Documentación Tarea 2 - Principios de Sistemas Operativos

Isaac Mena López - 2016130651

March 2019

1 Introducción

1.1 Requerimientos funcionales:

Su programa tendrá la misión de poner a ejecutar a otro programa (digámosle Prog), pasarle los argumentos seleccionados por el usuario y rastrear todos los system calls utilizados por Prog.
En todo caso, al final de la ejecución de Prog, rastreador siempre desplegará en la salida estándar una tabla acumulativa que muestre todos los System Calls utilizados por Prog, ası como el número de veces que fue utilizado cada uno.

1.2 Requerimientos técnicos:

La sintaxis de ejecución desde lınea de comando es:
rastreador [opciones rastreador] Prog [opciones de Prog]

- Las [**opciones rastreador**] podrıan no venir del todo o aparecer en cualquier orden o combinación válida.
- Las [**opciones de Prog**] no serán analizadas ni consideradas por rastreador, sino que simplemente serán pasadas a Prog al iniciar su ejecución.

Las opciones válidas para rastreador son:

- -v desplegará un mensaje cada vez que detecte un System Call de Prog . Se debe desplegar la mayor cantidad posible de detalles respecto a cada System Call.
- -V será idéntico a la opción -v , pero hará una pausa hasta que el usuario presione cualquier tecla para continuar la ejecución de Prog.

El desarrollo se debe de realizar utilizando el lenguaje de programación C para GNU/Linux.

2 Ambiente de desarrollo

Las herramientas utilizadas para la implementación del proyecto fueron las siguientes:

- Laptop Acer-Aspire-RT.
- Sistema Operativo Ubuntu 18.04 LTS.
- Atom 1.29 como editor de texto.
- GCC como compilador de C.

3 Estructuras de datos usadas y funciones

Las 3 estructuras de datos principales del programa son:

- Array de argumentos: en este array se manipulan los argumentos ingresados por el usuario desde que se ingresan hasta que son analizados y separados. Se distribuyen entre PROG y Rastreador.
- Struct de SysCalls: se implementó un módulo aparte para el manejo más ordenado de los Sys Calls. Dentro de éste módulo se incluye un struct llamado *sys_call* el cual almacena la información pertinente respecto a cada syscall como lo es el nombre, el número o id y la cantidad de veces que fue llamada por PROG.
- Array de SysCalls: en este arreglo se manipulan y almacenan todos los structs *sys_call* para poder leerlos y modificarlos de una manera más sencilla.

Las funciones y procedimientos principales del programa son los siguientes:
En rastreador.c

- main: en esta función se realiza la bifurcación de procesos y es donde se identifican las *sys_calls* llamadas por el proceso *hijo*.
- manage_arguments: Este procedimiento identifica los argumentos introducidos por el usuario y establece una serie de banderas dependiendo de la entrada así como completar el array de argumentos de PROG correctamente.
- take_prog_arguments_from_argv: Este procedimiento coloca los argumentos repectivos en el array de argumentos de PROG dependiendo de los operadores de Rastreador (-v, -V o ninguna).
- prog_at_position_has_arguments: determina si PROG tiene arguementos introducidos por el usuario.

- set_null_pointer_at_end: todo arreglo de argumentos que se le pasan a las funciones *exec()* debe terminar con un NULL al final. Este procedimiento se encarga de esa tarea.

En SysCalls.c

- set_syscalls_names: en este procedimiento se establecen los nombres de todos los sys calls, según el repositorio oficial de Linus Torvalds, consultado el 10/03/19.
- manage_arguments: Este procedimiento identifica los argumentos introducidos por el usuario y establece una serie de banderas dependiendo de la entrada así como completar el array de argumentos de PROG correctamente.
- print_sys_call_info: muestra toda la información importante de un struct sys_call (nombre, id, veces llamada).
- initialize_sys_calls_array: creae inicializa todos los structs sys_call y los coloca en el array de syscalls

4 Instrucciones para ejectuar el programa

Entre los archivos se incluye un archivo *makefile* el cual implementa las instrucciones necesarias para ejectuar-compilar el programa.

- Utilizar un sistema operativo linux.
- Colocarse en el directorio donde se encuentren los archivos.
- Abrir una terminal en dicho directorio.
- Si los programas (rastreador.c y SysCalls.c) no están compilado se debe realizar mediante la instrucción *make* la cual utiliza el compilador GCC para compilar todos los archivos correspodientes utilizando el *makefile*.
- Una vez compilados los programas para utilizar el programa se debe seguir la sintaxis especificada en la **Introducción** (rastreador [opciones rastreador] Prog [opciones de Prog]).

5 Actividades realizadas por el estudiante

- 6/03
  - Investigar sobre sys calls en c y como utilizarlas - 1 hr
- 7/03
  - Investigar sobre paso de argumentos en c - 1 hr

  - Investigar sobre el uso de fork() - 1 hr 30 min
  - Esribir ejemplo de uso de exec() y paso de argumentos. - 2 hr
  - Lograr ejecutar un programa del shell, este caso ls con y sin argumentos - 2 hr

- 8/03
  - Investigar sobre ptrace - 2 hr
  - Crear nevo programa-ejemplo para leer y rastrear sys calls usando ptrace (no terminado) - 2 hr
  - Lograr rastrear la primer sys call utilizada por un proceso hijo. - 1 hr 30 min
  - implementar el menejo de argumentos -v y -V - 1 hr

- 9/03
  - Finalizar ejemplo con ptrace - 1 hr
  - Lograr rastrear todos los sys calls de un programa (solo el numero) - 2 hr
  - Investigar sobre la tabla de sys calls - 1 hr
  - Planear el manejo de dicha tabla para lograr hacer el mapeo con el nombre y otros datos - 1 hr 30
  - Implementar módulo de sys calls para faciliar su manejo.(incluye .h y .c) - 2 hr 30 min
  - Inlcuir el modulo de sys calls en el ejemplo original. - 2 hr

- 10/03
  - Escribir la presente documentación. 1 hr 30 min

6 Estado del programa

El estado final del programa fue exitoso. Se logró completar la funcionalidad esperada.

7 Conclusiones

Con esta asignación se logró un mayor comprendimiento del uso de fork() y funciones como ptrace() para detectar sys calls.

Como recomendación puedo decir que lo más importante es saber qué buscar y dónde. A mi parecer lo ideal sería buscar en el siguiente orden:

1. Uso de fork()

2. Buscar en el manual de linux las siguientes instrucciones:

    - wait
    - exit
    - ptrace
    - strace
    - exec

Para cada una de las funciones anteriores tratar de programar un ejemplo en el que se usen 1 o más de estas para lograr entender mejor cómo funcionan en conjunto.
Después de investigado esto es recomendable encontrar la tabla de sys calls adecuada ya que las versiones son muy variadas dependiendo de la versión del kernel. En lo personal yo la encontré en el **repositorio oficial de Linus Torvalds en GitHub**.
También es importante modularizar y separar el código para llevar un mejor orden en especial con el manejo de sys calls.
Por último, una recomendación muy importante. Aunque la asignación se entrega individual no significa que no se puede conversar, pensar y deliverar en conjunto con otros compañeros (ojo que tampoco es pasarse el trabajo ni copiar).

8 Bibliografía

- https://www.linuxjournal.com/article/6100
- http://www.tldp.org/LDP/LG/issue81/sandeep.html
- http://blog.rchapman.org/posts/Linux$_{S}ystem_{C}all_{T}able_{f}or_{x}86_{6}4$/
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall$_{6}4.tbl$
- https://www.poftut.com/trace-system-calls-signals-strace-command-examples/
- https://stackoverflow.com/questions/11081859/how-to-trace-a-process-for-system-calls
- https://www.youtube.com/watch?v=mB79rNrpOhg
- https://stackoverflow.com/questions/23249373/how-to-obtain-linux-syscall-name-from-the-syscall-number
- https://medium.com/@bjammal/process-tracing-system-call-analysis-with-ptrace-685a8844dbfa