Manejo Dinámico de memoria

Laboratorio 2

Astrid Carolina Díaz Gómez

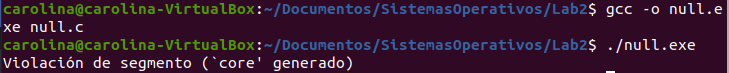
Julio Maria Monterosa Paternina

Respuestas

1. Cuando se ejecuta el programa sale Violación de segmento core debido a que se está accediendo a un espacio de memoria que no está permitido. El puntero no apunta a ninguna dirección (null), al intentar poner un valor en ninguna dirección se obtiene un error.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente



1. Cuando se ejecuta con gdb muestra el error que se presenta un poco más detallada y además información de la línea dentro del código que presenta el error

Texto

Descripción generada automáticamente

1. Cuando se corre el programa se puede ver bastante información, inicialmente sale un aviso que indica que hubo un error de memoria detectado, y sale que hay un escrito inválido en el tamaño de memoria, parece ser porque no se tiene un tamaño de memoria determinado antes de asignar el valor numérico.

Texto

Descripción generada automáticamente

1. Al ejecutar el programa no sale error, sale que se evidencia correctamente, al correrlo con gdb también sale correctamente sin errores, pero al ejecutar el programa con valgrind se puede ver un error de memoria nuevamente, donde indica que al finalizar aun hay uso de memoria que no fue liberada.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. Cuando se corre el programa normal no sale ningún aviso de error, cuando se ejecuta con valgrind sale error al acceder a la posición 100, esto debido a que el programa está accediendo a una posición del array que no existe, ya que este va de 0 a 99 y no existe la posición 100.

Texto

Descripción generada automáticamente

1. Cuando se corre el programa el imprime un dato, pero cuando se corre con valgrind de nuevo sale error de memoria debido a que se está liberando la memoria antes de acceder al dato.

Texto

Descripción generada automáticamente

1. Cuando se ejecuta el programa sale un aviso como si se estuviera liberando memoria de un puntero inválido, al hacer debuger sale el mismo aviso más una información de que el programa fue finalizado (abortado). Al ejecutar el programa con valgrind sale la información de invalid free(), e indica que parte exacta es donde se está generando la falla, ya que la liberación de memoria no es correcta.



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. El código pedido en este punto es igual al visto en clases:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

El programa imprime el numero de puestos reservados en memoria dinámica. Así que para 81 datos la salida es la siguiente:



Reservó 160 puestos de los cuales usa 81, hay memoria reservada sin usar que durante la ejecución del programa queda ocupada. Si el programa durara largo tiempo con solo 81 posiciones usadas sería una perdida de memoria en vano.

Usando el Valgrind:

Texto

Descripción generada automáticamente

No muestra errores porque se hace la liberación de esa memoria. Pero se puede ver la memoria dinámica que usó el proceso: 2284 bytes.

Un programa para comparar con este sería lo mismo, pero usando una lista ligada, así:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Se insertan 81 datos al igual que con el mismo programa que funciona con vectores. Al compilarlo y ejecutarlo en el Valgrind:



Texto

Descripción generada automáticamente

Como no se hace la liberación en ningún momento (que se puede hacer) muestra detalles de la fuga. Pero lo importante en este ejercicio es la comparación de los siguientes datos:

Para el que funciona con lista ligada:

Texto

Descripción generada automáticamente con confianza media

Para el que funciona con vectores:

Texto

Descripción generada automáticamente

El programa que funciona con vectores usa más memoria para los mismos 81 datos, 2284 bytes frente a los 1296 bytes del que funciona con lista ligada. Además de eso. Quizá el más eficiente en tiempo de ejecución es el que usa vectores, porque uso solo 7 bloques en el heap, son accesos a memoria y tienen un costo, en cambio el que usa lista ligada si usa un bloque por cada dato, entonces al final fueron 81 bloques. Quien vaya a usar un programa así debe tener en cuenta qué es más valioso. Si el tiempo o la memoria. o puede hacer un nuevo programa que esté a mitad de camino entre estas dos maneras para llegar a un termino medio. Sería bueno. Este ejercicio es interesante.

1. Cuando se compila el programa salen errores, pero aun asó este crea el archivo .exe, el aviso dice que la función retorna la dirección de una variable local y luego dice que hay una declaración implícita del printf debido a que no se importa stdio, al colocar el include y compilar sale igualmente el primer aviso con el return message.

Con el gdb no sale error específico al correrlo solo String: (null) que es lo mismo que sale con valgrind por lo que no da más información del error con estos comandos y no son necesarios para especificar el error.

El error que sale es debido a que la variable message está creada dentro del bloque del método getString y al ser una variable local su ciclo de vida es solo dentro de este método, cuando se usa o solicita fuera del bloque ya esta variable no está en memoria.

Texto

Descripción generada automáticamente

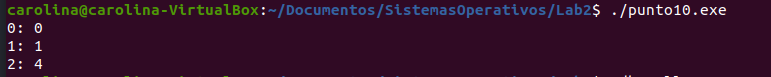
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. No se encuentra error con GDB, cuando se revisa con valgrind sale varios errores, primero sale una escritura válida de 4 bytes, luego más adelante indica que el for o ciclo depende te un valor no inicializado, este sale en varias ocasiones, otro error que se puede ver es el uso de un valor que no está inicializado. Se puede ver que es una fuga de memoria y no de código, por lo cual valgrind sirve para identificar el error.



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. En este caso salen varios errores, pero el principal que se debe tener el cuenta es la liberación de memoria dentro del for, lo que hace que haya liberación de memoria dos veces, y cuando se libera en la primera ocasión y entra de nuevo al for como ya se liberó heights se está haciendo un acceso a un puntero que se liberó de la memoria. Cuando se ejecuta valgrind con comando simple sale varios errores, entre ellos los dichos anteriormente.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. Cuando se ejecuta el gdb se puede ver error, pero no se especifica igualmente con valgrind con el comando que se ha usado.

Usando el gdb uno se da cuenta que nunca entra dentro del if de la línea 9:

Texto

Descripción generada automáticamente

Entonces se corrige el error de sintaxis



El compilador ya no dice nada y no hay ninguna salida:



Por lo tanto, no se reserva más memoria.

Al final hace un free y se libera la memoria. No hay ningún error por ahora.

Usando el valgrind:

Texto

Descripción generada automáticamente

Tampoco reporta ningún error.

1. –
2. Se seleccionó helgrind que es un detector de condiciones de competencia de datos que utilicen hilos

Las opciones que tiene son

* --free.is-writen=no|yes [default: no]= Este sirve para tratar la liberación de memoria dinámica que fue escrita inmediatamente antes de liberarla
* --track-lockorders=no|yes [default: yes] = Cuando está habilitado por defecto, helgrind mejora la categorización del orden de bloqueos. Para algunos programas que presentan bugs, el máximo numero de orden de bloqueos reportados pueden ser molestos, particularmente si solo se está interesado en correr errores.
* --check-stack-refs=no|yes [default: yes] = Por defecto Helgrind revisa todos los accesos de datos de memoria realizados en el programa. Esta bandera permite saltar los accesos a trabes de la pila de variables locales. Esto puede mejorar el rendimiento pero tiene el costo de perder los registros ubicados en la pila estática.
* --conflict-cache-size=N [default: 1000000] = Esta bandera solo tiene ujn efecto en --history-level=full. La información sobre antiguos accesos conflictivos es almacenada en una caché de almacenamiento limitado. Esto es necesario porque no es particular almacenar un registro de pila para cada único acceso de memoria hecho por el programa. La información histórica a la que no se haya tenido acceso últimamente es descartada de manera periódica, para liberar espacio en la caché.

Esta herramienta detecta tres tipos de errores:

* El mal uso del API POSIX
* Potenciales interbloqueos
* Condiciones de carrera

Texto

Descripción generada automáticamente

Se crea un programa donde los hilos comparten información sobre una variable global counter, esto provoca que los hilos lean y escriban la variable sin importar el orden, creando una condición de carrera como lo muestra al ejecutar el Helgrind.