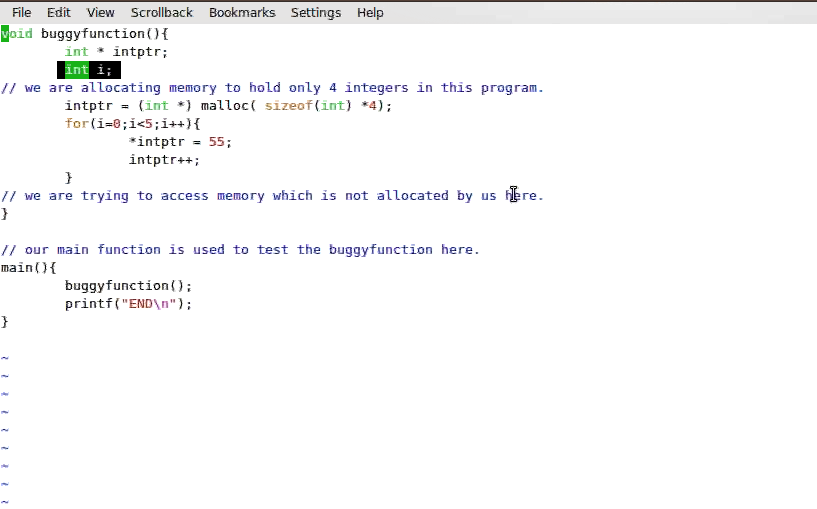
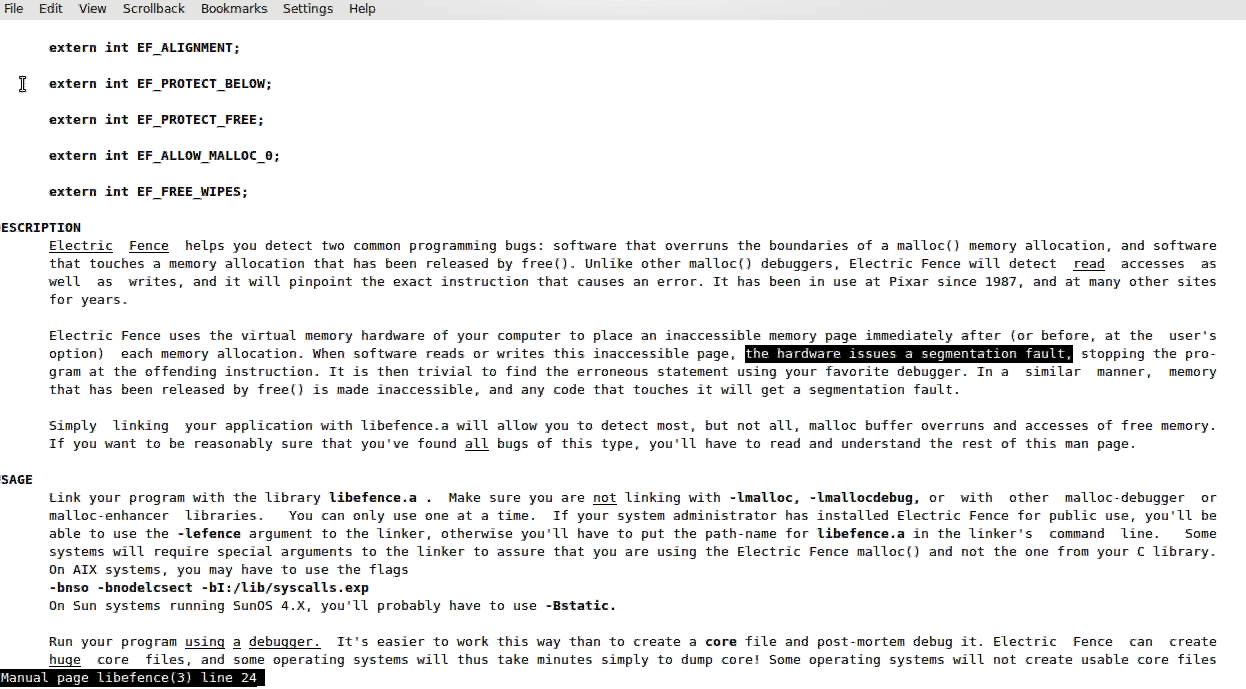
Declare the variable or scope of a variable within the function. Static variable is through the life time of a function or may be program. The variable decides by memory.



Electric Fence uses the virtual memory hardware of your computer to place an in accessible memory page immediately after (or before at the user’s options) each Memory location

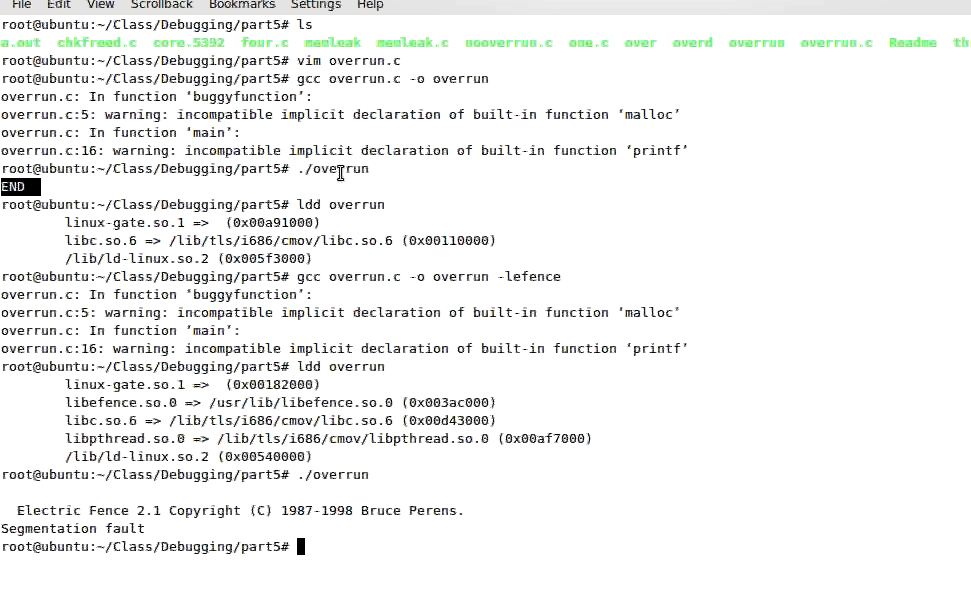
These memory is added to the processor.The processor has a process ID. The memory location which is loaded is known as process address space. Demand for memory during running allocated memory is also added front process address space.This memory could be in two blocks,When we normally allocate memory.



Extern int EF\_PROTECT\_BELOW;

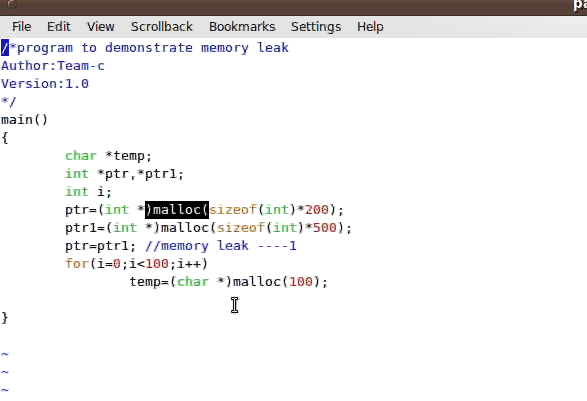
Observation of this program will take now when we recompile with new library -o overrun which we link a library -lefence.

When we compiling at first we successfully return the result END. When buggy function returns on lefence library the segmentation volation has been with this tool I can understand violating some kind of memory when I come to know this I need to take correct so that I won’t repeat this mistake violating any memory and performs these job.

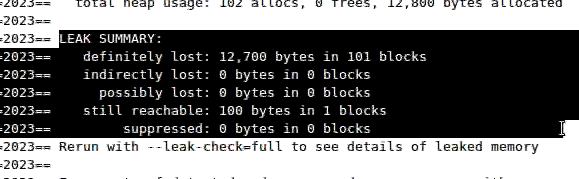


When lefence library was checking these function and based on the switch EF\_PROTECT\_BELOW. Now set to zero overall condition EF\_PROTECT\_BELOW. A small drawback on lefence and check overall one other time this is managed by a variable EF\_PROTECT\_BELOW.

In built of this valgrind the total memory of allocation is 12,700 bytes in 101 blocks by this program summary of this leak. We need to see where exactly how of the bytes are loss –leak-check=full



What has been the memory that the application has been return and trying to allocate the memory and the integer another 500 integers or 1000 charecters all these memory has been allocated and terminates his memory. In execution or would be execution and this memory returns to your application respecting that this memory is used and that block of memory is associated with your application has been used and to be released them application is respected to memory.



Based on the GBD debugger they have a few features GDB debugger has been provided some functionalities basic fuctionalities that has been.

* Start your program. Specifying that might affected its behaviour.
* Make your program start on specified conditions.
* Examine what has happened, when your program has stopped.
* Change things in your program, so you can expereriment with correcting the effects of one bug and go on to learn an other

GDB is invoked with the shell command **gdb**. Once started, it reads commands from the terminals until you tell it to exit with the GDB Command quit.

You can run GDB with no arguments or options; but the most usual way to start gdb is with one argument or two, specifying an executable program has the argument:

Gdb program

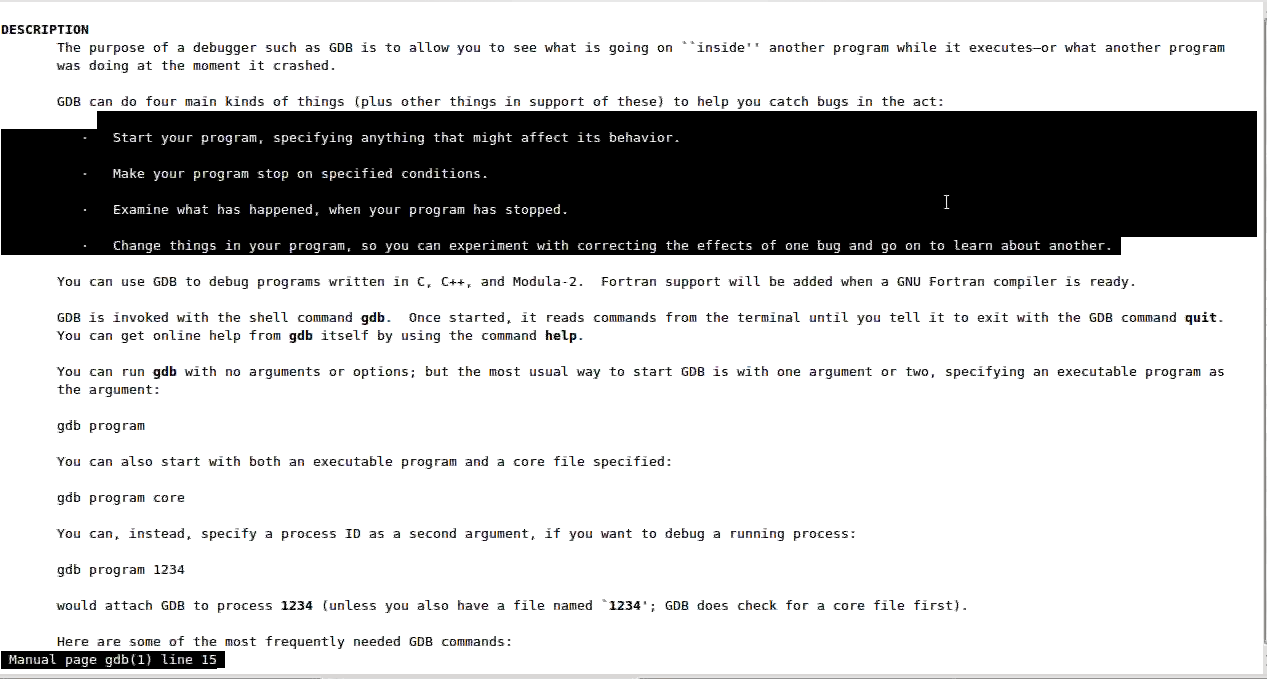
We can also start with both an executable program and a core file specified:

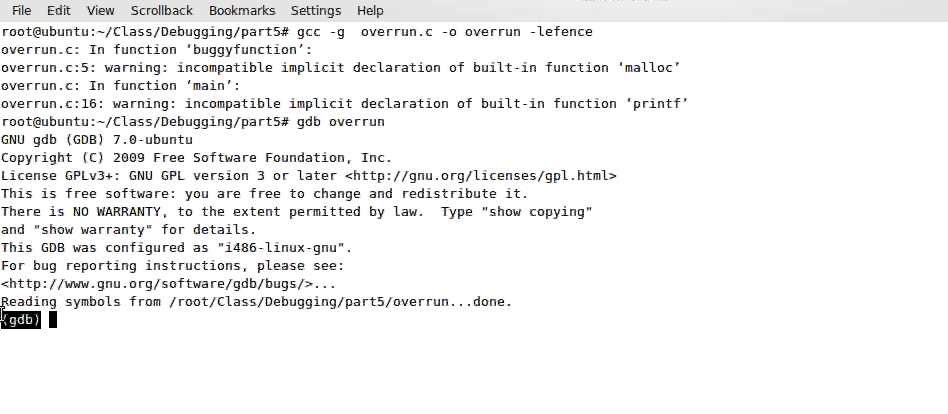
Gdb program core

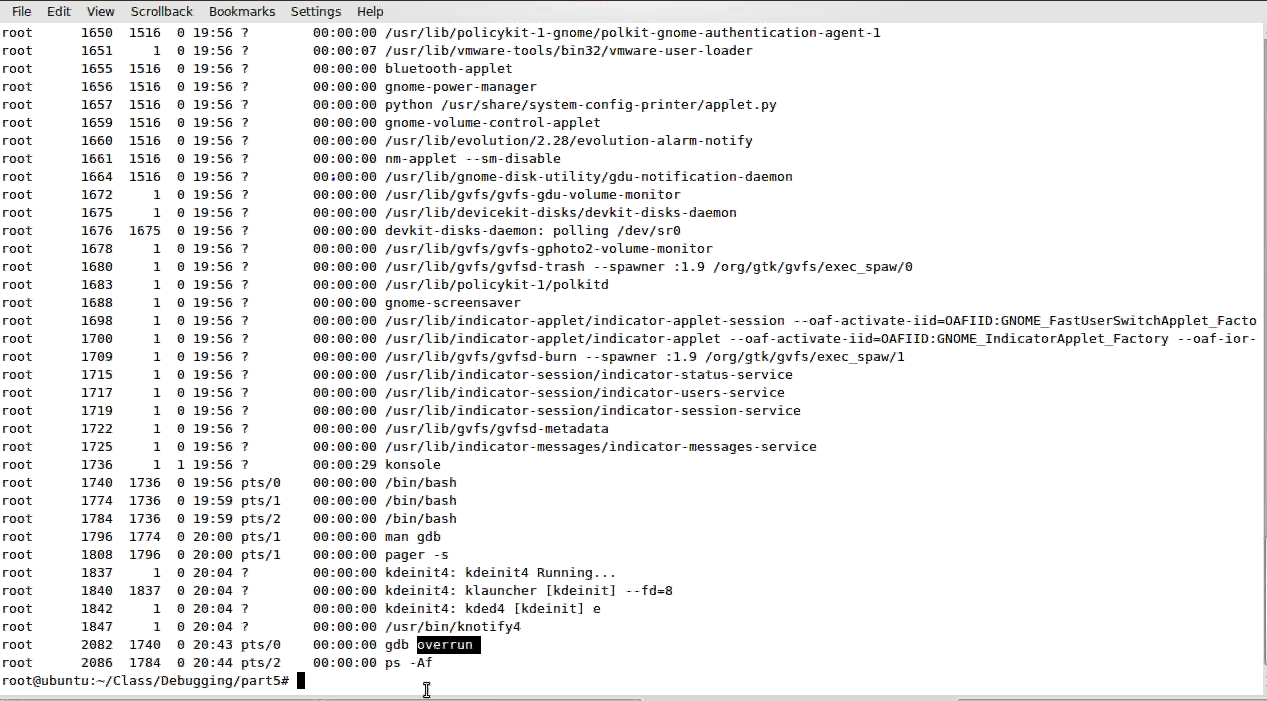
You can, instead, specify a process ID as a second argument, if you want to debug a running process:

Gdb program 1234

Would attach gdb to process 1234. (unless you also have a file named ‘1234’;gdb does check for a core file first:

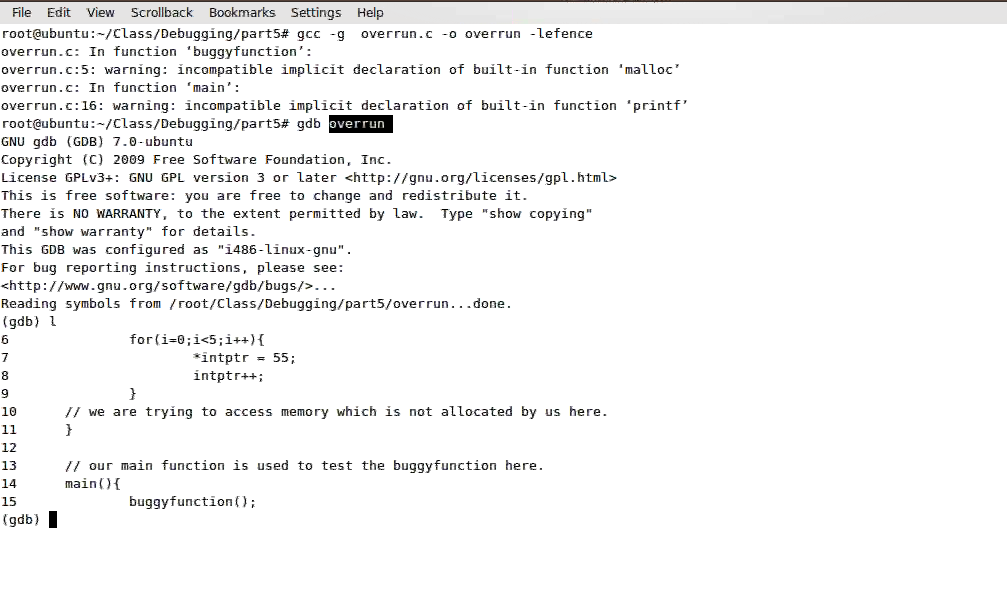




The benefits of the console gdb over which overrun in such cases. 

In 2082 gdb has overrun application by PID of application ready to be started once an application on gdb lot of information of the application symbols information to list the program command list or -l.

It will list out all the below program:



Reading symbols from /root/Class/debugging/part 5/overrun….done.

-g does not returns the reading symbols if it could not be there.

Putting variable and values together called as symbols.