3)

El comando **strace** en Linux es una herramienta poderosa para rastrear las llamadas al sistema y las señales que un proceso realiza o recibe. Según la documentación oficial, su propósito principal es interceptar y registrar estas interacciones, proporcionando detalles como el nombre de la llamada al sistema, sus argumentos y el valor de retorno.

Es especialmente útil para:

Diagnóstico y depuración: Ayuda a identificar problemas en programas sin necesidad de acceder al código fuente.

Aprendizaje: Permite a estudiantes y curiosos entender cómo funcionan las llamadas al sistema.

Aislamiento de errores: Los programadores pueden usarlo para examinar el límite entre el usuario y el kernel, lo que es útil para detectar condiciones de carrera o errores.

Por ejemplo, al ejecutar strace ls, podrías ver las llamadas al sistema que realiza el comando ls para listar archivos. Si necesitas más detalles o ejemplos, ¡puedo ayudarte a explorarlo! 😊

c)

Detalle de las opciones:

* -a <column>: Ajusta la alineación de la salida para mejorar la legibilidad.
* -e <expr>: Filtra las llamadas al sistema según una expresión específica (por ejemplo, solo mostrar open y read).
* -o <file>: Guarda la salida en un archivo en lugar de mostrarla en pantalla.
* -p <PID>: Adjunta strace a un proceso en ejecución para monitorear sus llamadas al sistema.
* -s <strsize>: Define el tamaño máximo de las cadenas de texto que se muestran en la salida.

4)

a)

El comando **ps** en Linux se utiliza para mostrar información sobre los procesos en ejecución en el sistema. De acuerdo con la documentación oficial, permite ver una instantánea del estado de los procesos en el momento en que se ejecuta el comando.

Características principales de ps:

Información básica de los procesos: Muestra el ID de proceso (PID), el usuario que ejecuta el proceso, el uso de CPU y memoria, el tiempo de ejecución, entre otros detalles.

Filtros personalizados: Puedes usar opciones y argumentos para filtrar y mostrar procesos específicos.

Formatos de salida flexibles: Permite personalizar el formato en que se muestran los datos.

b)

La sintaxis básica de ps es: ps [opciones] Si se ejecuta sin opciones, muestra solo los procesos asociados a la terminal actual.

c)

El comando ps admite **dos estilos de sintaxis** principales:

1. **Estilo BSD (sin guiones)**
   * Se usa sin guiones (-) y permite agrupar opciones.
   * Ejemplo: ps aux
2. **Estilo UNIX (con guiones)**

* Usa guiones (-) y no permite agrupar opciones.
* Ejemplo: ps -ef
* Opciones comunes:
* -e: Muestra todos los procesos del sistema.
* -f: Formato completo con detalles adicionales.

5)

En C, las llamadas al sistema están definidas en archivos de cabecera como <unistd.h> y <sys/types.h>.

**a. Maneras de realizar llamadas al sistema**

1. **Funciones estándar de la biblioteca C**, como getcwd(), chdir(), time(), que encapsulan llamadas al sistema.
2. **Uso directo de** syscall(), que permite invocar llamadas al sistema con su número correspondiente.

**b. Información sobre getcwd, chdir y time**

* getcwd(char \*buf, size\_t size): Obtiene el directorio actual. Retorna la ruta en buf o NULL en caso de error.
* chdir(const char \*path): Cambia el directorio de trabajo. Retorna 0 si tiene éxito o -1 si falla.
* time(time\_t \*t): Obtiene el tiempo actual en segundos desde 1970. Retorna el tiempo o -1 si hay error.

**d. Uso de ps y strace**

Ejecuta el programa con strace ./programa para ver las llamadas al sistema. Usa ps aux | grep programa para verificar su ejecución.

**e. Significado de los valores retornados**

Cada llamada devuelve un valor que indica éxito o error. getcwd retorna la ruta, chdir devuelve 0 si cambia el directorio y time retorna el tiempo en segundos desde 1970.

6) b)

Las diferencias que encuentro con la forma en la que se realiza la apertura, lectura y cerrado del archivo son casi pocas, siguen la misma lógica que el programa escrito en C, pero las system calls tiene mas parámetros, pero las funciones de apertura y cerrado del archivo son muy similares. El comando getline() de C es similar al read se las system calls, asi que las diferencias solamente puede ser que radiquen en la cantidad de parámetros que manejas las funciones del SO.