**LABORATORIO 1**

**Ejercicio 1.**

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

Porque cada vez que se ejecuta el programa, el proceso que lo ejecuta es diferente.

Graphical user interface, text

Description automatically generated

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

Porque con la llamada a sistema ***fork* ()** se creó un proceso hijo, el cual cuenta con un id distinto al del padre.

Graphical user interface, text, application

Description automatically generated

* **¿Por qué el primer y el segundo número son iguales?**

Esto ocurre, ya que el valor que retorna la llamada a sistema ***fork()*** para el proceso padre es mayor a 0. Por lo tal, en el programa, esto significa que entra en la cláusula ***else*** imprimiendo primero el id del proceso padre, y luego al ejecutar el programa del inciso anterior, muestra en pantalla nuevamente dicho id.

* **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?**

El proceso con identificador 1, se conoce también como ***init*,** es en el encargado de generar todos los demás procesos ejecutados por el kernel durante la carga del sistema. De igual forma, define cómo el sistema debe de ser configurado en cada nivel de ejecución y determina un nivel por defecto. Se inicia automáticamente en el sistema Linux, y lo primero que hace es leer un archivo de configuración del ambiente. (Introduction lo Linux, s. f.)

**Ejercicio 2.**

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

**Text

Description automatically generated**

La llamada a sistema ***execve*** ejecuta el programa que tiene como primer parámetro, y reemplaza el programa que se está ejecutando en el proceso de llamada, inicializando segmentos de dato y de pila. Es debido a esto que se encuentra como la primera llamada en la secuencia de ejecución del programa ya que es la llamada a sistema que se encarga de ejecutar dicho programa. (LINUX Manual, s. f.-b)

* **Ubique las llamadas a sistema realizadas por usted. ¿Qué significan los resultados (números que están luego del signo ‘=’)?**
* Open
* Read
* Close
* Write

Strace retorna las llamadas a sistema realizadas, mostrando en pantalla, el tipo de llamada a sistema y el valor de retorno de dicha llamada, es decir, los números que se encuentran luego del “=”. Algo interesante es que, cuando el valor de retorno es -1 esto indica que hubo un error en la llamada de lo contrario retorna valore enteros mayores a -1.

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

Luego de observar los datos mostrados por *strace* y revisar cuidadosamente entre cada una de las llamadas realizadas por mí, no se pudo encontrar un read vacío.

Text

Description automatically generated

* **Identifique 3 servicios distintos provistos por el sistema operativo. Lístelos y explíquelos brevemente.**

1. **Brk(0) →** se utiliza para modificar la cantidad de memoria asignada al proceso de llamada. Pasando como parámetro el valor que establece el final del segmento de datos. (LINUX Manual, s. f.-a)
2. **Open(/etc/ld.so.cache, O\_RDONLY)**
3. **Mmap(NULL, 64828, PROT\_READ, MAP\_PRIVATE, 3, 0) →** mapea o asigna dispositivos o archivos en memoria. Su valor de retorno indica el lugar en el cual el objeto se ha asignado. Cuenta con ciertos parámetros, en el caso del argumento PROT, este describe la protección deseada en memoria, y puede ser none, o como en este caso, read, lo que indica que las páginas deben de ser leídas. En el caso del argumento MAP\_PRIVATE, la cual crea una copia privada de escritura, (Tutorials Point, s. f.-a)
4. **Mprotect(0xb7790000, 4096, PROT\_NONE) →** define protección en una región de memoria. Básicamente, especifica la protección deseada para ciertas páginas de memoria. En este caso particular, PROT\_NONE significa que ese espacio de memoria no se puede ser accesado. Si la llamada a sistema ha sido exitosa entonces retorna 0, si por otro lado ocurre un error entonces devuelve -1. (Tutorials Point, s. f.-b)

**Ejercicio 3.**

**Graphical user interface, text, application

Description automatically generated**

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

La modificación se ha realizado en la interfaz de llamadas a sistema, esto ya que, como bien se sabe, una llamada a sistema es una manera en la cual se requiere un servicio del kernel del sistema operativo. Por lo tal, la implementación de las llamadas al sistema se implementa en el kernel, a diferencia del API, que se implementa en la biblioteca de funciones, y un API más que definir una llamada a sistema, o crearla, se encarga de encapsular dichas llamadas. (Mandark, 2020)

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

Esto ocurre debido a que, como se mencionó anteriormente, si bien, se creó una nueva llamada a sistema, esta no se ha encapsulado por medio de un API. El API básicamente son funciones que especifican cómo obtener un servicio específico, en este caso particular, una llamada a sistema, y actúa como mensajero entre las aplicaciones y el kernel. (Mandark, 2020)

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

En complemento con la respuesta anterior, las llamadas a sistema como read o fork, se pueden invocar mediante su nombre debido a que ya cuentan con una rutina de encapsulamiento en el API. Una rutina de encapsulamiento es una función cuyo único propósito es hacer la llamada a sistema. Cabe resaltar también que una función de un API puede realizar diversas llamadas a sistema. (Mandark, 2020)

* **Evidencia de la creación de llamada a sistema**

**Graphical user interface, text, application

Description automatically generated**

**Bibliografía**

Introduction lo Linux. (s. f.). *Boot process, Init and shutdown*. TLD. https://tldp.org/LDP/intro-linux/html/sect\_04\_02.html

LINUX Manual. (s. f.-a). *brk(2) - Linux manual page*. Man7.Org. https://man7.org/linux/man-pages/man2/brk.2.html

LINUX Manual. (s. f.-b). *execve(2) - Linux manual page*. Man7.Org. https://man7.org/linux/man-pages/man2/execve.2.html

Mandark, D. (2020, 9 febrero). *What is system call interface?* Stack Overflow. https://stackoverflow.com/questions/60136893/what-is-system-call-interface

Tutorials Point. (s. f.-a). *mmap() - Unix, Linux System Call*. https://www.tutorialspoint.com/unix\_system\_calls/mmap.htm

Tutorials Point. (s. f.-b). *mprotect() - Unix, Linux System Call*. https://www.tutorialspoint.com/unix\_system\_calls/mprotect.htm