Sesion-1-resuelta.pdf

KIKONASO

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Módulo II. Uso de los Servicios del SO mediante la API

Sesión 1. Llamadas al sistema para el SA (Parte I)

Ejercicio 1. ¿Qué hace el siguiente programa? Probad tras la ejecución del programa las siguientes órdenes del shell: $> cat archivo y $> od -c archivo

```
/*
tarea1.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10 compliant''
Probad tras la ejecución del programa: $>cat archivo y $> od -c archivo
*/
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
#include<errno.h>

char buf1[]="abcdefghij";
char buf2[]="ABCDEFGHIJ";
int main(int argc, char *argv[])
int fd;
if( (fd=open("archivo",O_CREAT|O_TRUNC|O_WRONLY,S_IRUSR|S_IWUSR))<0) {
        printf("\nError %d en open",errno);
        perror("\nError en open");
        exit(-1);
}
if(write(fd,buf1,10) != 10) {
        perror("\nError en primer write");
        exit(-1);
}
if(lseek(fd,40,SEEK_SET) < 0) {
        perror("\nError en lseek");
        exit(-1);
}
if(write(fd,buf2,10) != 10) {
        perror("\nError en segundo write");
        exit(-1);
}
close(fd);
return 0;
}
```

1. Apertura del archivo: Usa la llamada open para crear (o truncar si ya existe) el archivo archivo con permisos de lectura y escritura para el propietario (S_IRUSR | S_IWUSR). Si la apertura falla, muestra un error y termina el programa.
2. Primera escritura: Escribe los primeros 10 caracteres del arreglo buf1 ("abcdefghij") al comienzo del archivo.
3. Desplazamiento del offset: Usa lseek para mover el puntero de archivo 40 bytes hacia adelante desde el inicio del archivo (SEEK_SET). Esto crea un "hueco" en el archivo, es decir, un espacio vacío entre los datos escritos.
4. Segunda escritura: Escribe los 10 caracteres de buf2 ("ABCDEFGHIJ") a partir del byte 40 en el archivo, dejando el espacio intermedio vacío.
5. Cierre del archivo: Cierra el archivo con close.

Órdenes del shell

cat archivo: Mostrará el contenido visible del archivo (abcdefghijABCDEFGHIJ):

cat archivo
abcdefghijABCDEFGHIJ

od -c archivo: Permitirá ver los bytes del archivo en notación octal, incluyendo los 30 bytes del hueco (rellenos con \0), lo cual es característico en archivos dispersos (sparse files) en sistemas de archivos como el ext4:

od -c archivo
0000000 a b c d e f g h i j \0 \0 \0 \0 \0 \0
0000020 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000040 \0 \0 \0 \0 \0 \0 \0 \0 A B C D E F G H
0000060 I J
0000062

EJERCICIO 2. Implementa un programa que realice la siguiente funcionalidad. El programa acepta como argumento el nombre de un archivo (pathname), lo abre y lo lee en bloques de tamaño 80 Bytes, y crea un nuevo archivo de salida, salida.txt, en el que debe aparecer la siguiente información:

*** Bloque 1**
// Aquí van los primeros 80 Bytes del archivo pasado como argumento.
Bloque 2
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento.
...
Bloque n
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento.

El código está en la página de abajo.

```c
/*
tarea2.c
Implementa un programa que realice la siguiente funcionalidad. El programa
acepta como argumento el nombre de un archivo (pathname), lo abre y lo lee en bloques de
tamaño 80 Bytes, y crea un nuevo archivo de salida, salida.txt, en el que debe aparecer la
siguiente información:
 * Bloque 1
// Aquí van los primeros 80 Bytes del archivo pasado como argumento.
Bloque 2
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento.
...
Bloque n
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento.

*/

#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
#include<errno.h>


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

   // Verificación de argumentos
    if (argc!=2){
            perror("\nError, el programa tiene que ser usado con al menos un parámetro. Uso: ./tarea2c <archivo_texto> \n");
            exit(-1);
    }

    int fd_entrada, fd_salida;
    char buffer[80];
    int leidos=0;
    char header[20];
    int bloque_num=1;


    // Apertura del archivo de entrada en modo de solo lectura
    if( (fd_entrada=open(argv[1],O_RDONLY)) < 0) {
    printf("\nError %d en open de entrada",errno);
    perror("\nError en open");
    exit(-1);
    }

    // Creación del archivo de salida con permisos de escritura
    if( (fd_salida = open("salida.txt", O_CREAT | O_WRONLY | O_TRUNC, S_IRUSR | S_IWUSR)) < 0) {
    printf("\nError %d en open de salida",errno);
    perror("\nError en open");
    exit(-1);
    }

    while((leidos= read(fd_entrada, buffer, 80)) > 0){
        // Crear el encabezado del bloque
            //snprintf: Esta función se utiliza para formatear una cadena de texto (similar a sprintf)
            //y almacenarla en un buffer (en este caso, header). A diferencia de sprintf, snprintf limita
            //la cantidad de caracteres escritos en el buffer a un valor especifico, evitando desbordamientos de memoria.
            //int snprintf(char *str, size_t size, const char *format, ...);
        int len = snprintf(header, sizeof(header), "Bloque %d\n", bloque_num++);

        // Escribir el encabezado en el archivo de salida
        if (write(fd_salida, header, len) != len) {
            perror("Error al escribir el encabezado en el archivo de salida");
            close(fd_entrada);
            close(fd_salida);
            exit(-1);
        }

        // Escribir el contenido del bloque en el archivo de salida
        if (write(fd_salida, buffer, leidos) != leidos) {
            perror("Error al escribir en el archivo de salida");
            close(fd_entrada);
            close(fd_salida);
            exit(-1);
        }

        // Escribir una línea en blanco para separar bloques
        write(fd_salida, "\n\n", 2);
    }

    // Manejo de errores en la lectura
    if (leidos < 0) {
        perror("Error al leer el archivo de entrada");
    }

    // Cierre de archivos
    close(fd_entrada);
    close(fd_salida);
    return 0;


}
```

Modificación adicional. ¿Cómo tendrías que modificar el programa para que una vez finalizada la escritura en el archivo de salida y antes de cerrarlo, pudiésemos indicar en su primera línea el número de etiquetas "Bloque i" escritas de forma que tuviese la siguiente apariencia?:
El número de bloques es <nº_bloques>
Bloque 1
// Aquí van los primeros 80 Bytes del archivo pasado como argumento.
Bloque 2
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento.
...

```c

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

   // Verificación de argumentos
    if (argc!=2){
            perror("\nError, el programa tiene que ser usado con al menos un parámetro. Uso: ./tarea2c <archivo_texto> \n");
            exit(-1);
    }

    int fd_entrada, fd_salida;
    char buffer[80];
    int leidos=0;
    char header[50];
    int bloque_num=1;


    // Apertura del archivo de entrada en modo de solo lectura
    if( (fd_entrada=open(argv[1],O_RDONLY)) < 0) {
    printf("\nError %d en open de entrada",errno);
    perror("\nError en open");
    exit(-1);
    }

    // Creación del archivo de salida con permisos de escritura
    if( (fd_salida = open("salida.txt", O_CREAT | O_WRONLY | O_TRUNC, S_IRUSR | S_IWUSR)) < 0) {
    printf("\nError %d en open de salida",errno);
    perror("\nError en open");
    exit(-1);
    }

    //ESTO SE VA A SOBREESCRIBIR, MIRAR LINEA 101
    int aux = snprintf(header, sizeof(header), "Total de bloques escritos: 1\n");

        // Escribir el encabezado en el archivo de salida
        if (write(fd_salida, header, aux) != aux) {
            perror("Error al escribir el encabezado en el archivo de salida");
            close(fd_entrada);
            close(fd_salida);
            exit(-1);
        }

    while((leidos= read(fd_entrada, buffer, 80)) > 0){
        // Crear el encabezado del bloque
            //snprintf: Esta función se utiliza para formatear una cadena de texto (similar a sprintf)
            //y almacenarla en un buffer (en este caso, header). A diferencia de sprintf, snprintf limita
            //la cantidad de caracteres escritos en el buffer a un valor especifico, evitando desbordamientos de memoria.
            //int snprintf(char *str, size_t size, const char *format, ...);
        int len = snprintf(header, sizeof(header), "Bloque %d\n", bloque_num++);

        // Escribir el encabezado en el archivo de salida
        if (write(fd_salida, header, len) != len) {
            perror("Error al escribir el encabezado en el archivo de salida");
            close(fd_entrada);
            close(fd_salida);
            exit(-1);
        }

        // Escribir el contenido del bloque en el archivo de salida
        if (write(fd_salida, buffer, leidos) != leidos) {
            perror("Error al escribir en el archivo de salida");
            close(fd_entrada);
            close(fd_salida);
            exit(-1);
        }

        // Escribir una línea en blanco para separar bloques
        write(fd_salida, "\n\n", 2);
    }

    // Manejo de errores en la lectura
    if (leidos < 0) {
        perror("Error al leer el archivo de entrada");
    }

    // Escribir la línea con el número total de bloques al principio del archivo
    lseek(fd_salida, 0, SEEK_SET);  // Mover el puntero del archivo al inicio

    // Limpia el header antes de escribir el total de bloques
    snprintf(header, sizeof(header), "Total de bloques escritos: %d\n", bloque_num-1);

    // Asegúrate de escribir el número total de bloques correctamente
    if (write(fd_salida, header, strlen(header)) != strlen(header)) {
        perror("Error al escribir el total de bloques");
        close(fd_entrada);
        close(fd_salida);
        exit(-1);
    }

    // Cierre de archivos
    close(fd_entrada);
    close(fd_salida);
    return 0;


}
```

Ejercicio 3. ¿Qué hace el siguiente programa?

```
tarea2.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10 compliant''
*/
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>
int main(int argc, char *argv[])
{
int i;
struct stat atributos;
char tipoArchivo[30];
if(argc<2) {
printf("\nSintaxis de ejecucion: tarea2 [<nombre_archivo>]+\n\n");
exit(-1);
}
for(i=1;i<argc;i++) {
  printf("%s: ", argv[i]);
  if(lstat(argv[i],&atributos) < 0) {
    printf("\nError al intentar acceder a los atributos de %s",argv[i]);
    perror("\nError en lstat");
   }
   else {
    if(S_ISREG(atributos.st_mode)) strcpy(tipoArchivo,"Regular");
     else if(S_ISDIR(atributos.st_mode)) strcpy(tipoArchivo,"Directorio");
     else if(S_ISCHR(atributos.st_mode)) strcpy(tipoArchivo,"Especial de
caracteres");
     else if(S_ISBLK(atributos.st_mode)) strcpy(tipoArchivo,"Especial de
bloques");
     else if(S_ISFIFO(atributos.st_mode)) strcpy(tipoArchivo,"Cauce con nombre
(FIFO)");
     else if(S_ISLNK(atributos.st_mode)) strcpy(tipoArchivo,"Enlace relativo
(soft)");
     else if(S_ISSOCK(atributos.st_mode)) strcpy(tipoArchivo,"Socket");
     else strcpy(tipoArchivo,"Tipo de archivo desconocido");
     printf("%s\n",tipoArchivo);
   }
}
return 0;
}
```

Función main:

- Verifica que al menos un argumento (nombre de archivo) haya sido pasado al programa, además del nombre del propio ejecutable. Si no hay suficientes argumentos, imprime un mensaje de uso y termina.
- Luego, para cada archivo pasado como argumento, intenta obtener sus atributos mediante lstat(). Si lstat() falla, imprime un mensaje de error y pasa al siguiente archivo.

Identificación del tipo de archivo:

- Si lstat() es exitoso, usa macros de POSIX para determinar el tipo de archivo.
- Dependiendo del tipo, asigna una cadena descriptiva a tipoArchivo:
    - Regular: archivo regular.
    - Directorio: directorio.

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

- Especial de caracteres: archivo especial de caracteres.
- Especial de bloques: archivo especial de bloques.
- Cauce con nombre (FIFO): archivo FIFO o pipe.
- Enlace relativo (soft): enlace simbólico.
- Socket: archivo de socket.
- Si no coincide con ninguno de estos tipos, se asigna como "Tipo de archivo desconocido".

Salida:

- Imprime el tipo de cada archivo en la terminal.

Ejercicio 4. Define una macro en lenguaje C que tenga la misma funcionalidad que la macro S_ISREG(mode) usando para ello los flags definidos en <sys/stat.h> para el campo st_mode de la struct stat, y comprueba que funciona en un programa simple. Consulta en un libro de C o en internet cómo se especifica una macro con argumento en C.

#define S_ISREG2(mode) …

Para definir una macro en C que tenga la misma funcionalidad que S_ISREG(mode), necesitamos verificar que el valor de mode (que corresponde al campo st_mode en una estructura stat) indica que el archivo es regular.

La macro S_ISREG(mode) está definida en <sys/stat.h> y usa ciertos bits específicos en mode para determinar si el archivo es regular. En sistemas POSIX, el archivo es considerado regular si el campo st_mode contiene el flag S_IFREG.

Podemos definir nuestra propia macro S_ISREG2(mode) usando la comparación de bits con el valor S_IFMT, que se usa para extraer el tipo de archivo del campo st_mode. El valor S_IFREG indica específicamente que el archivo es regular.

```c
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>

#define S_ISREG2(mode) (((mode) & S_IFMT) == S_IFREG)

int main(int argc, char *argv[]) {
    struct stat atributos;

    if (argc < 2) {
        printf("Uso: %s <nombre_archivo>\n", argv[0]);
        return 1;
    }

    if (stat(argv[1], &atributos) < 0) {
        perror("Error al obtener atributos del archivo");
        return 1;
    }

    if (S_ISREG2(atributos.st_mode)) {
        printf("El archivo '%s' es un archivo regular.\n", argv[1]);
    } else {
        printf("El archivo '%s' NO es un archivo regular.\n", argv[1]);
    }

    return 0;
}

```

Explicación:

- S_IFMT es una máscara de bits que extrae el tipo de archivo del campo st_mode.
- S_IFREG es el valor que identifica un archivo regular.
- La macro S_ISREG2(mode) verifica que, al aplicar la máscara S_IFMT al campo mode, el resultado sea igual a S_IFREG.