Organización y Arquitectura de las Computadoras

Práctica 08: Llamadas al Sistema

Bonilla Reyes Dafne
319089660

Medina Guzmán Sergio
314332428

1. Objetivo

Introducir al alumno a los conceptos básicos de operaciones de entrada/salida, cubriendo la interacción entre el usuario, las rutinas, el sistema operativo y el hardware.

2. Procedimiento

Escribe las rutinas necesarias en lenguaje ensamblador de MIPS respetando los siguientes lineamientos:

- El programa debe terminar con la llamada al sistema exit.
- Deberás entregar un único archivo de código fuente.
- No olvides documentar el código.

3. Ejercicios

1. Desarrolla los 6 comandos descritos en la sección 4.2.
2. Agrega 2 comandos extra, debe ser de una complejidad similar a los otros. No olvides documentarlo para el comando help.

4. Preguntas

1. ¿Qué es `shell`? Da un pequeño resumen sobre que es shell, para qué se usa y como funciona.

   El shell es la capa de interfaz más visible y accesible para el usuario en el sistema operativo. Los diferentes tipos de shell ofrecen la posibilidad de utilizar un lenguaje de programación específico para la gestión de procesos y archivos, así como para lanzar y controlar otros programas. Su función principal es facilitar la interacción entre el usuario y el sistema operativo mediante la interpretación de las órdenes introducidas por el usuario y su comunicación con el núcleo del sistema operativo. El shell es el encargado de solicitar y procesar las entradas del usuario, y de gestionar cualquier información devuelta por el sistema operativo. [1]

   Los shells permiten establecer una comunicación con el sistema operativo, lo cual se realiza de dos maneras: de forma interactiva, en la que se responde directamente a la entrada del teclado, o a través de un script de shell, que es una serie de comandos del sistema operativo y del shell que se guarda en un archivo.

   Cuando se inicia sesión en el sistema, el sistema busca un programa de shell para ejecutar. Una vez ejecutado, el shell muestra un símbolo del sistema, que a menudo es un signo de dólar ($). Al ingresar un comando en el símbolo y presionar `enter`, el shell evalúa el comando y trata de ejecutarlo. Dependiendo de las instrucciones del comando, el shell mostrará la salida del comando en la pantalla

o la redirigirá a otra parte. Luego, el shell regresará al símbolo del sistema y esperará que se ingrese otro comando. [2]

Una línea de comando se refiere a la línea donde se escribe el texto, que incluye el indicador de shell. El formato básico para cada línea es el siguiente:

```
$ [Comando] [Opción] [Argumento]
```

2. ¿Cuál es el propósito de un intérprete de comandos para cualquier sistema?

   Un intérprete de comandos es un programa que permite al usuario interactuar con otro programa mediante la introducción de comandos en forma de líneas de texto. Este enfoque fue muy común hasta la década de 1970. No obstante, en la actualidad, muchas de estas interfaces de texto han sido reemplazadas por interfaces gráficas de usuario o interfaces basadas en menús.

   Los intérpretes de comandos pueden ser útiles para diversos fines y, en ocasiones, pueden resultar más convenientes que las interfaces gráficas de usuario. A continuación, se proporcionarán más detalles acerca de estas situaciones específicas: [3]

   - Existen sistemas que no poseen los recursos necesarios para soportar interfaces gráficas de usuario, por lo que en dichas situaciones se recurre a los intérpretes de comandos.
   - Los intérpretes de comandos ofrecen una gran variedad de comandos y consultas para llevar a cabo distintas operaciones. Además, escribir comandos es mucho más rápido que hacer clic, como se hace en las interfaces gráficas de usuario.
   - En ambientes científicos, los científicos e ingenieros frecuentemente utilizan intérpretes de comandos. Asimismo, los usuarios con conocimientos técnicos avanzados suelen preferir los intérpretes de comandos por encima de las interfaces gráficas de usuario.
   - Las personas con discapacidades visuales utilizan intérpretes de comandos debido a que no pueden operar con interfaces gráficas de usuario. Los comandos e instrucciones pueden ser mostrados a través de pantallas braille en los intérpretes de comandos.

3. Si tuviéramos que implementar el intérprete de comandos en un sistema operativo. ¿Por qué es importante usar subrutinas?

   Las subrutinas son útiles por varias razones, incluyendo la reutilización de código. En lugar de tener que escribir el mismo código varias veces en diferentes partes del programa, se puede escribir una sola vez en una subrutina y llamar a esa subrutina desde diferentes lugares. Esto no solo ahorra tiempo y reduce la cantidad de código que se necesita escribir, sino que también hace que el mantenimiento y la actualización del programa sea más fácil. Si se necesita hacer un cambio en el código de la subrutina, solo se debe hacer en un solo lugar, y no en varios lugares en todo el programa.

   Además, las subrutinas también fomentan la modularidad del código. Al dividir el código en módulos más pequeños y manejables, se puede hacer que el código sea más fácil de entender y mantener. Las subrutinas también pueden ser utilizadas para separar la lógica del programa de la implementación, lo que hace que el código sea más comprensible para los programadores.

   Las subrutinas también facilitan la depuración del código. Si se divide el código en subrutinas más pequeñas, se pueden aislar problemas y encontrar la causa raíz en una subrutina específica, lo que hace que sea más fácil solucionar los problemas. Además, si se necesita cambiar el comportamiento de una subrutina, solo se debe hacer en un solo lugar en lugar de tener que buscar y modificar el mismo código en múltiples lugares.

   Finalmente, las subrutinas también permiten ahorrar tiempo y memoria al evitar la necesidad de repetir el mismo código varias veces. En lugar de cargar múltiples copias del mismo código en la memoria, solo se debe cargar una sola copia de la subrutina para ser reutilizada cada vez que sea necesario. Esto ahorra tiempo en la carga del programa y también ahorra memoria, lo que es especialmente importante en sistemas con recursos limitados. [4]

4. Considerando los componentes básicos de una computadora moderna, además de las llamadas al sistema descritas en la práctica, ¿qué otras llamadas al sistema resultarían necesarias?

   Existen muchas otras llamadas al sistema que resultan necesarias para el funcionamiento de una computadora moderna. Algunas de las más comunes incluyen: [5]

   - *Gestión de procesos:* Estas llamadas permiten a los programas crear, modificar y controlar procesos en el sistema operativo. Algunas de las llamadas más comunes son `fork`, `execute`, `wait`, y `exit`.
   - *Gestión de memoria:* Estas llamadas permiten a los programas asignar y liberar memoria en el sistema. Algunas de las llamadas más comunes incluyen `malloc`, `free`, `map`, y `protect`.
   - *Gestión de red:* Estas llamadas permiten a los programas interactuar con la red y realizar operaciones de comunicación. Algunas de las llamadas más comunes incluyen `socket`, `bind`, `connect`, y `send`.
   - *Gestión de tiempo:* Estas llamadas permiten a los programas acceder al reloj del sistema y a otros recursos de temporización. Algunas de las llamadas más comunes incluyen `gettimeofday`, `sleep`, y `nanosleep`.

5. De forma general, ¿Qué comandos son básicos para un intérprete de comandos de un sistema operativo?

   Como se mencionó en el ejercicio anterior, para cualquier sistema operativo es importante la gestión de archivos, redes, sistema y procesos. Considerando que estamos trabajando sobre un sistema operativo Unix/Linux, los siguientes comandos serían básicos para el intérprete, aunque de manera análoga se pueden tomar los mismos comando en semántica, pero adaptados en sintaxis al sistema operativo que se desee: [6]

   - **Comandos para la gestión de archivos y directorios:**
     - `cd:` Cambiar el directorio actual
     - `cp:` Copiar archivos y directorios
     - `ls:` Listar el contenido del directorio actual
     - `mkdir:` Crear un nuevo directorio
     - `rm:` Eliminar archivos y directorios
     - `mv:` Mover o renombrar archivos y directorios
     - `pwd:` Imprimir el directorio actual
   - **Comandos para la gestión de procesos:**
     - `ps:` Listar los procesos en ejecución
     - `kill:` Terminar un proceso
     - `top:` Mostrar los procesos que están consumiendo más recursos
   - **Comandos para la gestión de redes:**
     - `ping:` Verificar la conectividad con otra máquina en la red
     - `ifconfig:` Configurar la interfaz de red y mostrar información sobre las interfaces de red
     - `netstat:` Mostrar información sobre las conexiones de red
   - **Comandos para la gestión del sistema:**
     - `su:` Cambiar al usuario root (superusuario)
     - `sudo:` Ejecutar un comando con los privilegios del superusuario
     - `apt-get:` Instalar, actualizar o eliminar paquetes en sistemas basados en Debian.

6. Investiga y describe con tus propias palabras ¿cómo resuelve una llamada al sistema el sistema operativo?

   Una llamada al sistema es un mecanismo que permite a un programa interactuar con el sistema operativo. El programa envía solicitudes de servicios al sistema operativo y este responde mediante la

invocación de una serie de llamadas al sistema. Estas llamadas pueden ser escritas en lenguaje ensamblador o en un lenguaje de alto nivel.

Una llamada al sistema es una forma en que un programa de computadora puede solicitar un servicio del kernel del sistema operativo en el que se está ejecutando. Es básicamente una petición que el software de la computadora hace al núcleo del sistema operativo para obtener ciertas funcionalidades o recursos.

La Interfaz de Programa de Aplicación (API) conecta las funciones del sistema operativo con los programas de usuario, actuando como un puente entre el sistema operativo y un proceso. De esta forma, los programas de nivel de usuario pueden solicitar servicios al sistema operativo. Las llamadas al sistema son el único medio para acceder al sistema kernel y son necesarias para cualquier programa que requiera recursos.

Las aplicaciones se ejecutan en un área específica de memoria llamada espacio de usuario. Cuando una aplicación crea una llamada al sistema, esta se conecta al kernel del sistema operativo que se ejecuta en otra área de memoria llamada espacio del kernel. Para obtener permiso del kernel, la aplicación debe enviar una solicitud de interrupción, que detiene el proceso actual y transfiere el control al núcleo del sistema operativo.

Si la solicitud de la aplicación es aprobada, el kernel del sistema operativo lleva a cabo la tarea solicitada, como por ejemplo, crear o eliminar un archivo. La aplicación recibe los resultados de la tarea llevada a cabo por el kernel. Después de recibir la respuesta, la aplicación continua con su proceso. Cuando la tarea se completa, el kernel del sistema operativo devuelve los resultados a la aplicación y mueve los datos de la memoria del kernel al espacio de usuario en la memoria.

Se puede tardar unos pocos nanosegundos para obtener un resultado simple, como la fecha y hora del sistema, mediante una llamada al sistema. Sin embargo, para una solicitud más compleja, como conectarse a un dispositivo de red, puede tardar varios segundos. Para evitar cuellos de botella, la mayoría de los sistemas operativos crean un subproceso de kernel separado para cada llamada al sistema. Los sistemas operativos modernos tienen múltiples subprocesos, lo que les permite manejar varias llamadas al sistema simultáneamente. [7]

Referencias

[1] Operating System Shells

[2] Introduction to Linux Shell and Shell Scripting

[3] What is the purpose of the command interpreter?

[4] Bach, M. J. (1986). *The design of the Unix operating system.* Prentice Hall.

[5] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts.* (10th ed.). Wiley.

[6] Basic Shell Commands in Linux

[7] System Calls in Operating System