Universidad Nacional
Facultad de Ciencias Exactas y Naturales
Escuela de Informática

EIF-212 Sistemas Operativos
Prof. José Pablo Calvo Suárez

I ciclo 2024

Laboratorio #1 – Exploración del espacio de direcciones de un proceso

Introducción

El proceso es la unidad básica de trabajo que utiliza el sistema operativo para la ejecución de los programas. Una vez que el usuario selecciona el programa que desea correr, el sistema operativo debe asignar los recursos necesarios para proporcionar un ambiente seguro y efectivo para que se puedan realizar las tareas respectivas.

En este laboratorio exploraremos el espacio de direcciones de un proceso, así como algunos aspectos importantes para explicar el funcionamiento del sistema operativo desde la perspectiva de la administración de procesos.

Palabras clave: proceso, runtime, espacio de direcciones, registros, stack

Instrucciones

1) Junto con este laboratorio, se le proporciona un archivo con código fuente en C, "*hola.c*". Asegúrese de contar con las herramientas **gcc** y **gdb** instaladas – en la mayoría de las distribuciones Linux estas herramientas ya vienen instaladas.

Nota: Este laboratorio debe realizarlo en una máquina corriendo el kernel de Linux y preferiblemente utilizando **bash**.

Descarguen el archivo y cópienlo en alguna carpeta de su elección.

2) Compilen el archivo, creando un archivo ejecutable llamado hola. Luego de compilar, verifiquen que el código compiló correctamente y luego ejecuten el programa.

user@linux: **gcc hola.c -o hola -Wall**

user@linux: **ls -l**

user@linux: **./hola**

¿Se visualiza en pantalla el resultado que esperaban?

2) ¿Qué es lo que contiene ese archivo ejecutable? Observemos el código x86-64 (también conocido como amd64). Utilicemos el siguiente comando para ver el código con las instrucciones del procesador:

user@linux: **objdump -M intel -D hola | grep -A20 main.:**

user@linux: **ls -l**

user@linux: **./hola**

¿Pueden explicar cómo funciona el '|' en la instrucción anterior? ¿Pueden describir qué hace grep? ¿Qué ocurre si eliminan '**| grep A-20 main.:**' de la instrucción? Finalmente, ¿pueden describir la salida de la instrucción?

2) ¿Qué es lo que contiene ese archivo ejecutable? Observemos el código x86-64 (también conocido como amd64). Utilicemos el siguiente comando para ver el código con las instrucciones del procesador:

user@linux: **objdump -M intel -D hola | grep -A20 main.:**

user@linux: **ls -l**

user@linux: **./hola**

¿Pueden explicar cómo funciona el '|' en la instrucción anterior? ¿Pueden describir qué hace grep? ¿Qué ocurre si eliminan '**| grep A-20 main.:**' de la instrucción? Finalmente, ¿pueden describir la salida al ejecutar esa instrucción y determinar si el código está en 32-bit o 64-bit?

3) Procedamos a utilizar gdb con la siguiente secuencia de comandos:

user@linux: **gdb -q hola**

(gdb) **set disassembly-flavor intel**

(gdb) **break main**

(gdb) **run**

(gdb) **info registers**

(gdb) **quit**

¿Qué es gdb? De la información desplegada después de ejecutar esas instrucciones, ¿pueden encontrar cuáles son los registros de propósito general del CPU? ¿Pueden identificar los registros stack pointer, base pointer, source index y destination index? ¿De qué tamaño son los registros que se observan en la salida? ¿Cuál de esos apuntadores es el que lleva en el CPU la instrucción que se está leyendo en ese momento?

3) Ahora que ya tenemos una mejor idea de la vista de un proceso en ejecución, adentrémonos aún más en los detalles.

user@linux: **gcc -g hola.c -o hola -Wall**

user@linux: **gdb -q hola**

(gdb) **break main**

(gdb) **run**

(gdb) **list**

(gdb) **disassemble main**

(gdb) **info register rip**

¿Pueden interpretar el código en ensamblador y entender el flujo con respecto al programa codificado en C? ¿Qué información contiene el registro rip?

4) Continuando la ejecución anterior, prueben las siguientes instrucciones:

(gdb) **x/4xb $rip**

(gdb) **x/4ub $rip**

(gdb) **x/1xw $rip**

¿Cómo almacena el procesador las variables: little-endian o big endian? ¿Qué tipo de información permiten visualizar los comandos anteriores?

5) Continuando la ejecución anterior, prueben las siguientes instrucciones:

(gdb) **info register $rip**

(gdb) **x/i $rip**

(gdb) **x/3i $rip**

¿Qué información les proporciona la salida de estas instrucciones?

6) Continuando la ejecución anterior, prueben las siguientes instrucciones:

(gdb) **next i**

(gdb) **info register $rip**

(gdb) **x/i $rip**

(gdb) **info register $rbp**

(gdb) **info register $rsp**

¿Qué información les proporciona la salida de estas instrucciones? ¿Podrían completar la ejecución del programa paso a paso? ¿Podrían determinar en cuál segmento de memoria del programa se almacena el código de la función puts, y en cuál dirección? ¿En cuál dirección de memoria está almacenada la variable i, y en cuál segmento de memoria?