Name – Indranil Bain
DEPT. – Computer Science & Technology
Enrolment No. – 2020CSB039
Group – GX (6)
Subject – Software Engineering

What is a GNU debugger?

- The GNU Debugger (GDB) is a popular debugger used on Linux to debug programs written in C, C++, Ada, and other languages. It allows you to inspect the state of a program while it is running or stopped and provides information on what is causing the program to malfunction. GDB can be used to find and fix bugs, monitor the execution of a program, and analyze core dumps. It supports multi-threaded and multi-process programs and can be used both from the command line and from integrated development environments (IDEs).

- GDB is often used by developers to debug programs running on a local machine, but it can also be used to debug programs running on remote systems, such as embedded devices or servers. In addition, GDB can be used to debug core files, which are generated when a program crashes, in order to determine the cause of the crash.

- One of the most useful features of GDB is its ability to interact with the program being debugged in real-time. This allows you to modify variables, set breakpoints, and step through code in a controlled and reproducible way. GDB also provides a graphical user interface called GDB TUI (Text User Interface), which allows you to see the source code, disassembly, and register contents all in one window.

Commands for GNU Debugger

1. Running a Program:

- When we want to debug a program, we have to first make its executable format.

  Example: If we want to debug a c++ file, then first we have to compile it and make the .exe file from that.

  Then to run a program we have to use start the debugging using "gdb <sample program name>".
  Then we have to use the "run" command.

2. Loading Symbol Table:

- **Symbol Table:**
  The symbol table is a data structure used by compilers and linkers to store information about the symbols defined in a program. A symbol can be a variable, a function, a constant, or any other named entity defined in the code.

  The symbol table typically contains information such as the name of the symbol, its type, its location in memory (e.g. address or offset), and other relevant information such as its size, scope, and visibility. The symbol table is used by the linker to resolve references to symbols, for example, when a function defined in one module is called from another module. The symbol table is also used by the debugger to provide information about symbols, such as the value of a variable or the address of a function.

  Command to Load Symbol Table:
  We can load the symbol table using "file <program Name>" command.

3. Setting Break Point:

- **Break Point:**
  A breakpoint in a debugger is a point in the code where the execution of the program is temporarily suspended. When a breakpoint is reached, the debugger provides an interface to inspect the state of the program, including the values of variables, the contents of memory, and the call stack. Breakpoints are useful for identifying the source of bugs and for verifying that the program is executing as expected.

  We can set breakpoint according to the line number of our program or we can also give a function name, to suspend the execution of the program when it reaches that specific function.

  Commands:

  "break <Line Number>"
  "break <Function Name>"

4. Listing variables and examining their values:

- In GDB, we can list variables and examine their values using the following commands:

  info locals: displays a list of local variables in the current function and their values.

  info args: displays a list of arguments passed to the current function, along with their values.

  print <variable>: prints the value of the specified variable.

  display <variable>: sets a display for the specified variable, so that its value is automatically printed each time the program stops.

5. Printing content of an array or contiguous memory:

6. Printing function arguments

- In GDB, we can print the arguments of a function by using the info args command while stopped at a breakpoint within the function. The "info args" command displays a list of the arguments and their values that were passed to the current function.

7. Next, Continue, Set command:

- **Next :**

- **Continue:**

  The continue command in GDB is used to resume the execution of the program after a breakpoint. When the program reaches a breakpoint, it stops and waits for you to take further action. You can then use the continue command to continue the execution of the program until it either terminates or reaches another breakpoint.

- **SET :**

  The set command in GDB is used to set various options and parameters in GDB. For example, you can use the set command to enable or disable printing of the source code with set print source on or set print source off. You can also use the set command to set the prompt in GDB with set prompt <prompt-string>

Single stepping into a function:

8. Listing all breakpoints:

- To check all the breakpoints we can use "info breakpoints"

9. Ignoring a break-point for N occurrence:

In GDB, you can ignore a breakpoint for a specified number of occurrences using the ignore command.

Commands:

Ignore <break point Number> <count>

10. Enable/disable a breakpoint:

'disable [breakpoint_number]' : disables the specified breakpoint. If the breakpoint number is not specified, all breakpoints are disabled.

'enable [breakpoint_number]': enables the specified breakpoint. If the breakpoint number is not specified, all breakpoints are enabled.

11. Break condition and Command:

We can specify some conditions when we give a break statement. In GDB, you can set a conditional breakpoint that only stops the program when a certain condition is met. To set a conditional breakpoint, use the break command followed by the line number or function name, and specify the condition using the if option.

Command: "break 8 if i==5"

Examining stack trace:

13. Examining Stack Trace:

A stack trace, also known as a call stack or a backtrace, is a record of the function calls that are executed to reach the current point in the program's execution. The stack trace is organized as a series of nested function calls, with the most recently called function at the top and the original function call at the bottom.

Command: "bt"

Examining stack trace for multi-threaded program:

14. Core file debugging:

Core file debugging is the process of analyzing a core dump to determine the root cause of a crash or malfunction in a software application. A core dump is a file generated by an operating system when a program crashes, which contains a snapshot of the program's state and memory contents at the time of the crash.

The process of core file debugging involves the following steps:

Obtaining the core dump file: This can be done using the operating system's core dump generation mechanism or by using tools like GDB (GNU Debugger) or LLDB (Low-Level Debugger).

Analysing the core dump file: This involves using tools like GDB or LLDB to examine the contents of the core dump and identify the cause of the crash.

Identifying the problem: Once the cause of the crash has been identified, it is important to determine the exact line of code that caused the crash and any relevant variables or parameters.

Debugging and fixing the issue: Based on the information gathered from the core dump analysis, the problem can be fixed by making changes to the code and retesting the application.

Core file debugging is an important tool for software developers to troubleshoot crashes and improve the reliability and stability of their applications.

15. Debugging of an already running program:

Debugging of an already running program refers to the process of analysing and fixing errors in a program that is already running on a computer. This type of debugging is typically performed using a debugger tool that allows developers to interact with the running program and observe its behaviour as it executes.

The steps involved in debugging a running program are as follows:

Attaching the debugger to the running program: This involves using the debugger tool to connect to the running program and start observing its behaviour.

Setting breakpoints: Breakpoints are points in the code where the debugger will stop execution and allow the developer to inspect the program's state and variables.

Stepping through the code: The debugger allows the developer to step through the code one line at a time, observing the values of variables and the behaviour of the program as it executes.

Debugging: Based on the information gathered while stepping through the code, the developer can identify and fix any errors or bugs in the program.

Resuming execution: Once the bugs have been fixed, the developer can continue execution of the program and observe its behaviour to ensure that the changes have had the desired effect.

Debugging of a running program is an important tool for software developers to troubleshoot and fix issues with their applications. It allows them to observe the behaviour of the program in real-time, identify the root cause of any problems, and make changes to the code to fix the bugs.

-:-