Universidad del Valle de Guatemala Rodrigo Alvarado
16106
Sistemas Operativos
Tomás Gálvez

Laboratorio 1

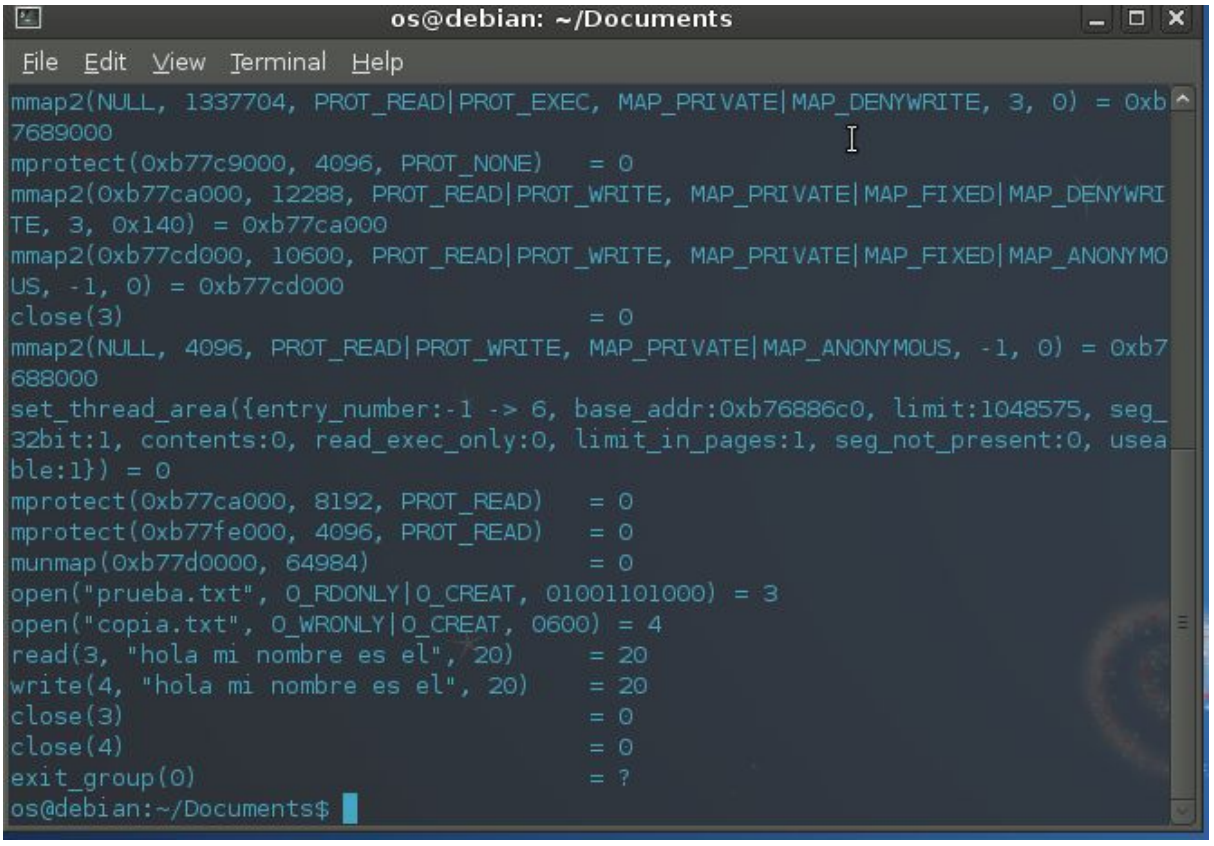
Ejercicio 1

Compile el primer programa y ejecútalo varias veces. Responda:

- **¿Por qué aparecen números diferentes cada vez?**
  Porque cada vez que se corre es un proceso diferente, por lo tanto tienen un identificador diferente a pesar de ser el mismo programa.
- **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?**
  Dado que el proceso hijo tiene un identificador diferente al del padre para que el sistema operativo sea capaz de diferenciarlos. Un número es el resultado de la llamada de getpid en el hijo y otro en el padre.
- **¿Por qué el primer y el segundo números son iguales?**
  Dado que el identificador del proceso hijo creado con el fork se imprime por segunda vez al ejecutar el programa “Hello world”. exc no genera un proceso separado
- **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?**
  Xorg: Es un display server, permite tener GUI.

Ejercicio 2

- **Observe el resultado desplegado. ¿Por qué la primera llamada que aparece es execve?**
  Porque es el comando que ejecuta el programa.
- **Ubique las llamadas de sistema realizadas por usted. ¿Qué significan los resultados (números que están luego del signo ‘=’)?**
  El return value de cada línea
- **¿Por qué entre las llamadas realizadas por usted hay un Read vacío?**
  No tengo tal read vacío



- **Identifique tres servicios distintos provistos por el sistema operativo en este Strace. Liste y explique brevemente las llamadas a sistema que corresponden a los servicios identificados (puede incluir read, write, Open o close que el sistema haga por usted, no los que usted haya producido directamente con su programa)**
    - mprotect asigna permisos a un espacio de memoria.
    - mmap mapea un área de memoria virtual perteneciente al proceso que lo llama.
    - munmap borra los mappings de cierto rango de direcciones en memoria. Además, hace que llamados que caigan sobre ese rango de direcciones devuelvan un *invalid memory reference*

Ejercicio 3

- **¿Qué ha modificado aquí, la interfaz de llamadas de sistema o el API? Justifique su respuesta.**
- **¿Por qué usamos el número de nuestra llamada de sistema en lugar de su nombre?**
  Dado que no existe nuestra llamada en el API de C.
- **¿Por qué las llamadas de sistema existentes como read o fork se pueden llamar por nombre?**
  Dado que estas están definidas en el API de C, permitiendo llamarlas con tan solo importar la librería correspondiente.