Arquitectura de Sistemas Operativos

Módulo II	**Comprender como el Sistema Operativo realiza la administración de recursos del Sistema de Computación**
Unidad 6	**Gestión de archivos y directorios.**
	Trabajo práctico requerido Nº 13: **Codificación de servicios de Sistemas de Archivos**

Presentación

Este trabajo requiere que el alumno utilice un sistema operativo Linux en cualquiera de sus distribuciones y versiones.
Dicho sistema operativo deberá tener instalado un editor de texto y el compilador del Lenguaje de Programación C, comúnmente llamado gcc o cc.

El objetivo de esta práctica es el estudio de las llamadas al sistema básicas para la gestión de archivos en el Sistema Operativo LINUX, y pretende otorgarle una visión real de los conceptos estudiados en la teoría hasta el momento.
La práctica se basa en la presentación de ejemplos que usted debe implementar en Sistema Operativo LINUX y analizar los resultados que se obtienen luego de ejecutar dichos procesos y en la resolución de ejercicios similares a los ejemplos propuestos.

Este trabajo intenta favorecerle el acceso a las siguientes metas de aprendizaje:

- Aplicar los conceptos adquiridos en la implementación de programas que representan la creación, la escritura, la lectura de archivos y el acceso a la información de los directorios.

- Poner manos a la obra en el Sistema Operativo LINUX, como ya sabemos para aprender a andar en bicicleta hay que inflarle las ruedas, subirse y caerese algunas veces, luego en un momento dado salimos andando sin darnos cuenta.

- Prepararnos para entender las herramientas utilizadas en los servicios que brinda el Sistema Operativo para la gestión de información en los dispositivos de almacenamiento magnético.

- Comenzar a ser operadores y programadores en un Sistema Operativo.

Consignas

Ejercicio 1

System call open() y close()
Ejemplo del system call open() utilizado por el sistema operativo para abrir un archivo en modo lectura y el system call close() para cerrar el archivo.

El alumno debe ejecutar este programa desde la línea de comandos pasando como argumento el nombre del archivo a utilizar.

```
# include <fcntl.h>
main(int argc, char* argv[])
{
    int fd;
    if((fd = open(argv[1], O_RDONLY)) == -1 )
      {
         printf("No puede abrir archivo %s",argv[1]);
         exit(1);
      }
    printf("Archivo %s fue abierto exitosamente",argv[1]);
    printf("Su descriptor de archivo fue %d",fd);
    close(fd);
    exit(0);

 }
```

a) Generar un programa ejecutable.
b) Verificar los resultados obtenidos luego de la ejecución del programa.

Ejercicio 2

System call creat()
Ejemplo del system call craet() utilizado por el sistema operativo para crear un archivo .

El alumno debe ejecutar este programa desde la línea de comandos pasando como argumento el nombre del archivo a utilizar.

```
#define PERM 0644
main(int argc, char* argv[])
{
    int fd;
    if((fd = creat(argv[1], PERM)) == -1 )
      {
         printf("No puede crear archivo %s",argv[1]);
         exit(1);
      }
        printf("Archivo %s creado exitosamente",argv[1]);
        printf("Verifique la existencia del archivo con ls -l");
        printf("En el momento de observar los permisos del archivo tenga en
cuenta el umask");
        close(fd);
        exit(0);
}
```

a) Generar un programa ejecutable.
b) Verificar los resultados obtenidos luego de la ejecución del programa.

Ejercicio 3

System Call read()

El alumno debe ejecutar este programa desde la línea de comandos pasando como argumento el nombre del archivo a utilizar.

Se recomienda que el alumno modifique el valor de BUFSIZE y analice los resultados obtenidos.

```
#include <fcntl.h>
#define BUFSIZE 512
main(int argc, char* argv[])
{
     char buffer[BUFSIZE];
     int fd;
     if((fd = open(argv[1], O_RDONLY)) < 0 )
       {
          printf("No puede abrir archivo %s ",argv[1]);
          exit(1);
       }
while ((read(fd, buffer, BUFSIZE)) > 0) printf("%s ",buffer);
close(fd);
exit(0);
}
```

a) Generar un programa ejecutable.
b) Verificar los resultados obtenidos luego de la ejecución del programa.

Ejercicio 4

System Call write()

El alumno debe ejecutar este programa desde la línea de comandos pasando como argumento el nombre del archivo a utilizar.

```
#include <stdio.h>
#define PERM 0644
#define BUFSIZE 512
main(int argc, char* argv[])
{
      int fd , n ;
      char buf[BUFSIZE];
      if((fd = creat(argv[1], PERM)) == -1 )
         {
               printf("No puede crear archivo %s",argv[1]);
               exit(1);
         }
printf("Escriba lo que quiera guardar en el archivo %s ",argv[1]);
printf("Para terminar oprime CTRL-D ");
while ((n = read(0,buf, BUFSIZE)) > 0 ) write(fd,buf,n);
printf("Archivo %s creado exitosamente",argv[1]);
printf("Verifique el contenido del archivo");
close(fd);
exit(0);
}
```

a) Generar un programa ejecutable.
b) Verificar los resultados obtenidos luego de la ejecución del programa.

Ejercicio 5

System Call lseek()

El alumno debe ejecutar este programa desde la línea de comandos pasando como argumento el nombre del archivo a utilizar.

```
#include <stdio.h>
#include <ctype.h>
#include <fcntl.h>
#define BUFSIZE 512
main(int argc, char* argv[])
{
   int fd , posicion;
   char buf[BUFSIZE];
   if((fd = open(argv[1], O_RDONLY)) == -1 )
    {
          printf("No puede abrir archivo %s",argv[1]);
          exit(1);
      }
printf("Lectura desde el byte %s en el archivo %s ",argv[2],argv[1]);
posicion = atoi(argv[2]);
lseek(fd,posicion,1);
read(fd,buf,BUFSIZE);
printf("%s",buf);
close(fd);
exit(0);
}
```

a) Generar un programa ejecutable.
b) Verificar los resultados obtenidos luego de la ejecución del programa.

Ejercicio 6

System call link()

Ejemplo del system call link(), utilizado por el sistema operativo para realizar un vínculo entre dos archivos donde sólo en uno existen los datos físicamente. Tener en cuenta que dicho system call crea un vínculo físico, es decir existen en el directorio una entrada por cada vinculación pero ellas contienen el mismo identificador del inodo.

El alumno debe ejecutar este programa desde la línea de comandos pasando dos argumentos: el primero debe ser el nombre del archivo que contiene los datos físicos y el segundo es el archivo a vincular.

```
#include <fcntl.h>
main(int argc, char* argv[])
{
      if(link(argv[1],argv[2]) == -1)
         {
            printf("No puede Realizar el link");
            exit(1);
         }
printf("Observar el resultado con el comando ls -li ");
printf("Qué, relación existe entre los números de los inodos ");
exit(0);
}
```

a) Generar un programa ejecutable.
b) Verificar los resultados obtenidos luego de la ejecución del programa.

Ejercicio 7

System Call unlink()

Ejemplo del system call unlink() utilizado por el sistema operativo para eliminar un vínculo generado con el system call link().

Hay que tener en cuenta que el system call unlink elimina una vinculación física es decir que se elimina la entrada en el directorio correspondiente al argumento del unlink.

El alumno debe ejecutar este programa desde la línea de comandos pasando como argumento el nombre del archivo vinculado .

```
#include <fcntl.h>
main(int argc, char* argv[])
{
      if(unlink(argv[1]) == -1)
            {
               printf("No puede Realizar unlink");
               exit(1);
            }
printf("Observar el resultado con el comando ls -li");
exit(0);
}
```

a) Generar un programa ejecutable.
b) Verificar los resultados obtenidos luego de la ejecución del programa.

Ejercicio 8

Lectura de un directorio.

Realizar un programa que muestre el número de inodo y los nombres de los archivos que existen dentro de un directorio. El nombre del directorio se ingresa como argumento desde la línea de comandos.

Ejercicio 9

Realizar un proceso productor y un proceso consumidor independientes, donde el proceso productor escribe en un archivo llamado buffer la producción para el proceso consumidor que realiza la lectura del archivo y borra la información del mismo, se deben sincronizar los procesos para que el productor no produzca si el consumidor aún no ha consumido y que el consumidor no consuma si no hay producción, de esta manera con un archivo simulamos un buffer limitado. La sincronización se debe realizar utilizando un archivo llamado sincro cuyo tamaño es de 1 byte y que simula el comportamiento de un semáforo binario el cual puede tomar únicamente dos valores, cero o uno. Por supuesto que, para el archivo que simula el semáforo se tienen que implementar las operaciones espera y señal que adolecen de un inconveniente (no son operaciones atómicas).

Criterios de corrección

En la corrección de este Trabajo Práctico, tendremos en cuenta los siguientes criterios:

Se evaluará la presentación del trabajo en tiempo y forma, las conclusiones obtenidas y el funcionamiento de los ejercicios en el Sistema Operativo LINUX.

Si tiene dudas o no puede resolver los ejercicios, no dude en consultar a su compañeros/as o su tutor/a.