1.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Resultado:

Texto

Descripción generada automáticamente

El puntero no apunta a ninguna dirección (null), al intentar poner un valor en ninguna dirección se obtiene un error.

2.

Texto

Descripción generada automáticamente

Gdb muestra la línea de código que ocasionó el segmentation fault. Y una dirección que creo que es la dirección en memoria de esa línea de código en el segmento code del address space de ese proceso.

3.

Al usar Valgrind…

Texto

Descripción generada automáticamente

Dice: Escritura invalida de tamaño 4 (¿bytes por tratarse de un int?) dentro de la función main en la línea 9 del código. Luego da un detalle del error como la dirección a la que se intentó escribir y dice que no está en la pila ni en el heap ni fue liberada recientemente.

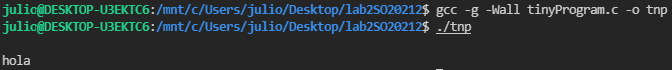
Se perdieron 0 bytes porque no se usó memoria dinámica y todos los bloques del heap están libres para ese proceso y no es posible una fuga.

4.

Texto

Descripción generada automáticamente

El programa se ejecuta sin problemas



Como el programa no tiene ningún problema el uso del GDB no aplica para encontrar el problema. En este caso sabemos que hay un problema y no de compilación y con el GDB podríamos ver el valor de las variables nada más.

Con Valgrind si se puede detectar la fuga de memoria implantada ahí puesto que justo nos muestra esa información:

Texto

Descripción generada automáticamente

Nos dice que se perdieron 10 bytes en una función malloc de la librería dentro del main, y ese malloc está en la línea 8.

5.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Se ejecuta el programa



No pasa nada!. Esperaba un segmentation fault o algún error.

Al ejecutarlo con el Valgrind:

Texto

Descripción generada automáticamente

Ahí si me dice que hay una escritura invalida de 4 bytes por la línea 9 del main, y que se perdieron 400 bytes reservados con un malloc en la línea 8 del main.

El programa no es correcto porque se está intentando acceder a la posición 100 de un array que solo llega hasta la posición 99. Creo que no lanzó un segmentation fault porque la memoria que está después de la posición 99 del array data, sigue siendo parte del address space de ese mismo proceso, porque cuando uno intenta acceder a la posición 100 mil de ese vector ahí si salta un segmentation fault.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

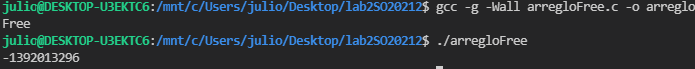


6.

Texto

Descripción generada automáticamente

Al ejecutar el programa:



El programa si corre e imprime un número que no tiene sentido. Es la basura que había ahí en esa posición de memoria.

Al usar el Valgrind:

Texto

Descripción generada automáticamente

Informa sobre la lectura invalida de 4 bytes en la línea 10 del main y que está accediendo a 8 bytes del comienzo de un bloque de 400 bytes liberados. No hay fuga de memoria, pero si presenta informacion sobre el heap.

7.

Texto

Descripción generada automáticamente

Al ejecutar el programa:

Texto

Descripción generada automáticamente

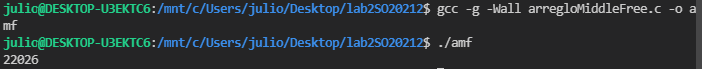
De primeras ya hay un error evidente en la ejecución: hubo un puntero invalido en la función free.

Supongo que free necesita el puntero al primer elemento del ‘objeto’ que se quiere liberar porque cuando ejecuto lo siguiente:

Texto

Descripción generada automáticamente

Ya no aparece ese error:



Entonces no necesito herramientas como Valgrind para detectar el error.

8.

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.

9.

Texto

Descripción generada automáticamente

Al ejecutar este código:

Texto

Descripción generada automáticamente

El compilador ya me dice que se está retornando una variable local (en el stack). Constantemente el stack está creciendo y decreciendo en cuanto se usan las funciones, es peligroso.

El gdb no dice mucho:

Texto

Descripción generada automáticamente

Y el Valgrind…

Texto

Descripción generada automáticamente

Tampoco reporta errores.

Ahí sirvió más el compilador.

10.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

El error aquí es que solo se reservaron 3 bytes para 3 enteros que ocupan 4 bytes cada uno. Aún así el programa funciona:

Texto

Descripción generada automáticamente

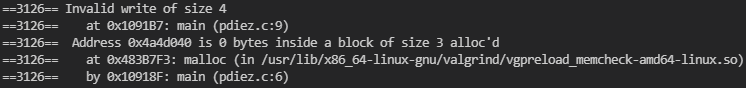
Pensé que quizá los números que se guardan en las posiciones del vector como no se salen del rando de 1 byte, se pueden guardar los 3 solo en 1 byte cada uno. Pero luego con el GDB uno se da cuenta que en realidad están ocupando 4 bytes cada uno:

Texto

Descripción generada automáticamente

Así que no me dice nada más.

Utilizando el Valgrind salen, entre otros, estos errores:



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Ahí si marca el error que es que se quiere colocar al principio de un espacio de 3 bytes, un dato de 4 bytes y luego que se quiere leer un dato de 4 bytes en la reserva de 3 bytes. Y que se pierden 3 bytes porque no se liberaron.

11.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

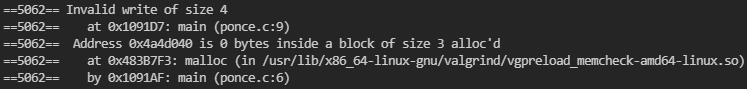
Ejecutando el programa:

Texto

Descripción generada automáticamente

El error es que se está reservando solo 3 bytes para 3 enteros, además también se está liberando memoria a la primera que entre en el bucle for habiendo más iteraciones por atender, además en la segunda iteración, se está accediendo a un espacio de memoria liberado que pudo haber sido sobrescrito porque esa memoria quedó libre, no fue el caso, pero pudo ser y pudo imprimir basura.

Usando el Valgrind:



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

El Valgrind avisa sobre la memoria insuficiente para leer o escribir un int en ese espacio de 3 bytes tal como en los ejemplos anteriores. Además, notifica sobre un free inválido, pero nada más acerca del free más que la memoria donde va a liberar. También que no hay fugas de memoria.

12.

Texto

Descripción generada automáticamente

De primeras hay un error de sintaxis que el compilador identifica, al ejecutar el programa, no hay ninguna salida:

Texto

Descripción generada automáticamente

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:



Usando el gdb, se muestra que tampoco entra dentro del if:

Texto

Descripción generada automáticamente

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.

13.

Al ejecutar en el Valgrind ”./psinfo -l <pid> …. <pid>” muestra los errores siguientes

Texto

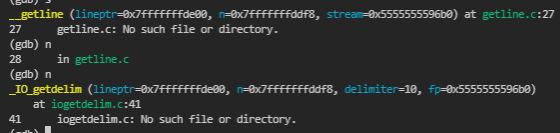
Descripción generada automáticamente

El programa reserva 120 bytes de espacio de memoria por cada proceso valido que consulte. Al consultar dos procesos reserva 240 bytes, en la imagen de ejemplo se consultaron 3 procesos, el error se origina cuando se usa el método ‘pillarDatos’ así que cuando se le pasa un PID invalido solo consulta los pid validos que estén antes y solo se reserva la memoria de esos procesos.

Los errores no corresponden a mi programación si no a las librerías de C que uso, puesto que en el código nunca se escribe un malloc, así que no puedo solucionarlos.

El método getdelim tampoco lo uso directamente En el programa aun así se está haciendo llamados a ese método.

Usando el gdb para identificar cuando se llama a tal método:



El método se llama dentro del código del getline que se encarga de leer línea a línea de los archivos a consultar.

Ejecutando en el Valgrind ”./psinfo -r <pid> …. <pid>” pasa exactamente lo mismo:

Texto

Descripción generada automáticamente

Y por ultimo la ruta de ejecución más sencilla ‘./psinfo <pid>’:

Texto

Descripción generada automáticamente

Ocurre lo mismo.

Este laboratorio que hice tiene el problema que no tiene mucho que arreglar puesto que no usé mem. Dinámica. Por eso propongo hacerle seguimiento a el mismo lab desarrollado desde cero esta vez si usando memoria dinámica.

Texto

Descripción generada automáticamente

Dice que aún hay memoria rescatable, cosa que no pasa con el lab1 original. Al pedirle que muestre detalles con

--show-reachable=yes

Texto

Descripción generada automáticamente

Ahí si uno puede ir a arreglar los errores ya identificados.

Para arreglar el programa hay que hacer la liberación de la memoria dinámica luego de usar malloc o realloc. En este caso se usa para guardar la ruta del vector y el vector mismo. Por eso la solución es liberar el espacio de esas variables una vez usadas.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

También hacer lo mismo en la función imprimir que es donde los datos ya dejan de servir puesto que ya se les imprime. A esa función se le pasa todo el array de arrays que contienen los datos. El array fue ubicado con un malloc y también cada una de sus posiciones. Por eso hay que liberar cada una de las posiciones y luego el puntero de punteros en si.

Texto

Descripción generada automáticamente

Compilándolo y ejecutando en el Valgrind:

Texto

Descripción generada automáticamente

Ya no hay fugas 😊

Por lo menos del propio programa.

14.