

Universidad Del Valle De Guatemala

Sebastian Galindo

Sistemas operativos



Laboratorio 1

Jose Andres Hernández Guerra 20053

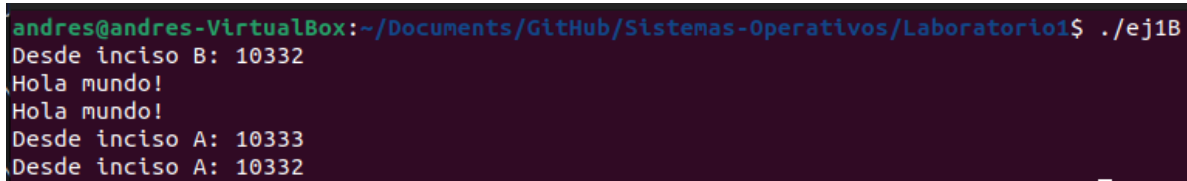
Guatemala, 21 de enero del 2023

Ejercicio 1:

Compile el primer programa y ejecútalo varias veces. Responda: ¿por qué aparecen números diferentes cada vez?

La función `getpid()` retorna el ID del proceso que se está llamando en ese instante, cada vez que se corre el programa un nuevo proceso se crea con un ID único. Por consiguiente cada vez que se corre el programa y se ejecuta el método se retorna un número distinto.

Proceda a compilar el segundo programa y ejecútalo una vez. ¿Por qué aparecen dos números distintos a pesar de que estamos ejecutando un único programa?



```
andres@andres-VirtualBox:~/Documents/GitHub/Sistemas-Operativos/Laboratorio1$ ./ej1B
Desde inciso B: 10332
Hola mundo!
Hola mundo!
Desde inciso A: 10333
Desde inciso A: 10332
```

Esto se debe a que `fork` crea un segundo proceso con una copia exacta del programa, por lo que en realidad hay dos procesos abiertos, esto pasa por las siguientes razones:

- El segundo programa es ejecutado.
- Se llama a la función `fork()` en el segundo programa. Esto crea un proceso hijo nuevo con una copia exacta del código del proceso padre.
- En el proceso hijo, la variable `"f"` es igual a 0, por lo que se entra en el primer ramo del bloque `if/else`.
- En el proceso hijo, se llama a `execl("ej1A", (char*)NULL)`. Esto sustituye el código actual del proceso hijo con el código del primer programa.
- El primer programa es ejecutado en el proceso hijo, imprimiendo "Hola mundo!" y "Desde inciso A: [PID del proceso hijo]" en la consola.
- El proceso hijo termina de ejecutar el primer programa y finaliza.
- En el proceso padre, la variable `"f"` no es igual a 0, por lo que se entra en el segundo ramo del bloque `if/else`.
- En el proceso padre, se llama a `execl("ej1A", (char*)NULL)`. Esto sustituye el código actual del proceso padre con el código del primer programa.
- El primer programa es ejecutado en el proceso padre, imprimiendo "Desde inciso B: [PID del proceso padre]" y "¡Hola mundo!" y "Desde inciso A: [PID del proceso padre]" en la consola.
- El proceso padre termina de ejecutar el primer programa y finaliza.

¿Por qué el primer y el segundo números son iguales?

Esto se debe a que se están ejecutando en un solo proceso del sistema.

En la terminal, ejecute el comando `top` (que despliega el top de procesos en cuanto a consumo de CPU) y note cuál es el primer proceso en la lista (con identificador 1). ¿Para qué sirve este proceso?

Un proceso con un ID de proceso (PID) de 1 es típicamente el primer proceso que se inicia en un sistema Linux después de que el núcleo se ha inicializado. Este proceso se llama el proceso "init" y es responsable de inicializar los servicios del sistema y de iniciar otros procesos. El proceso init típicamente tiene un PID de 1 y suele ser el proceso padre de todos los demás procesos que se ejecutan en el sistema.

Ejercicio 2

Observe el resultado desplegado. ¿Por qué la primera llamada que aparece es `execve`?

Cuando se ejecuta el comando `strace` en un programa, `strace` intercepta y registra las llamadas al sistema realizadas por el programa. La primera llamada al sistema realizada por el programa es típicamente `execve`, que se utiliza para ejecutar una nueva instancia del programa.

Ubique las llamadas de sistema realizadas por usted. ¿Qué significan los resultados (números que están luego del signo '=')?

Los números después de los signos de igual en cada llamada al sistema representan el valor de retorno de la llamada al sistema. En caso de que haya algún error, el valor de retorno puede ser un valor negativo que representa un código de error, o un valor positivo que representa un valor específico dependiendo de la llamada al sistema.

¿Por qué entre las llamadas realizadas por usted hay un `read` vacío?

`read(3, "", 1024) = 0`, indica que la llamada está intentando leer datos del descriptor de archivo 3, pero el buffer pasado está vacío y el número de bytes a leer es 1024. Y el valor de retorno es 0, lo que significa que no se han leído bytes.

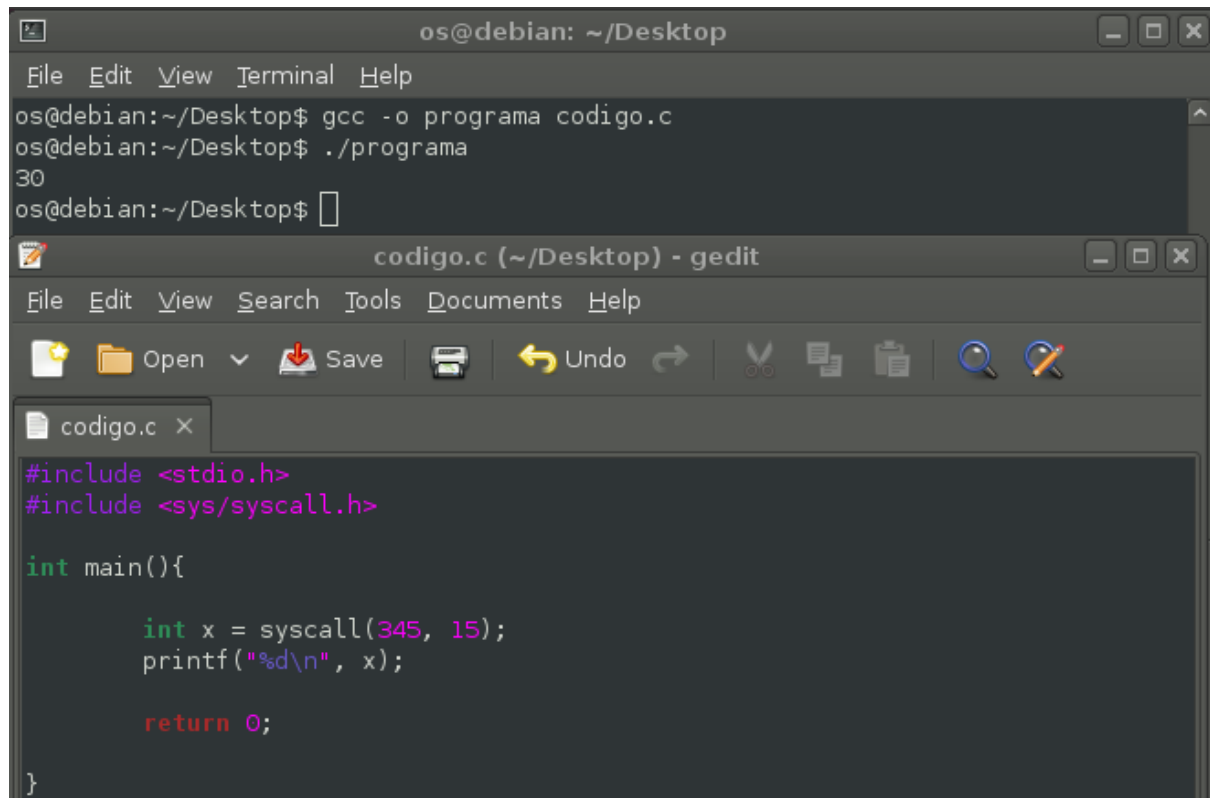
Identifique tres servicios distintos provistos por el sistema operativo en este `strace`. Liste y Explique brevemente las llamadas al sistema que corresponden a los servicios identificados.

1. `execve`: Este servicio es el encargado de ejecutar un programa en el sistema. En este caso, la llamada al sistema `execve` se usa para ejecutar el programa `./ej2copy`, y los argumentos y variables de entorno se pasan como parámetros.
2. `brk`: Este servicio es el encargado de establecer la ubicación del punto de interrupción del programa, que es el final del segmento de datos en la memoria. En este caso, la llamada al sistema `brk` se usa para consultar la ubicación actual del punto de interrupción del programa y se pasa como parámetro `NULL`.
3. `mmap`: Este servicio es el encargado de mapear un archivo o una región de memoria en el espacio de direcciones de un proceso. En este caso, las llamadas al sistema `mmap` se utilizan para mapear varias regiones de memoria, con diferentes permisos de lectura y escritura, y se pasa como parámetro `NULL` para indicar que se debe asignar una nueva ubicación en la memoria para el mapeo.

Ejercicio 3

Compile y ejecute su programa. Si se despliega la suma de 15 con el número que usted haya dejado en la llamada a sistema, ha agregado exitosamente su propia llamada de sistema.

El número que yo he elegido colocar en la system call fue '15'.



The screenshot shows two windows. The top window is a terminal titled 'os@debian: ~/Desktop'. It shows the following commands and output:

```
os@debian:~/Desktop$ gcc -o programa codigo.c
os@debian:~/Desktop$ ./programa
30
os@debian:~/Desktop$
```

The bottom window is a code editor titled 'codigo.c (~/Desktop) - gedit'. It shows the following C code:

```
#include <stdio.h>
#include <sys/syscall.h>

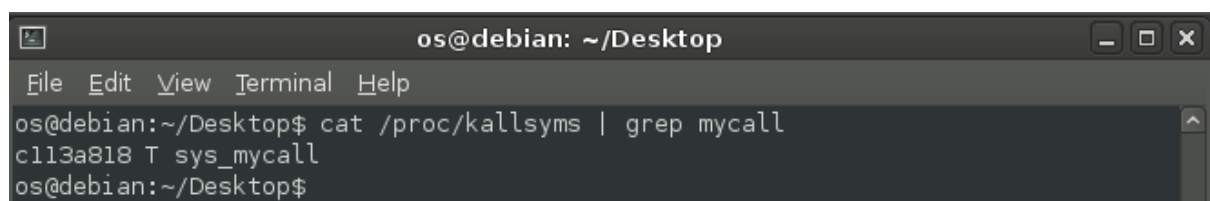
int main(){

    int x = syscall(345, 15);
    printf("%d\n", x);

    return 0;
}
```

puede revisar que el sistema operativo cuente con la llamada sys mycall usando el siguiente comando en la terminal:

cat /proc/kallsyms | grep mycall



The screenshot shows a terminal window titled 'os@debian: ~/Desktop'. It shows the following command and output:

```
os@debian:~/Desktop$ cat /proc/kallsyms | grep mycall
c113a818 T sys_mycall
os@debian:~/Desktop$
```

¿Qué ha modificado aquí, la interfaz de llamadas del sistema o el API? Justifique su respuesta.

Al haber creado mi propio encabezado y lo estoy llamando como 'sys/syscall.h' estoy haciendo uso de una interfaz diferente a la que me proporciona la biblioteca de C. Por lo que no estoy haciendo uso de su API, más exactamente lo que estoy haciendo es utilizando las llamadas al sistema para que mi código acceda a servicios del kernel de mi sistema

operativo. En este caso simplemente estoy realizando una suma que escribí como la instrucción 345 de mi linux-2.6.39.4/mycall.

¿Por qué usamos el número de nuestra llamada de sistema en lugar de su nombre?



Esto es porque la función `syscall()` toma un argumento entero que especifica el número de la llamada al sistema a ser ejecutada. En UNIX las llamadas al sistema se identifican mediante un número único y cada llamada al sistema tiene un número único asociado a ella.

¿Por qué las llamadas de sistema existentes como `read` o `fork` se pueden llamar por nombre?

Las llamadas al sistema como `read` o `fork` se pueden llamar por nombre y no solo por su número, debido a que existen funciones de la biblioteca en C que proporcionan una interfaz para invocar esas llamadas al sistema. Estas funciones de biblioteca están definidas en el encabezado de `<unistd.h>`. Por ejemplo, en lugar de tener que llamar a `syscall()` con el número de llamada al sistema correspondiente a `read`, se puede utilizar la función `read()` de la biblioteca estándar de C, que se encarga de ejecutar la llamada al sistema correspondiente. Esto es simplemente una interfaz de alto nivel para invocar llamadas al sistema de forma más sencilla y legible.