mb de Disco o el entorno necesario para instalar el simulador elegido.

**Requisitos de Entrega**

Lugar y Fecha de entrega:

- La fecha y hora de entrega para este práctico es la que figura en el cronograma de la materia. (se alienta y acepta la entrega del trabajo, en su totalidad, en forma anticipada)
- Los trabajos deben ser entregados PERSONALMENTE a alguno de los docentes de la cátedra en los horarios de clase o de consulta. No se aceptarán trabajos depositados en el Departamento de Computación o en cualquier otro lugar.
- No se aceptarán trabajos incompletos.

**Formato de Entrega**

Se deberá entregar en un medio digital, preferentemente CD, una o varias imágenes del sistema operativo, con los cambios efectuados al mismo y que contenga además todas las demás resoluciones en formato fuente, formato ejecutable y los programas de prueba que se utilizaron para comprobar que los cambios o resoluciones funcionan correctamente. Además se deberán entregar todos los archivos necesarios para que esa imagen o imágenes ejecuten perfectamente.
Se deberá, además, entregar un DOCUMENTO IMPRESO. Ese documento debe reunir las siguientes características:

1. Formato de Presentación
   *a*) Impreso en hojas de tamaño A4 encarpetadas.
2. Secciones del documento (Todas obligatorias):
   *a*) Carátula de presentación: Debe incluir OBLIGATORIAMENTE:
      1) Asignatura
      2) Número y Descripción del trabajo práctico
      3) Año y Cuatrimestre de Cursado
      4) Identificación del Grupo
      5) Nombres, Apellidos y direcciones de correo electrónico de TODOS los Integrantes del grupo
3. Sección Principal: Aquí debe incluirse la resolución de cada uno de los problemas planteados. Para cada respuesta debe indicarse OBLIGATORIAMENTE, el número y título del problema al que corresponde tal como aparece en el enunciado y los comandos y/o programas utilizados para resolverlos. Se deberá indicar claramente en que directorio y bajo que nombre se encuentran los fuentes, los ejecutables y los programas de prueba

**Cambios al enunciado del práctico, fechas de entrega, etc.**

Cualquier cambio en los enunciados, fechas de entrega, etc. será informado utilizando dos métodos:

- La página Web de la Materia.
- La lista de correo sisop@listas.dc.uba.ar.

Ud. no puede alegar que no estaba al tanto de los cambios si esos cambios fueron anunciados utilizando alguno de los dos métodos.
SUGERENCIA: Consulte frecuentemente la página de la materia y asegúrese de que ha

sido incorporado a la lista de correos.
Los grupos serán de hasta un máximo de tres (3) integrantes.

# Principal

## Ejercicios

Ayudas:
`http://www.dc.uba.ar/people/materias/so/html/minix.html`

Setear Teclado Español

```
cp /usr/lib/keymaps/spanish.map /etc/keymap
echo loadkeys /etc/keymap >> /etc/rc
sync; sync; shutdown
```

Pruebe el ash, el man, el apropos y el mined

# 13. Lectores-Escritores

Resuelva el problema de Lectores/Escritores con prioridad Lectores. (Tests o pruebas mencionados en Forma de entrega).

## 13.1. Introducción

Este problema surgió en un principio para modelar el acceso de varios procesos a un mismo registro de una base de datos. En esta situación no es prohibitivo que varios procesos accedan al recurso simultáneamente, siempre y cuando los accesos sean solo de lectura. Diferente es el caso en el que un proceso esta accediendo al recurso para actualizarlo, ya que ningún otro proceso podrá accederlo hasta que el anterior termine de actualizar el recurso. En este problema, los procesos que acceden al recurso para modificarlo son llamados escritores y los procesos que acceden al recurso solo para lectura son llamados lectores.

Los lectores que intenten acceder al recurso mientras un escritor lo esta actualizando, se deberán encolar a la espera de que el escritor libere el recurso. De la misma manera se encolaran los escritores que intenten acceder a un recurso que esta siendo accedido por un lector.

El hecho de que los lectores tengan prioridad sobre los escritores implica que si un escritor está esperando para que uno o mas lectores liberen el recurso y llega otro lector, este podrá acceder al recurso aunque el escritor ya estaba encolado esperando a que se libere. Este esquema presenta la desventaja de que permite que haya inanición de escritores ya que si la tasa de arribo de lectores es menor que el tiempo de lectura del recurso, e ingresan lectores constantemente, los escritores no podrán acceder al recurso hasta que la totalidad de los lectores lo liberen.

Es fácil ver que si en este problema ni los lectores ni los escritores tiene prioridad, no tendríamos problemas de inanición para ningún tipo de proceso ya que los lectores que llegan mientras hay un escritor esperando, deberán encolarse detrás de este. La desventaja de esta solución es que permite menor concurrencia.

## 13.2. Implementación

En nuestra implementación utilizamos un archivo como recurso compartido entre los procesos. En este buffer cada proceso escritor escribirá su id de proceso. Los lectores lo abren como sólo lectura, mientras que los escritores como sólo escritura. Vale la pena aclarar que siempre comienza un escritor, para salvar el caso en que el archivo no exista y se cree uno nuevo, caso en que los lectores no tendrían nada para leer hasta que no den lugar al primer escritor.

Utilizamos tres semáforos para controlar el acceso al buffer permitiendo la concurrencia únicamente entre lectores. El semáforo sem_buffer controla el acceso al buffer propiamente dicho. El semáforo sem_cant_lect es usado como contador de la cantidad de lectores que están accediendo simultáneamente al buffer, cuando este contador es cero se habilita el acceso a escritores. Por ultimo el semáforo sem_exlucion es utilizado para obtener exclusión mutua para acceder al contador cant_lectores, ya que luego de incrementarla o decrementarla se debe leer su valor para decidir qué acción realizar, y si entre éstas dos operaciones se modificara el valor del contador, el resultado sería imprevisible.

La desición de utilizar un semaforo para contabilizar la cantidad de lectores que acceden simultaneamente al buffer en vez de utilizar una variable, se tomo porque en Minix se realiza una copia de los datos compartidos al hijo, si éste último modifica algún valor solo será visible por el mismo(y sus hijos), por lo que ni su padre ni sus "hermanos" verán el cambio realizado. Otra posibilidad para superar éste obstáculo era usar pipes, pero preferimos por simplicidad utilizar un semáforo como contador. Se usara semV(sem_cant_lect) para incrementar el contador y semP(sem_cant_lect) para decrementarlo.

Para poder realizar un seguimiento de lo que sucede con cada proceso cuando interactuan mediante los semaforos, imprimimos por pantalla cada acción relevante realizada (lecturas y escrituras en el buffer, bloqueo del buffer para escritura, etc...) indentificandolas con el id del proceso que realiza cada accion.

A continuación presentamos un pseudocódigo de nuestra implementación. En el mismo se puede ver el manejo de los tres semaforos antes de que un proceso lector o escritor acceda al buffer.

```
lector(){
  p(sem_exclusion)  /*exclusion mutua para acceder a sem_cant_lect*/
  p(sem_cant_lect)  /*se incrementa la cantidad de lectores*/
  si sem_cant_lect=1
   entonces p(sem_buffer)  /*se cierra el buffer para los escritores*/
  v(sem_exclusion)

  leer_buffer()

  p(sem_exclusion)  /*exclusion mutua para acceder a sem_cant_lectores*/
  P(sem_cant_lect)  /*se decrementa la cantidad de lectores*/
  si sem_cant_lect=0
   entonces p(sem_buffer)  /*se abre el buffer para los escritores*/
  v(exclusion)
}

escritor(){
  preparar_datos()
  p(sem_buffer)
  escribir_buffer()
  v(sem_buffer)
}
```

## 13.3. Pruebas

Para realizar las pruebas nos encontramos algunos obstáculos, sobre todo para obtener casos de concurrencia de varios lectores, y a la vez hacer posible que éstos den lugar a entrar a los escritores(no todos juntos), de modo que no trabajen primero todos los lectores y luego todos los escritores(o viceversa).

Para lograr esto, se crearon por un lado retardos(*delays*) tanto para la lectura de buffer y como para la escritura, y por otro lado se definieron dos constantes para controlar la tasa de arribos de lectores y de escritores. Adicionalmente se agregaron dos constantes para definir la cantidad de lectores y la cantidad de escritores que intervienen en la prueba.

1. `CANT_LECTORES`: cantidad total de lectores
2. `CANT_ESCRITORES`: cantidad total de escritores

3. `DELAY_ENTRE_ESCRITORES`: tiempo de espera entre el ingreso de un escritor y el siguiente
4. `DELAY_ENTRE_LECTORES`: tiempo de espera entre el ingreso de un lector y el siguiente
5. `DELAY_ESCRIBIR`: tiempo que le toma a un escritor modificar el buffer
6. `DELAY_LEER`: tiempo que le toma a un lector leer el buffer

Haciendo pruebas con diferentes valores para las variables anteriores, se pudo lograr que los escritores queden repartidos entre los lectores.

A continuación mostramos la salida (con las líneas numeradas) de una de las pruebas realizadas en la que intervienen seis lectores y tres escritores.

```
 1- El proceso 37 escribe el dato '37' en el buffer.
 2- El proceso 39 intenta escribir. Cantidad de lectores: 0.
 3- El proceso 39 escribe el dato '39' en el buffer.
 4- El proceso 40 intenta leer. Cantidad de lectores: 1.
 5- El proceso 40 bloquea el buffer para escritura.
 6- El proceso 40 lee el dato '39' del buffer.
 7- El proceso 41 intenta leer. Cantidad de lectores: 2.
 8- El proceso 41 lee el dato '39' del buffer.
 9- El proceso 42 intenta escribir. Cantidad de lectores: 1.
10- El proceso 43 intenta leer. Cantidad de lectores: 2.
11- El proceso 43 lee el dato '39' del buffer.
12- El proceso 44 intenta leer. Cantidad de lectores: 2.
13- El proceso 44 lee el dato '39' del buffer.
14- El proceso 45 intenta escribir. Cantidad de lectores: 1.
15- El proceso 46 intenta leer. Cantidad de lectores: 2.
16- El proceso 46 lee el dato '39' del buffer.
17- El proceso 47 intenta leer. Cantidad de lectores: 2.
18- El proceso 47 lee el dato '39' del buffer.
19- El proceso 47 desbloquea el buffer para escritura.
20- El proceso 42 escribe el dato '42' en el buffer.
21- El proceso 45 escribe el dato '45' en el buffer.
```

En este ejemplo se ven claramente varios aspectos esperados de esta implementación. Por un lado, vemos que se da la concurrencia entre varios lectores al buffer. Por ejemplo, en las líneas 7 y 8 se ve el acceso del lector 41 al buffer aún cuando el lector 40 se

encuentra leyendo del buffer. Por otro lado, ve puede ver el bloqueo de la escritura en el buffer cuando un lector esta accediéndolo. Por ejemplo, en la línea 14 el escritor 45 intenta escribir cuando hay un lector accediendo al buffer, por lo que queda bloqueado hasta que el buffer es liberado.

La prueba anterior también muestra otro aspecto muy importante, que es el hecho de que los lectores tienen prioridad sobre los escritores. Si observamos la linea 9 el escritor 42 intenta escribir en el buffer pero queda bloqueado hasta la linea 20, ya que hay un lector accediendo al buffer. Sin embargo, desde que este escritor intenta acceder al buffer hasta que finalmente escribe en él, los lectores 43, 44, 46 y 47 acceden al buffer aun cuando su intento de acceso fue posterior al del escritor encolado.

## 13.4. Código

A continuación presentamos el código de nuestra implementación del problema de Lectores/Escritores, junto con la prueba citada en la sección anterior.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <semaforo.h>
#include <lib.h>

/*DELAYS*/
#define DELAY_ENTRE_ESCRITORES 10000000
#define DELAY_ENTRE_LECTORES 10000000
#define DELAY_ESCRIBIR 1000000
#define DELAY_LEER 10000000

const int CANT_LECTORES = 6;
const int CANT_ESCRITORES = 3;
const char ARCHIVO[] = "buffer";

char sem_buffer[M3_STRING]="sem_buffer";
char sem_exclusion[M3_STRING]="sem_exclusion";
```

```
char sem_cant_lect[M3_STRING]="sem_cant_lect";

void leer_buffer(void);
void escribir_buffer(void);
void delay(int valor);

void lector(void)
{
semP(sem_exclusion);/* entro en region critica */
semV(sem_cant_lect);/* incremento la cantidad de lectores */
fflush(0);

/* el primer lector cierra el buffer*/
if(semget(sem_cant_lect) == 1)
  {
semP(sem_buffer);
fflush(0);
}

  semV(sem_exclusion);/* salgo de region critica */

  leer_buffer();

  semP(sem_exclusion); /* entro a una nueva region critica */
semP(sem_cant_lect); /*un lector menos,nunca sera negativo*/

/* si no hay mas lectores abro el buffer */
if(semget(sem_cant_lect) <= 0)
  {
fflush(0);
semV(sem_buffer);
}

  semV(sem_exclusion);
}

void escritor(void)
{
  fflush(0);
```

```
  semP(sem_buffer);

  escribir_buffer();

  semV(sem_buffer);
}

int crear_escritores(void)
{
int i=0;
  semset(sem_buffer ,1);
emset(sem_exclusion ,1);
semset(sem_cant_lect ,0);

for(i=0 ; i < CANT_ESCRITORES ; i++)
  {
   if(fork()==0)
    {
     semasing(sem_buffer);
semasing(sem_exclusion);
  semasing(sem_cant_lect);

  escritor();

      semdasing(sem_buffer);
  semdasing(sem_exclusion);
  semdasing(sem_cant_lect);

      exit(0);
    }
    else
    {
       delay(DELAY_ENTRE_ESCRITORES);
    }
}
for(i=0; i< CANT_ESCRITORES; i++) wait(0);
return(0);
}

int darTiempoParaEscribir(int nro_lector)
```

```
{
int cantLectoresPorEscritor = CANT_LECTORES / CANT_ESCRITORES;
while(nro_lector>0)
{
nro_lector = nro_lector - cantLectoresPorEscritor;
}
return nro_lector==0;
}

int crear_lectores(void)
{
int i=0;
for(i=0; i< CANT_LECTORES; i++)
{
if(fork()==0)
{
semasing(sem_buffer);
semasing(sem_exclusion);
semasing(sem_cant_lect);

lector();

semdasing(sem_buffer);
semdasing(sem_exclusion);
semdasing(sem_cant_lect);

exit(0);
}
else
{
delay(DELAY_ENTRE_LECTORES);
}
}
for(i=0; i< CANT_LECTORES; i++) wait(0);
return 0;
}

int main(int argc, char* argv[])
{
int i = 0;
```

```
open(ARCHIVO, O_CREAT);
escribir_buffer();

semcreat(sem_buffer)
semcreat(sem_cant_lect)
semcreat(sem_exclusion)

if (fork() == 0)
{
crear_Lectores();
exit(0);
}
else
{
crear_escritores();
  exit(0);
}
  wait(0);
return 0;
}

void escribir_buffer(void)
{
int fd = open(ARCHIVO, O_WRONLY);
int pid = getpid();
write(fd, &pid, 3);
close(fd);
fflush(0);
delay(DELAY_ESCRIBIR);
}

void leer_buffer(void)
{
int fd = open(ARCHIVO, O_RDONLY);
int leido;
read(fd, &leido, 3);
close(fd);
fflush(0);
delay(DELAY_LEER);
}
```

```
void delay(int valor)
{
while(valor-- > 0);
}
```

## 14. Filósofos Chinos

Resuelva el problema de los Filósofos Chinos con la herramienta adecuada. (Tests o pruebas mencionados en Forma de entrega).

**Rta.:**

Decidimos implementar una solución al problema de los filósofos chinos mediante semáforos. La solución tradicional, es decir, aquella en la que todos los filósofos tienen el mismo comportamiento y toman los palitos en el mismo órden puede generar inanición, dado que si todos toman el palito a su izquierda antes de que cualquiera pueda tomar el palito a su derecha todos se quedarán esperando a que otro termine y eso nunca ocurrirá.

Para solucionar este problema proponemos lo siguiente:
el principal problema aquí es que puede ocurrir una espera circular. Si conseguimos evitar este tipo de espera el problema comentado en el párrafo anterior desaparece. Una forma de hacerlo es haciendo que un filósofo se comporte de forma distinta al resto, en el sentido de que intenta tomar los palitos en el orden inverso al resto de los filósofos.

En nuestra implementación los filósofos se numeran del 0 al 4, al igual que los palitos, siendo el *i-ésimo* palito el que se encuentra a la izquierda del filósofo $i$ y el $(i+1)$-ésimo palito el que se encuentra a su derecha (en el caso del filósofo 0 el de su izquierda es el palito 4, y en el caso del filósofo 4 el palito a su derecha es el 0). Los semaforos se identifican con letras desde la A en adelande, segun se necesite. Se asignan el orden al $i$-esimo palillo.

El código de nuestra resolución del problema de los filósofos chinos es el siguiente (omitimos los `printf` que muestran el proceso paso a paso para mayor claridad):

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <lib.h>
#include <semaforo.h>


#define NUM_FILOSOFOS 5
#define NUM_PLATOS_DE_ARROZ_POR_FILOSOFO 1
char* sem_names[NUM_FILOSOFOS]={"A","B","C","D","E"}; /* Semaforos para los palitos */
```

```
void filosofoNoUltimo(int i);
void filosofoUltimo();
void piensa();
void come();

void filosofoNoUltimo(int i)
{
/* Cantidad de veces que el filosofo va a comer */
int contador = NUM_PLATOS_DE_ARROZ_POR_FILOSOFO;
/* Al iniciar el proceso adquirimos el derecho a utilizar los */
/* semaforos creados e inicializados por el proceso padre */

semasign(sem_names[i]);
semasign(sem_names[i+1]);

while (contador > 0)
{
piensa(i);
/* Tomamos primero el palito de la izquierda y luego el */
/* de la derecha */
semP(sem_names[i]);
semP(sem_names[i+1]);

come(i);

semV(sem_names[i + 1]);
semV(sem_names[i]);

contador--;
  }

semdasign(sem_names[i+1]);
semdasign(sem_names[i]);

}

void filosofoUltimo()
{
/* Cantidad de veces que el filosofo va a comer */
int contador = NUM_PLATOS_DE_ARROZ_POR_FILOSOFO;
```

```
/* Al iniciar el proceso adquirimos el derecho a utilizar los */
/* semaforos creados e inicializados por el proceso padre */
semasign(sem_names[0]);
semasign(sem_names[NUM_FILOSOFOS - 1]);

while (contador > 0)
{
piensa(NUM_FILOSOFOS - 1);
/* Tomamos primero el palito de la derecha y luego el */
/* de la izquierda */
semP(sem_names[0]);
semP(sem_names[NUM_FILOSOFOS - 1]);

come(NUM_FILOSOFOS - 1);
/* Liberamos los palitos en el orden inverso */
semV(sem_names[NUM_FILOSOFOS - 1]);
semV(sem_names[0]);
contador--;
}

semdasign(sem_names[NUM_FILOSOFOS - 1]);
semdasign(sem_names[0]);

}

void piensa(int i)
{
printf("* %i esta pensando.\n", i);
fflush(0);
/* Aca iria un algoritmo de pensamiento */
}

void come(int i)
{
/* Delay, o cantidad de arroces por plato :P */
int contador = 1000000;
printf("* %i esta comiendo.\n", i);
fflush(0);
/* Come los arroces */
while (contador > 0)
```

```
contador--;
}

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

int i;
/* Creamos tantos semaforos como filosofos haya en la mesa (palitos) */
for (i = 0; i < NUM_FILOSOFOS; i++){
semcreat(sem_names[i]);
semset(sem_names[i], 1);
}
/* Creamos los primeros N - 1 filosofos que tienen similar */
/* comportamiento */
for (i = 0; i < NUM_FILOSOFOS - 1; i++)
{
if (fork() == 0)
{
filosofoNoUltimo(i);
exit(0);
}
}
/* Creamos el ultimo filosofo, el que tiene comportamiento distinto */
/* a los demas */
if (fork() == 0)
{
filosofoUltimo();
exit(0);
}
return 0;
}
```

El test analiza el funcionamiento de la implementación aplicando una espera en el procedimiento `come` e indicando a cada filósofo que coma 50 veces (valor arbitrario para dar fin al programa). Una muestra limitada de la salida obtenida es:

```
1.  * 4 esta pensando.
2.  - 4 desea tomar el palito 0 (derecha)
3.  - 4 tomo el palito 0 (derecha)
4.  - 4 desea tomar el palito 4 (izquierda)
5.  - 4 tomo el palito 4 (izquierda)
```

```
6.  * 4 esta comiendo.
7.  * 3 esta pensando.
8.  - 3 desea tomar el palito 3 (izquierda)
9.  - 3 tomo el palito 3 (izquierda)
10. - 3 desea tomar el palito 4 (derecha)
11. * 2 esta pensando.
12. - 2 desea tomar el palito 2 (izquierda)
13. - 2 tomo el palito 2 (izquierda)
14. - 2 desea tomar el palito 3 (derecha)
15. * 1 esta pensando.
16. - 1 desea tomar el palito 1 (izquierda)
17. - 1 tomo el palito 1 (izquierda)
18. - 1 desea tomar el palito 2 (derecha)
19. * 0 esta pensando.
20. - 0 desea tomar el palito 0 (izquierda)
21. - 4 libero el palito 4 (izquierda)
22. - 4 libero el palito 0 (derecha)
23. * 4 esta pensando.
24. - 4 desea tomar el palito 0 (derecha)
25. - 3 tomo el palito 4 (derecha)
26. * 3 esta comiendo.
27. - 3 libero el palito 4 (derecha)
28. - 3 libero el palito 3 (izquierda)
```

En síntesis, ocurre lo siguiente:

1-6 : El filósofo 4 toma los palitos 0 y 4 (en ese orden) y se pone a comer.

7-10 : El filósofo 3 quiere comer, toma el palito 3 pero al querer tomar el palito 4 debe esperar.

11-14 : El filósofo 2 quiere comer, toma el palito 2 pero al querer tomar el palito 3 debe esperar.

15-18 : El filósofo 1 quiere comer, toma el palito 1 pero al querer tomar el palito 2 debe esperar.

19-20 : El filósofo 0 quiere comer, pero no puede tomar el palito a su izquierda (el 0) porque lo tiene el filósofo 4. Espera.

21-22 : El filósofo 4 libera los palitos 0 y 4

23-24 : El filósofo 4 quiere volver a comer, pero no llega a tomar el palito 4 nuevamente.

25-28 : El filósofo 3, que estaba esperando por la liberación del palito 4, toma el palito recién liberado, come y devuelve los palitos a la mesa.

# 15. Aplicación que muestra un inodo

Genere una aplicación que muestre el contenido del inodo de un archivo cuyo `/$HOME/nombre` se pasa como parámetro.

**Rta.:**

## 15.1. Resumen de lo implementado

Resolvimos el problema con un programa que denominamos "inode" hecho íntegramente en código C. El mismo recibe como único parámetro el nombre del archivo sobre el cual deseamos mostrar información por pantalla. Cabe recordar que, debido a que en Minix **todo** es un archivo, este programa aceptará cualquier entrada que se pueda ver con el comando `ls`, más precisamente, aceptará como parámetro:

- archivos;
- directorios;
- dispositivos de bloque;
- dispositivos de caracteres;
- pipes;
- sockets;
- etc...

El programa muestra los siguientes datos:

- El tipo de archivo (si es un *file*, un directorio, dispositivo, etc.);
- los permisos asociados a él (en el mismo formato que se ve en un `ls -l`);
- el dispositivo dónde reside físicamente el directorio (si está en un disco rígido, un diskette, CD-ROM, etc...);
- el número del inodo;
- la cantidad de *hard links* que posee el inodo;
- el *userId*;
- el *groupId*;
- el tamaño del archivo;
- la fecha de último acceso;
- la fecha de modificación de datos y;
- la fecha de modificación del inodo del archivo en cuestión.

## 15.2. Implementación

Recurrimos al libro de Tanenbaum [1] e investigando descubrimos que existe un *system call* que nos era útil para la finalidad de nuestro programa. Dicho *system call* se llama `stat` y posee la siguiente sintáxis:

```
int stat(const char *path, struct stat *buf)
```

dónde `path` es la ruta (absoluta o relativa) del archivo que deseo revisar y;
`buf` es un puntero a una estructura predefinida por Minix dónde el *system call* deja el resultado de la exploración del archivo. Dicho `struct` se denomina `stat`, y su estructura interna es la siguiente:

```
struct stat {
    dev_t   st_dev;   /* dispositivo donde el inodo reside */
    ino_t   st_ino;   /* el numero de este inodo */
    mode_t  st_mode;  /* modo del archivo, bits de proteccion, etc. */
    nlink_t st_nlink; /* numero de hard links del archivo */
    uid_t   st_uid;   /* user-id del owner del archivo */
    gid_t   st_gid;   /* group-id del owner del archivo */
    dev_t   st_rdev;  /* el tipo de dispositivo, si es dispositivo */
    off_t   st_size;  /* el tamanio del archivo */
    time_t  st_atime; /* hora de ultimo acceso */
    time_t  st_mtime; /* hora de la ultima modificacion de datos */
    time_t  st_ctime; /* hora de la ultima modificacion del inodo */
};
```

Dadas estas herramientas, lo demas es sólo codificar la salida por pantalla de los datos, considerando la estructura interna de aquellos datos que están adentro del `struct` para poder utilizar correctamente la función `printf`. Pero todavía quedan algunos problemas a resolver:

- cómo mostrar correctamente las tres horas de tipo `time_t` (un renombre de `long`).
- cómo mostrar los permisos de archivo para *owner*, *group* y *other*, siendo que estos datos están "ocultos" en la variable `st_mode`.

Para resolver el primer problema, utilizando el `man` de Minix, descubrimos la función `ctime()` definida en `time.h`. La misma toma como parametro un puntero a `time_t` que es justamente lo que tenemos.

Para el segundo problema, tuvimos que investigar más y descubrir que existían máscaras predefinidas para poder aislar los permisos específicos. Dichas máscaras se

encuentran en `stat.h` y son las siguientes:

```
/* POSIX masks for st_mode. */
#define S_IRWXU   00700         /* owner:  rwx------ */
#define S_IRUSR   00400         /* owner:  r-------- */
#define S_IWUSR   00200         /* owner:  -w------- */
#define S_IXUSR   00100         /* owner:  --x------ */
#define S_IRWXG   00070         /* group:  ---rwx--- */
#define S_IRGRP   00040         /* group:  ---r----- */
#define S_IWGRP   00020         /* group:  ----w---- */
#define S_IXGRP   00010         /* group:  -----x--- */
#define S_IRWXO   00007         /* others: ------rwx */
#define S_IROTH   00004         /* others: ------r-- */
#define S_IWOTH   00002         /* others: -------w- */
#define S_IXOTH   00001         /* others: --------x */
```

En base a estas máscaras, efectuamos una serie de `#define`'s que luego se utilizarán para separar los permisos de *owner*, *group*, *other*. Esta idea de utilizar *defines* salieron de una porción de código de `stat.h` que también utilizamos para dirimir entre archivos, directorios, etc. y que mostraremos a continuación:

```
/* The following macros test st_mode (from POSIX Sec. 5.6.1.1). */
#define S_ISREG(m)   (((m) & S_IFMT) == S_IFREG)  /* is a reg file */
#define S_ISDIR(m)   (((m) & S_IFMT) == S_IFDIR)  /* is a directory */
#define S_ISCHR(m)   (((m) & S_IFMT) == S_IFCHR)  /* is a char spec */
#define S_ISBLK(m)   (((m) & S_IFMT) == S_IFBLK)  /* is a block spec */
#define S_ISFIFO(m)  (((m) & S_IFMT) == S_IFIFO)  /* is a pipe/FIFO */
```

Por último, mostraremos el código de nuestro programa, a saber:

```
/* inode.c ---- Un programa que muestra por pantalla la informacion
 * almacenada en un inodo, ya sea que este apunta a un archivo,
 * a un dispositivo de caracteres, de bloques, directorio u otro.
 * Alumnos: Valdez - Di Vincenzo
 */

#include <sys/types.h>
#include <sys/stat.h>/* strut stat */
#include <stdio.h>/* printf() */
#include <time.h>/* ctime()*/
#include <stdlib.h>/* exit() y system()*/
```

```
/* Mascaras para filtrar permisos de USER */
#define S_ISRWXU(m) (((m) & S_IRWXU) == S_IRWXU)
#define S_ISRUSR(m) (((m) & S_IRUSR) == S_IRUSR)
#define S_ISWUSR(m) (((m) & S_IWUSR) == S_IWUSR)
#define S_ISXUSR(m) (((m) & S_IXUSR) == S_IXUSR)

/* Mascaras para filtrar permisos de GROUP */
#define S_ISRWXG(m) (((m) & S_IRWXG) == S_IRWXG)
#define S_ISRGRP(m) (((m) & S_IRGRP) == S_IRGRP)
#define S_ISWGRP(m) (((m) & S_IWGRP) == S_IWGRP)
#define S_ISXGRP(m) (((m) & S_IXGRP) == S_IXGRP)

/* Mascaras para filtrar permisos de OTHER */
#define S_ISRWXO(m) (((m) & S_IRWXO) == S_IRWXO)
#define S_ISROTH(m) (((m) & S_IROTH) == S_IROTH)
#define S_ISWOTH(m) (((m) & S_IWOTH) == S_IWOTH)
#define S_ISXOTH(m) (((m) & S_IXOTH) == S_IXOTH)


/* Declaracion de funciones */
void showDevice(dev_t data);
void showINodeNumber(ino_t data);
void showModeWord(mode_t data, dev_t deviceType);
void showHardLinks(nlink_t data);
void showUid(uid_t data);
void showGid(gid_t data);
void showSize(off_t data);
void showLastAccess(time_t data);
void showLastDataModification(time_t data);
void showLastInodeModification(time_t data);
void showOwnerPermissionsWith(mode_t data);
void showGroupPermissionsWith(mode_t data);
void showOtherPermissionsWith(mode_t data);
void showInodeTypeWith(mode_t data, dev_t deviceType);
void showInode(struct stat *buffer);


/* Entry point del programa */
int main(int argc, char *argv[]){
```

```
  struct stat buffer;
  if (stat(argv[1],&buffer) == -1){
    perror(argv[0]);
    exit(-1);
  }
  else{
    system("clr");
    printf("Sumario del archivo: %s\n", argv[1]);
    printf("------------------------------------------------\n");
    showInode(&buffer);
  }
  return 0;
}


/* Muestra la estructura apuntada en 'buffer' */
void showInode(struct stat *buffer){
  showModeWord(buffer->st_mode, buffer->st_rdev);
  showDevice(buffer->st_dev);
  showINodeNumber(buffer->st_ino);
  showHardLinks(buffer->st_nlink);
  showUid(buffer->st_uid);
  showGid(buffer->st_gid);
  showSize(buffer->st_size);
  showLastAccess(buffer->st_atime);
  showLastDataModification(buffer->st_mtime);
  showLastInodeModification(buffer->st_ctime);
}


/* Muestra todos los datos relacionados con el modo del inodo */
void showModeWord(mode_t mode, dev_t deviceType){
  showInodeTypeWith(mode, deviceType);
  printf("* Permisos:\n");
  showOwnerPermissionsWith(mode);
  showGroupPermissionsWith(mode);
  showOtherPermissionsWith(mode);

  printf("* La palabra de modo es: %u\n", mode);
}
```

```
/* Muestra de que tipo es el inodo */
void showInodeTypeWith(mode_t mode, dev_t deviceType ){
  if(S_ISDIR(mode))
    printf("* El inodo apunta a un directorio\n");
  else if(S_ISCHR(mode))
    printf("* El inodo apunta al dispositivo de caracteres %u\n",
                                                        deviceType);
  else if(S_ISBLK(mode))
    printf("* El inodo apunta al dispositivo de bloques %u\n",
                                                        deviceType);
  else if(S_ISFIFO(mode))
    printf("* El inodo apunta a un pipe\n");
}


/* Muestra los permisos para el owner del archivo */
void showOwnerPermissionsWith(mode_t mode){
  printf("OWNER: ");
  if(S_ISRWXU(mode)) printf("rwx\n");
  else{
      if(S_ISRUSR(mode)) printf("r"); else printf("-");
      if(S_ISWUSR(mode)) printf("w"); else printf("-");
      if(S_ISXUSR(mode)) printf("x\n"); else printf("-\n");
}
}


/* Muestra los permisos para el grupo del archivo */
void showGroupPermissionsWith(mode_t mode){
  printf("GROUP: ");
  if(S_ISRWXG(mode)) printf("rwx\n");
  else{
      if(S_ISRGRP(mode)) printf("r"); else printf("-");
      if(S_ISWGRP(mode)) printf("w"); else printf("-");
      if(S_ISXGRP(mode)) printf("x\n"); else printf("-\n");
  }
}
```

```
/* Muestra los permisos para aquellos usuarios que
no son ni owner ni del grupo del owner */
void showOtherPermissionsWith(mode_t mode){
  printf("OTHER: ");
  if(S_ISRWXO(mode)) printf("rwx\n");
  else{
      if(S_ISROTH(mode)) printf("r"); else printf("-");
      if(S_ISWOTH(mode)) printf("w"); else printf("-");
      if(S_ISXOTH(mode)) printf("x\n"); else printf("-\n");
  }
}


/* Muestra el ID del dispositivo donde reside el inodo
(ej= diskette, cd-rom, rigido, etc...)*/
void showDevice(dev_t residentDevice){
  printf("* El inodo reside fisicamente en el dispositivo: %i\n",
    residentDevice);
}


/* Muestra el numero del inodo */
void showINodeNumber(ino_t inodeNumber){
  printf("* El numero del inodo es: %u\n", inodeNumber);
}


/*Muestra la cantidad de hard links asociados al inodo */
void showHardLinks(nlink_t number){
  if(number == 1)
    printf("* El archivo posee 1 hard link\n");
  else
    printf("* El archivo posee %u hard links\n", number);
}


/* Muestra el UserId del inodo */
void showUid(uid_t uid){
  printf("* El userId del archivo es: %u\n", uid);
```

```
}


/* Muestra el GroupId del inodo */
void showGid(gid_t gid){
  printf("* El groupId del archivo es: %u\n", gid);
}

/* Muestra el tamanio en bytes del archivo */
void showSize(off_t size){
  printf("* El tamano del archivo es de %u bytes\n", size);
}


/* Muestra la hora de ultimo acceso al archivo */
void showLastAccess(time_t time){
  printf("* La hora de ultimo acceso es: %s", ctime(&time));
}


/* Muestra la hora de ultima modificacion de datos */
void showLastDataModification(time_t time){
  printf("* La hora de la ultima modificacion de datos es: %s",
                                                ctime(&time));
}


/* Muestra la hora de ultima modificacion del inodo */
void showLastInodeModification(time_t time){
  printf("* La hora de la ultima modificacion del inodo es: %s",
                                                    ctime(&time));
}
```

Los fuentes de este programa se pueden encontrar en la imagen entregada en CD, en el directorio `/usr/srcTpFinal/inode/inode.c`. El ejecutable `/usr/srcTpFinal/inode/inode`, también se encuentra ya compilado en el mismo directorio; y sólo hay que correrlo pasandole como parametro el nombre de archivo (ruta absoluta o relativa) con o sin comillas.

# A. Ubicación y breve descripción de las fuentes y pruebas

- `/tests/lectores/lectores.out` - Resultados del test de Lectores-Escritores
- `/usr/srcTpFinal/lectores/lectores.c`: - Implementación de la solución al problema con test e información del proceso paso a paso.

- `/tests/filosofos/filosofos.out` - Resultados del test del problema de los filósofos chinos
- `/usr/srcTpFinal/filosofos/filosofos.c`: Implementación de la solución al problema con test e información del proceso paso a paso.

- `/tests/inode/barra.txt`: Salida del programa cuando ingresamos como parámetro el directorio raíz (`/`). Este constituye un ejemplo de cómo muestra el programa el inodo de un directorio.
- `/tests/inode/devnull.txt`: Salida del programa, mostrando el contenido de un inodo que no apunta a ningún bloque de disco sino que apunta a un dispositivo de bloques como es `/dev/null`.
- `/tests/inode/inode.txt`: Salida del archivo mismo que estoy ejecutando.
- `/usr/srcTpFinal/inode/inode.c`: Implementación del programa que muestra un inodo por pantalla.

# Referencias

[1] Sistemas Operativos. Diseño e Implementación. A. Tanenbaum-A. Woodhull. Segunda Edición (Prentice Hall)

[2] Semáforos y memoria compartida en UNIX. Página de Internet. `http://www.infor.uva.es/~isaac/SO/`

[3] Smx-the Solaris port of MINIX - Paul Ashton, paper disponible en `http://www.infor.uva.es/~benja/smx-html/`

[4] Implementando semáforos. URL: `http://www.cs.lafayette.edu/~pfaffmaj/classes/F04/cs406/`

[5] Operating Systems-EricFreudenthal, cursos, material disponible en: `http://rlab.cs.utep.edu/~freudent/courses/osLectureNotes/`

[6] Printf.3 Linux Man Page, disponible en: `http://maconlinux.net/linux-man-pages/es/printf.3.html`

[7] Llamadas al sistema. Uso de Stat. Presentación de la Universidad Politécnica de Madrid E.U. de Informática. Sistemas Operativos II.