Interfaz de llamadas del Sistema Operativo

Objetivos:

- Conocer la interfaz de llamadas del sistema operativo GNU/Linux exporta a las aplicaciones en espacio de usuario.
- Estudiar las llamadas del sistema operativo GNU/Linux relativas a la E/S de archivos.
- Comprender el concepto de descriptor de archivos, entrada, salida y error standard.
- Implementar programas en lenguaje C que invoquen rutinas definidas en espacio de kernel de forma alternativa a las rutinas definidas en la interfaz de llamadas standard de C.
- Comparar las implicaciones que involucran una llamada al sistema y una invocación de una rutina definida en espacio de usuario.

Pre-Laboratorio:

Investigue los puntos indicados a continuación, ya que la comprensión de los mismos facilitará el desarrollo del laboratorio:

1. Explore las páginas del manual relativas a las llamadas del sistema open *close read write lseek*. Invoque el comando *man -s 2 <llamada del sistema>* en un shell.
2. Investigue la utilidad, parámetros y valores de retorno de la rutina *malloc* y sus variantes (*calloc, free, realloc*)
3. Investigue acerca de la interfaz de errores standard estilo UNIX. Para ello aprenda el uso de la rutina *perror*(). ¿cómo se relaciona esta rutina con la variable de entorno *errno*?
4. Observe el contenido del archivo */usr/include/sys/errno.h* ¿cómo se manejan los permisos en Linux? En particular que significan los permisos *suid, sgid, sticky bit.* ¿cómo pueden ser asignados a un archivo?

Laboratorio

Cuando una computadora es encendida una serie de operaciones iniciales es ejecutada para poder arrancar el sistema operativo. Este programa controla mucha de la actividad realizada en el sistema. Esto incluye como se deberá gestionar el espacio de almacenamiento, como se administrará la memoria que dispone el sistema, como se comunicará el equipo con otros dispositivos externos y como será planificado el procesador, entre otras actividades.

La forma en que los programas en espacio de usuario se comunican con el sistema operativo es mediante la interfaz de llamadas al sistema. Una llamada al sistema (system call) puede lucir en primera instancia como la invocación de un procedimiento ordinario. Sin embargo son diferentes. Una llamada al sistema es una petición al sistema operativo para que éste realice una actividad.

Las llamadas al sistema en general suelen ser costosas. Esto es debido a que una llamada a procedimiento usualmente puede ser ejecutada por pocas instrucciones de máquina mientras que una llamada al sistema requiere que se preserve el estado en que se encontraba el sistema antes de ser interrumpido. Asimismo el sistema operativo debe tomar control del CPU, desempeñar alguna función, salvaguardar su estado y restaurar el estado del algún programa que se estaba ejecutando previamente.

Llamadas del sistema para las operaciones de E/S a archivos

Existen 5 llamadas al sistema que un sistema estilo Unix típico proporciona para realizar la E/S a archivos, estas son:

- *int open(char *path, int flags [, int mode]);*
- *int close(int fd);*
- *int read(int fd, char *buf, int size);*
- *int write(int fd, char *buf, int size);*
- *off_t lseek(int fd, off_t offset, int whence);*

Nótese que a pesar de que parecen llamadas a procedimientos regulares son llamadas al sistema, por tanto implican mucho más operaciones a ejecutar que una simple instrucción de salto a una rutina definida en otra parte del programa. A su vez una llamada a procedimiento en su implementación puede recurrir a la invocación de una llamada al sistema operativo.

Una razón por la cual el sistema operativo controla las operaciones de E/S es por seguridad. El sistema operativo debe asegurar que un programa defectuoso no afectará la ejecución de los demás programas. Por lo tanto si el programa en espacio de usuario necesita efectuar operaciones de E/S de red, disco o pantalla primero debe solicitar el servicio respectivo invocando la llamada al sistema apropiada.

Open

open realiza una petición al sistema operativo para usar un archivo. Si el archivo está disponible para ser usado retornará un descriptor de archivo. Esto es un número entero no negativo utilizado para hacer referencia al archivo en el código del programa. Retorna -1 en caso de error. La variable *errno* identifica el tipo de error ocurrido. Siempre que se desee manipular un archivo se debe especificar su descriptor de archivo correspondiente. Para obtener un descriptor de archivos el mismo debe ser solicitado invocando la respectiva llamada sistema.
Discuta con el docente el programa contenido en el archivo `o1.c`. Nótese la utilización de las banderas (flags) incluidas en el archivo de cabecera `fcntl.h`. Estas pueden ser encontradas en el directorio `/usr/include/fcntl.h` y `/usr/include/sys/fcntl.h`
Discuta con el docente los archivos `o2.c` y `o3.c` observe los permisos de apertura y el valor del argumento de modo `0644`.

Close

close es la llamada al sistema para cerrar un archivo. Esto se hace liberando el descriptor de archivo del archivo a cerrar. Esto se debe efectuar para evitar que un mismo archivo tenga varios descriptores de archivo asignados simultáneamente. Compile y ejecute el archivo `c1.c` y discuta con el docente las ventajas y desventajas de tener asignados varios descriptores de archivos diferentes al mismo archivo.

Read

read es la llamada del sistema que permite leer una cantidad específica de bytes de un archivo abierto para luego almacenarlos en una posición de memoria particular. Retornará cuantos bytes fueron realmente leídos del archivo. Si retorna cero entonces se ha alcanzado el final del archivo.
Discuta con el docente el programa contenido en el archivo `r1.c`. Nótese que incluso los caracteres no imprimibles son leídos por *read*

Write

Análogo a *read*, esta llamada escribirá en un archivo los bytes contenidos en una posición de memoria particular. Observe el archivo `w1.c` y discuta el código allí contenido.

Lseek

Todos los archivos abiertos poseen un apuntador de archivo asociado al mismo. Cuando el archivo es abierto, el apuntador de archivo apunta al principio del archivo. A medida que el archivo es leído o escrito este apuntador se desplaza tantos bytes como se hayan escrito o

leído. *lseek* permite desplazarse por el archivo de forma directa dependiendo los parámetros que sean pasados a la invocación de la llamada. Este desplazamiento puede ocurrir desde el principio, desde el final o desde la posición actual. El valor de retorno será el desplazamiento que haya incurrido el puntero.

Entrada Salida y Error Standard

Cada proceso en un sistema Unix posee los siguientes tres descriptores de archivos predefinidos:

- `standard input 0`
- `standard output 1`
- `standard error 2`

Discuta con el docente la importancia de estos descriptores de archivos.

Evaluación

La evaluación será de la siguiente manera:

- **Quiz de 3 preguntas acerca de lo visto en el laboratorio o pre-laboratorio *(*5 puntos)**
- **Corregir exitosamente el** *ejercicio_evaluado1.c* **(Eliminar basura al imprimir el buffer y impresión de perror = Success) (5 puntos)**
- **Simular el comando** *cat* **siguiendo la plantilla dada por el grupo docente** *ejercicio_evaluado2.c.* cat **funciona con parámetros y sin parámetros. Su implementación debe soportar ambos casos. (10 puntos)**

GDSO/cd