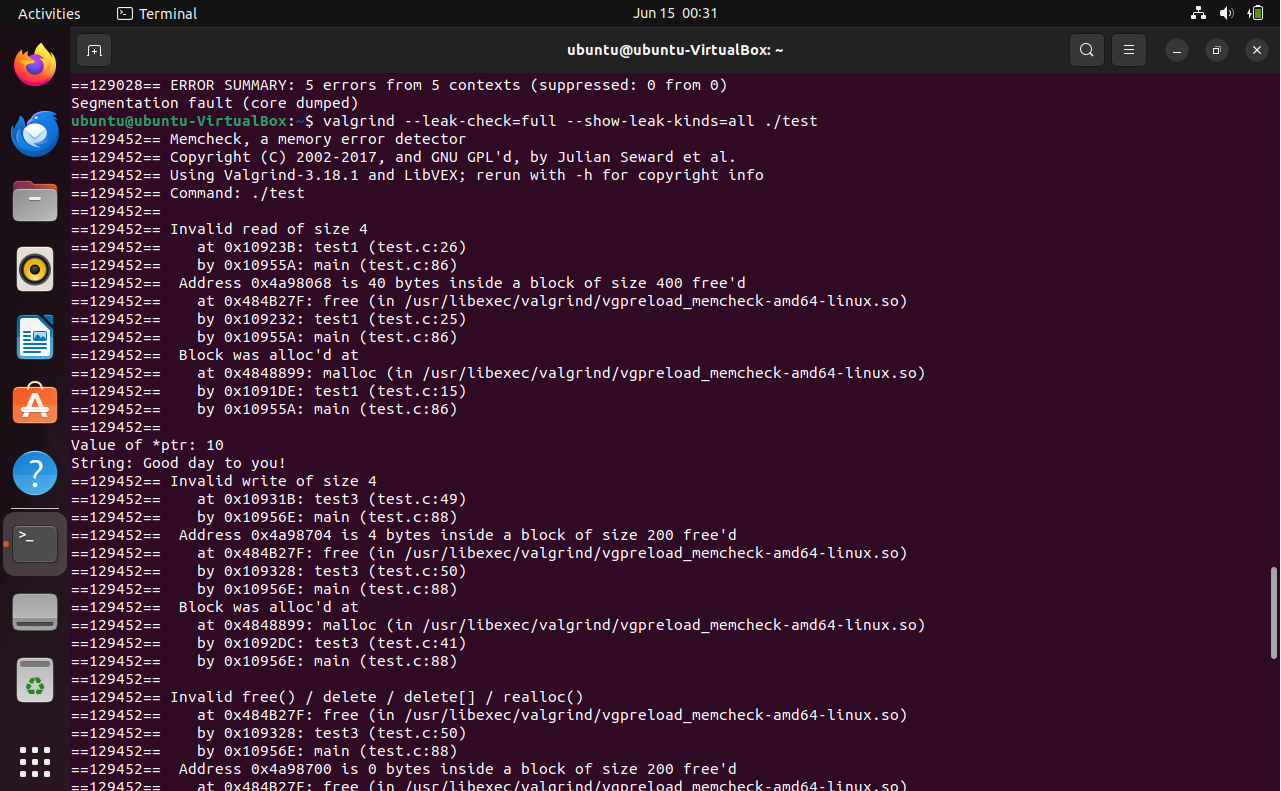
Module 4 – Assessment

**Code Debugging Tools (GBD and Valgrind)**

1. Using Valgrind identify memleaks in the given program. Explore optional flags in Valgrind.

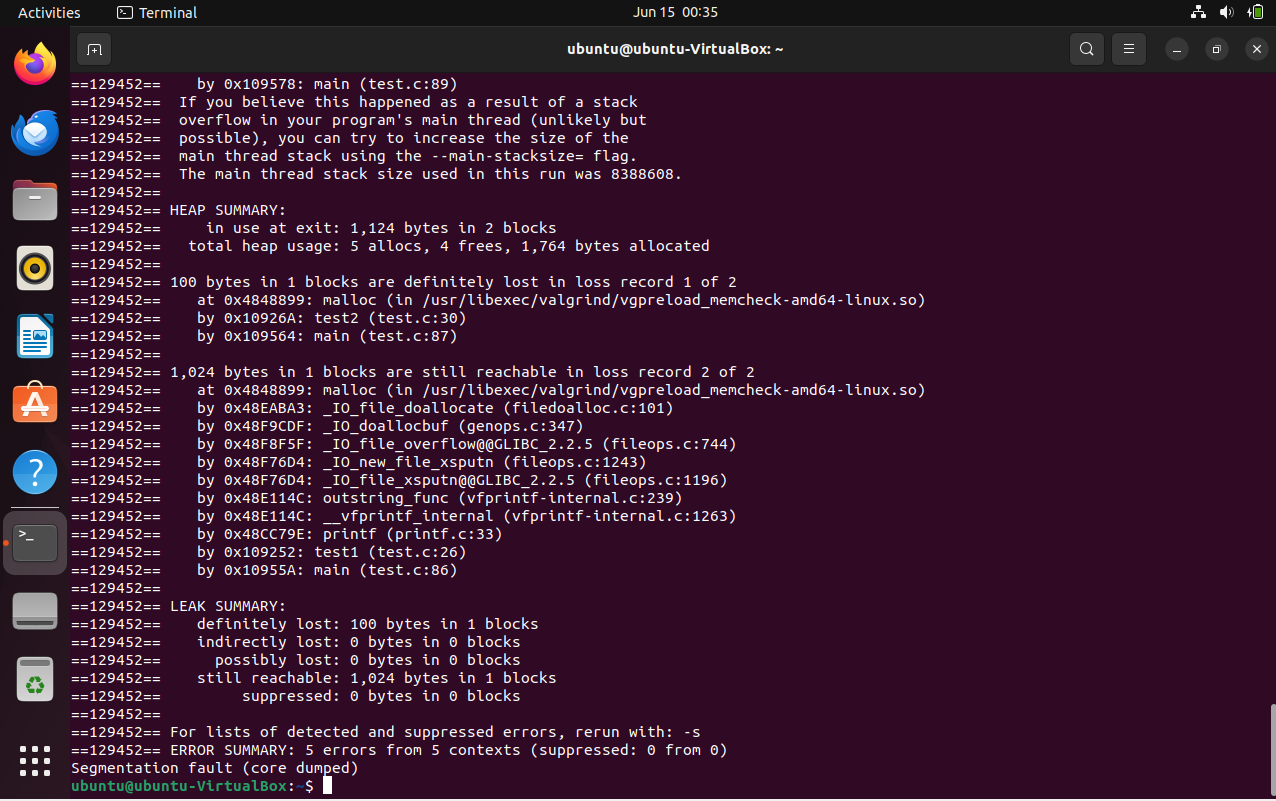
Solution:

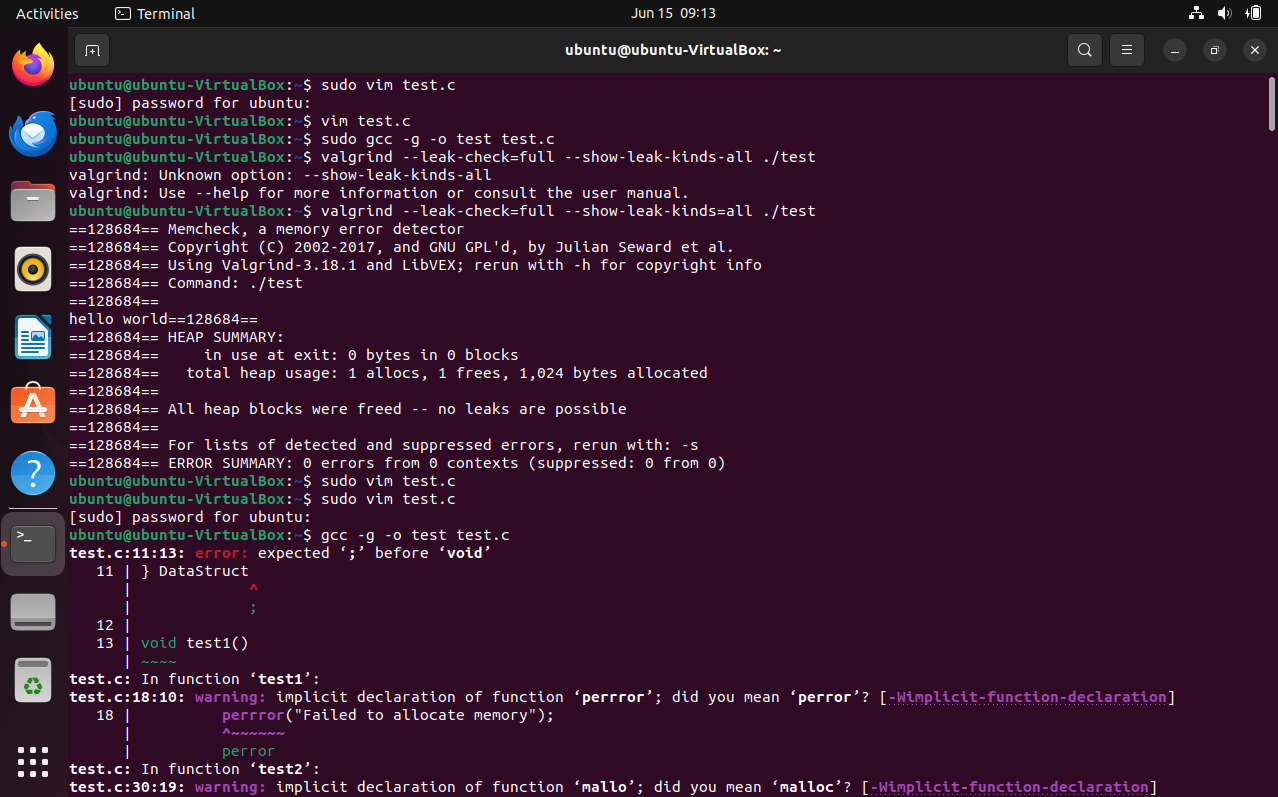
Step 1: Create a file using sudo vim test.c



A screenshot of a computer program

Description automatically generated





2) With the same program, using GDB, set breakpoints, run the program, list the code, run from one breakpoint to another, print the value of variables while execution, check assemble code, disable breakpoints, check registers info, explore optional flags.

To build the program in Linux - "gcc -g -o test test.c"

**Program** :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define NUM\_STRUCTS 50

typedef struct {

char \*name;

int id;

int \*values;

} DataStruct;

void test1() {

int \*ptr = malloc(sizeof(int) \* 100);

if (ptr == NULL) {

perror("Failed to allocate memory");

return;

}

for (int i = 0; i < 100; i++) {

ptr[i] = i;

}

free(ptr);

printf("Value of \*ptr: %d\n", ptr[10]);

}

void test2() {

char \*str = malloc(100 \* sizeof(char));

if (str == NULL) {

perror("Failed to allocate memory");

return;

}

strcpy(str, "Good day to you!");

printf("String: %s\n", str);

}

void test3() {

int \*ptr = malloc(sizeof(int) \* 50);

if (ptr == NULL) {

perror("Failed to allocate memory");

return;

}

for (int i = 0; i < 2; i++) {

ptr[i] = i \* 2;

free(ptr);

}

}

void test4() {

int \*ptr = malloc(sizeof(int) \* 10);

if (ptr == NULL) {

perror("Failed to allocate memory");

return;

}

for (int i = 0; i < 10; i++) {

ptr[i] = i \* 3;

}

free(ptr);

ptr = NULL;

printf("Value of \*ptr: %d\n", \*ptr);

}

void test5() {

DataStruct \*data = malloc(NUM\_STRUCTS \* sizeof(DataStruct));

for (int i = 0; i < NUM\_STRUCTS; i++) {

data[i].name = malloc(50 \* sizeof(char));

strcpy(data[i].name, "Example Name");

data[i].id = rand() % 1000;

data[i].values = malloc(10 \* sizeof(int));

for (int j = 0; j < 10; j++) {

data[i].values[j] = rand() % 100;

}

}

}

int main() {

test1();

test2();

test3();

test4();

test5();

return 0;

}

Solution:

Commands:

gcc -g -o test test.c

gdb ./test

break test1

break test2

break test3

break test4

break test5

run

list

continue

print ptr

print \*ptr

disassemble

disable 1

disable 2

info registers

next

step

bt

OUTPUTS:

