2021CSB001	2021CSB004	2021CSB006	2021CSB007
ANISH BANERJEE	DEVIKA KANCHAN SIMLAI	SUDIP DATTA	KARRA ASHISH REDDY

SOFTWARE ENGINEERING LABORATORY

ASSIGNMENT - 2

SHORT NOTE ON GDB DEBUGGER

- **GNU Debugger, also called “gdb" is a command line debugger primarily used in UNIX systems to debug programs of many languages like C, C++, etc.**
- **It was written by *Richard Stallman* in 1986 as a part of this GNU System, and since then it has been widely used as a go-to debugger till today.**
- **It provides a lot of features to help debug a program, which includes monitoring and modifying the values of internal variables at run time, calling functions independent of the actual program behaviour, and much more.**


RICHARD STALLMAN

HOW TO USE GDB DEBUGGER?

- These steps are to be followed:

INSTALLATION

- Open Terminal
- For Linux: “sudo apt install gdb”
- For Mac: “brew install gdb”

RUNNING THE PROGRAM USING DEBUGGER

- Generally, we perform “gcc hello.c -o hello”
- But to generate GDB accessible program, we will have to use “-g” flag
- “-g” flag is used to signal gcc to also generate a symbol table for gdb to read
- So “gcc hello.c -o hello -g”
- Now we just have to do “gdb hello” to start the debugger.

```c
// Hello.c
#include <stdio.h>

int main(){
    int value = 3;
    for (int i = 0; i < 10; i++){
        if (i == value){
            printf("Target Reached, breaking...\n");
            break;
        }
        printf("Hello World\n");
    }
}
```

PART - 1

RUNNING A PROGRAM INSIDE GDB

- Now we are inside gdb, to just run the program, we can do the following
- 1. “r” to run the program
- 2. “r 1 2” to pass command-line arguments to the program
- 3. “r < file1” to feed a file for writing the outputs

PART - 2

LOADING SYMBOL TABLE

- Symbol table stores names of variables, functions and types defined within our program

```
main()
{
char a[5];
int x;
.
.
```

SAMPLE PROGRAM

FORMAT OF SYMBOL TABLE

Name	Type	Size	Dimension	Line of Declaration	Line of Usage	Address

SYMBOL TABLE CORRESPONDING TO SAMPLE PROGRAM *(LEFT)*

Name	Type	Size	Dimension	Line of Declaration	Line of Usage	Address
x	int	4	0	4		
a	char	5	1	3		

SETTING A BREAKPOINT

OVERVIEW

- 1. There are many ways to setup Break Points in the GNU GDB
- 2. Two majorly used ones include:
- Setting a breakpoint at the start of a function()

```
break /*{function name}*/
```

- Setting a breakpoint at the start of a function()

```
break /*{line number}*/
```

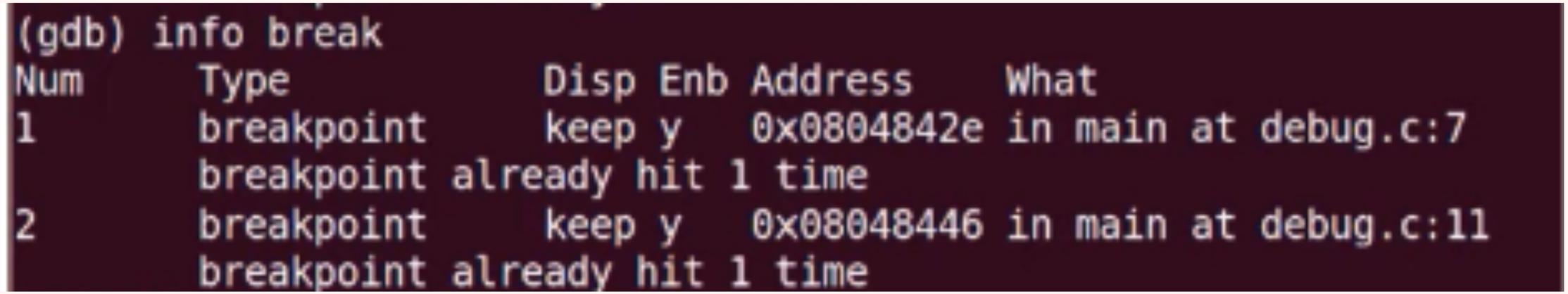
```
(gdb) break 7
Breakpoint 1 at 0x804842e: file debug.c, line 7.
(gdb) run
Starting program: /home/user/ctest/debug

Breakpoint 1, main () at debug.c:7
7                int b = 1;
(gdb) info locals
a = 0
b = -1207963648
c = 134513787
d = -1208221696
(gdb)
```

- This command will add a break point at the line number we specify.
- As soon as the execution reaches our line number, it will halt the execution process
- 3. After a Break Point is reached, we can view the state of every variable, function and the call stack up until that point.

PART - 3

- 4. The command “info locals” will show us the state of the local variables at out break point.
- 5. The command “info break” will show us all the break points currently held within our program
- 6. The command “continue” will resume the execution which was previously halted after reaching a break point.
- 7. The command “delete /*{breakpoint number}*/ will delete a breakpoint.



LISTING VARIABLES

OVERVIEW

- The info locals command displays the local variable values in the current frame. You can select frames using the frame, up and down
- Note that the info locals command does not display the information about the function arguments.
- Use the info args command to list function arguments.
- We will run the program, set a breakpoint in func() and use the info locals command to display the local variables in main():

```
#include <stdio.h>

void func(int arg)
{
    printf("func(%d)\n", arg);
}

int main(int argc, char *argv[])
{
    int localVar1 = 1, localVar2 = 2;
    func(localVar1 + localVar2);
    return 0;
}
```

OVERVIEW - PARTS 1 TO 4

CODE

```c
#include <stdio.h>

int add(int a, int b){

return a+b;

}

int main() {

int x = 5;

int y = 10;

int res = add(x, y);

printf("result: %d", res);

return 0;

}
```

OUTPUTS *(RIGHT)*

```
❯ gcc -g -o example example.c
❯ gdb
GNU gdb (GDB) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file example
Reading symbols from example...
(gdb) break example.c:11
Breakpoint 1 at 0x1163: file example.c, line 11.
(gdb) run
Starting program: /home/tulu/Documents/software/ass02/example

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.org>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, main () at example.c:11
11          int res = add(x, y);
(gdb) print x
$1 = 5
(gdb) print y
$2 = 10
(gdb) print res
$3 = 32767
(gdb)
```

PRINTING CONTENT OF AN ARRAY OR CONTIGUOUS MEMORY

PROCESS

- 1. Create object file using gcc (gcc -g -o array array.c)
- 2. Use gdb to select the object file (file array)
- 3. Create break point at line no 12 (break array.c:12)
- 4. Run the code until that point (run)
- 5. Print arr value (print arr)

CODE

```
#include<stdio.h>
int add(int arr[], int n){
    int res = 0;
    for(int i=0;i<n;i++)
        res += arr[i];
    return res;
}

int main(){
    int arr[5] = {1, 2, 3, 4, 5};
    int res = add(arr, 5);
    printf("result: %d\n", res);
    return 0;
}
```

OUTPUT

```
❯ gcc -g -o array array.c
❯ gdb
GNU gdb (GDB) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file array
Reading symbols from array...
(gdb) break array.c:12
Breakpoint 1 at 0x11c8: file array.c, line 12.
(gdb) run
Starting program: /home/tulu/Documents/software/ass02/array

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.org>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, main () at array.c:12
12          int res = add(arr, 5);
(gdb) print arr
$1 = {1, 2, 3, 4, 5}
```

RESULT

- At line 12 the array is initialised. So when we print arr it displays contents of arr.

PRINTING FUNCTION ARGUMENTS

PROCESS

- **1. Create object file using gcc (gcc -g -o local local.c)**
- **2. Use gdb to select the object file (file array)**
- **3. Create break point at line no 5 (break local.c:5)**
- **4. Run the code until that point (run)**
- **5. Print local variable values (info locals)**

CODE

```
#include <stdio.h>
int add(int a, int b) {
return a + b;
}
int main() {
int x = 5;
int y = 7;
int result = add(x, y);

int array[6] = {1, 2, 3, 4, 5, 6};
printf("Result: %d\n", result);
return 0;
}
```

OUTPUT

```
ubuntu@ubuntu:~/SWENGG2$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file sample
Reading symbols from sample...
(gdb) break add
Breakpoint 1 at 0x860: file sample.c, line 5.
(gdb) run
Starting program: /home/ubuntu/SWENGG2/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/aarch64-linux-gnu/libthread_db.so.1".

Breakpoint 1, add (a=5, b=7) at sample.c:5
5           return a + b;
(gdb) print add
$1 = {int (int, int)} 0xaaaaaaaa0854 <add>
(gdb) info args
a = 5
b = 7
```

RESULT

- **at line 5 c is initialized but d and e are not initialized. So in info locals c is given value 5 and others are 0.**

NEXT, CONTINUE, SET COMMAND

PROCESS

- 1. create object file using gcc (gcc -g -o array array.c)
- 2. use gdb to select the object file (file array)
- 3. create break point at line no 3 (break local.c:3) ## as line 3 is a function so it sets breakpoint in next line
- 4. using next command go to next line (next)
- 5. set res value (set variable res = 10)
- 6. print res value (print res)
- 7. create second breakpoint (break array.c:14)
- 8. continue to the breakpoint (continue)
- 9. print res value (print res)

CODE

```
#include<stdio.h>
int add(int arr[], int n){
    int res = 0;
    for(int i=0;i<n;i++)
        res += arr[i];
    return res;
}
int main(){
    int arr[5] = {1, 2, 3, 4, 5};
    int res = add(arr, 5);
    printf("result: %d\n", res);
    return 0;
}
```

OUTPUT

```
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file array
Reading symbols from array...
(gdb) break array.c:3
Breakpoint 1 at 0x1154: file array.c, line 4.
(gdb) next
The program is not being run.
(gdb) run
Starting program: /home/tulu/Documents/software/ass02/array

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.org>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, add (arr=0x7fffffffd9b0, n=5) at array.c:4
4           int res = 0;
(gdb) next
5           for(int i=0;i<n;i++)
(gdb) set variable res = 10
(gdb) print res
$1 = 10
(gdb) break array.c:14
Breakpoint 2 at 0x5555555551f5: file array.c, line 14.
(gdb) continue
Continuing.
result: 25

Breakpoint 2, main () at array.c:14
14          return 0;
(gdb) print res
$2 = 25
```

RESULT

Here using next value we initialized res value with 0. next using set command res value is changed to 10. The next breakpoint is at line 14 where looping is done. If res value would have been initialized with 0 then res would be 15 but as set command is used to set res value to 10. so after continuing to line 14 res becomes 25.

SINGLE STEPPING INTO FUNCTION

PROCESS

- 1. create object file using gcc (gcc -g -o array arry.c)
- 2. use gdb to select the object file (file array)
- 3. create break point at line no 12 (break array.c:12)
- 4. run the code until that point (run)
- 5. step into the add function (step)

CODE

```
#include <stdio.h>
int add(int a, int b) {
return a + b;
}
int main() {
int x = 5;
int y = 7;
int result = add(x, y);
int array[6] = {1, 2, 3, 4, 5, 6};
printf("Result: %d\n", result);
return 0;
}
```

OUTPUT

```
ubuntu@ubuntu:~/SWENGG2$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file sample
Reading symbols from sample...
(gdb) break main
Breakpoint 1 at 0x87c: file sample.c, line 8.
(gdb) run
Starting program: /home/ubuntu/SWENGG2/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/aarch64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at sample.c:8
8       int main() {
(gdb) step
9           int x = 5;
```

RESULT

- at line 12 the breakpoint is set. So the programme runs until that point. When step command is executed then the control moves inside the add function.

LISTING ALL BREAKPOINTS

PROCESS

- 1. create object file using gcc (gcc -g -o array arry.c)
- 2. use gdb to select the object file (file array)
- 3. create break point at line no 4 & 14 (break array.c:4 & break array.c:14)
- 4. run the code (run)
- 5. Use ‘info breakpoints’ to view the details of each breakpoint

OUTPUT

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555154 in add at array.c:4
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00005555555551f5 in main at array.c:14
        breakpoint already hit 1 time
(gdb)
```

CODE

```
#include<stdio.h>
int add(int arr[], int n){
    int res = 0;
    for(int i=0;i<n;i++)
        res += arr[i];
    return res;
}
int main(){
int arr[5] = {1, 2, 3, 4, 5};
int res = add(arr, 5);
printf("result: %d\n", res);
return 0;
}
```

RESULT

- The address, details and enable details of the breakpoints have been displayed.

IGNORING A BREAKPOINT FOR N-OCCURENCE

PROCESS

1. create object file using gcc (gcc -g -o array array.c)
2. use gdb to select the object file (file array)
3. create break point at line no 6 (break array.c:6)
4. Command ‘info breakpoints’ to view the existing breakpoints sequence
5. Use command ‘ignore 1 3’ to ignore next 3 occurrences of breakpoint 1
6. The program is executed through ‘run’

CODE

```
#include<stdio.h>
int add(int arr[], int n){
    int res = 0;
    for(int i=0;i<n;i++)
        res += arr[i];
    return res;
}
int main(){
    int arr[5] = {1, 2, 3, 4, 5};
    int res = add(arr, 5);
    printf("result: %d\n", res);
    return 0;
}
```

OUTPUT

```
❯ gdb
GNU gdb (GDB) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file array
Reading symbols from array...
(gdb) break array.c:6
Breakpoint 1 at 0x1164: file array.c, line 6.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001164 in add at array.c:6
(gdb) ignore 1 3
Will ignore next 3 crossings of breakpoint 1.
(gdb) run
Starting program: /home/tulu/Documents/software/ass02/array

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.org>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, add (arr=0x7fffffffd9b0, n=5) at array.c:6
6               res += arr[i];
(gdb) print res
$1 = 6
(gdb)
```

RESULT

Breakpoint occurs after 3 occurrences, and is verified through printing the value of result as 1+2+3=6 (3 iterations of the loop in add() function)

PROCESS

ENABLE / DISABLE A BREAKPOINT

- 1. Create object file using gcc (gcc -g -o array arry.c)
- 2. Use gdb to select the object file (file array)
- 3. Create break point at line no 6 (break array.c:6)
- 4. Run the code (run)
- 5. Use ‘enable breakpoints’ to set the breakpoints as enabled.
- 6. Running the program (through ‘continue’) indicates that the program stops at the breakpoint.
- 7. Use ‘disable breakpoints’ to set the breakpoints as disabled.
- 8. Running the program (through continue) indicated normal termination of program with result as 15.

CODE

```
#include<stdio.h>
int add(int arr[], int n){
int res = 0;
for(int i=0;i<n;i++)
res += arr[i];
return res;
}
int main(){
int arr[5] = {1, 2, 3, 4, 5};
int res = add(arr, 5);
printf("result: %d\n", res);
return 0;
}
```

OUTPUT

```
(gdb) enable breakpoints
(gdb) continue
Continuing.

Breakpoint 1, add (arr=0x7fffffffd9b0, n=5) at array.c:6
6                   res += arr[i];
(gdb) disable breakpoints
(gdb) continue
Continuing.
result: 15
[Inferior 1 (process 19442) exited normally]
(gdb)
```

RESULT

- The functionalities of both the statements have been demonstrated.

BREAK CONDITION AND COMMAND

PROCESS

- 1. create object file using gcc (gcc -g -o array array.c)
- 2. use gdb to select the object file (file array)
- 3. create break point at line no 6 (break array.c:6)
- 4. Use ‘commands 1’ to define commands for breakpoint 1, when it is encountered.
- 5. The program is executed through ‘run’

CODE

```
#include<stdio.h>
int add(int arr[], int n){
    int res = 0;
    for(int i=0;i<n;i++)
        res += arr[i];
    return res;
}
int main(){
    int arr[5] = {1, 2, 3, 4, 5};
    int res = add(arr, 5);
    printf("result: %d\n", res);
    return 0;
}
```

OUTPUT

```
❯ gdb
GNU gdb (GDB) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file array
Reading symbols from array...
(gdb) break array.c:6 if res>6
Breakpoint 1 at 0x1164: file array.c, line 6.
(gdb) commands 1
Type commands for breakpoint(s) 1, one per line.
End with a line saying just "end".
>print res
>end
(gdb) run
Starting program: /home/tulu/Documents/software/ass02/array

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.org>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, add (arr=0x7fffffffd9b0, n=5) at array.c:6
6               res += arr[i];
$1 = 10
(gdb)
```

RESULT

The breakpoint is reached and given commands are executed.

EXAMINING THE STACK TRACE

- During function calls stack is used to ensure the order and at a certain breakpoint to inspect the contents of the stack GDB debugger can be used
- Backtrace (bt) is a command used to inspect the stack trace
- This gives us the contents of stack (The functions that are invoked in order) . The top function executed first . (LIFO)

```
(gdb) bt
No stack.
(gdb) run
Starting program: /home/ashish5554/swelab/exam
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 2, main () at Example2.c:16
16              for(int i=0;i<3;i++){
(gdb) bt
#0  main () at Example2.c:16
(gdb) continue
Continuing.

Breakpoint 1, f1 (x=0) at Example2.c:7
7               printf("Iam inside function f1 and value of integer passed is %d\n",x);
(gdb) bt
#0  f1 (x=0) at Example2.c:7
#1  0x000055555555521f in main () at Example2.c:17
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


void f1(int x){
        printf("Iam inside function f1 and value of integer passed is %d\n",x);
}

void f2(){
        printf("Iam INside function f2\n");
}

int main(){
        for(int i=0;i<3;i++){
                f1(i);
        }
        f2();
        return 0;
}
```

- In the above program exam is a .exe file with debugging info of the below code and we used bt to analyse the stack trace at various breakpoints

- BREAKPOINTS ARE AT: LINES 7 AND 16

EXAMINING STACK TRACE FOR MULTI-THREADED PROGRAM:

- While working with multi-threaded programs there can be bugs relating to synchronisation issues, etc.
- Making deeply nested, recursive calls and using large automatic arrays can cause problems because multithreaded programs have a more limited stack size than single-threaded programs.
- Specifying an inadequate stack size, or using non-default stacks.
- To debug these multi-thread issues GDB provides few feature to choose threads and debug them.
- Here is the simple program (single threaded itself). But still in the same way we can debug multi threaded programs.

```
(gdb) info threads
  Id   Target Id                                      Frame
* 1    Thread 0x7ffff7fa7740 (LWP 12265) "exam" f1 (x=0) at Example2.c:7
(gdb) thread 1
[Switching to thread 1 (Thread 0x7ffff7fa7740 (LWP 12265))]
#0  f1 (x=0) at Example2.c:7
7               printf("Iam inside function f1 and value of integer passed is %d\n",x);
(gdb) bt
#0  f1 (x=0) at Example2.c:7
#1  0x000055555555521f in main () at Example2.c:17
(gdb)
```

- `info threads` command is used to know all the threads in thour program. As in the above picture there is only one thread.
- `thread id` : We can switch between different threads and inspect stack trace of different threads .
  As in the above picture I went to thread with thread id 1 and inspected its stack trace.

MORE INFORMATION

- https://docs.oracle.com/cd/E19455-01/806-5257/6je9h033m/index.html

CORE FILE DEBUGGING:

- A core file is an image of a process that has crashed
  It contains all process information pertinent to debugging: contents of hardware registers, process status, and process data. Gdb will allow you use this file to determine where your program crashed.

- First, the program must be compiled with debugging information .

- Second, the program must have crashed and left a core file. It should tell you if it has left a core file with the message "core dumped".

- The command line to start gdb to look at the core file is: gdb program core

- Consider a simple preogram which results in core dumped seg fault error.

```c
#include <stdio.h>
#include <stdlib.h>


int main(){
        int* ptr=NULL;
        printf("Dereferencing the integer pointer : %d\n",*ptr);
        return 0;
}
```

- Let exam3 be the .exe file of the following C program, after execution the program:

```
ashish5554@ashish5554-VirtualBox:~/swelab$ ./exam3
Segmentation fault (core dumped)
```

- Now to load the core file of the crashed program we use `gdb ./exam3 core` command

- Now we can use various commnds like bt, list print to analyse stack trace and variable values around the crash location.

DEBUGGING OF AN ALREADY RUNNING PROGRAM:

- To debug an already running program we required the pid of that program and by using attach command we need to debug it.
- Since C programs execute very fast we need to include a sleep(45) line .
- This helps us to increase the execution time by 45 sec.
- In one terminal execute the program and in another terminal find pid of this running process using command `ps -C program -o pid h`

```
Type "apropos word" to search for commands related to "word".
(gdb) attach 7863
Attaching to process 7863
Reading symbols from /home/ashish5554/swelab/exam...
Reading symbols from /lib/x86_64-linux-gnu/libc.so.6...
Reading symbols from /usr/lib/debug/.build-id/c2/89da5071a3399de893d2af81d6a30c62646e1e.debug...
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-id/15/921ea631d9f36502d20459c43e5c85b7d6ab76.debug...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
0x00007f1f61ae578a in __GI___clock_nanosleep (clock_id=clock_id@entry=0, flags=flags@entry=0, req=req@entry=0x7fff3fe870f0, rem=rem@entry=0x7fff3fe870f0) at ../sysdeps/unix/sysv/linux/clock
_nanosleep.c:78
78      ../sysdeps/unix/sysv/linux/clock_nanosleep.c: No such file or directory.
(gdb) q
A debugging session is active.

        Inferior 1 [process 7863] will be detached.

Quit anyway? (y or n) n
Not confirmed.
(gdb) continue
Continuing.
[Inferior 1 (process 7863) exited normally]
(gdb) break Example2.c:15
Breakpoint 1 at 0x560200ab1209: file Example2.c, line 15.
(gdb) run
Starting program: /home/ashish5554/swelab/exam
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at Example2.c:15
15              sleep(45);
(gdb) continue
Continuing.
Iam inside function f1 and value of integer passed is 0
Iam inside function f1 and value of integer passed is 1
Iam inside function f1 and value of integer passed is 2
Iam INside function f2
[Inferior 1 (process 7879) exited normally]
(gdb)
```

- The pid of the process is 7863
- `attach 7863` command can be used to attach the running programs debugging info to the gdb debugger and can be used to inspect the stack trace and variable values using other gdb commands.

EXTRA GDB COMMANDS: WATCHPOINT

- It is a command used to halt the program whenever the value of a specified variable changes .

- This can be used to track the values of the important variables values and when those variable change (Not good for the program) we can halt it using watchpoint and debug the reason of the change.

```
(gdb) watch i
Hardware watchpoint 2: i
(gdb) continue
Continuing.
Iam inside function f1 and value of integer passed is 0

Hardware watchpoint 2: i

Old value = 0
New value = 1
0x000055555555551d1 in main () at GDB_Tutorial.c:15
15                  for(int i=0;i<3;i++){
(gdb)
```

- As we can see I set a watchpoint on variable i when the for loop executed the i value changes and the program halts which helps us to check the state of i (*i.e* value changes)

- Now to remove watchpoint i just simply use the command `delete watch x`

- src: gnu gcc documents
  oracle for multithreading

MORE INFORMATION

- stackoverflow for core dump issue(File still not generated): https://stackoverflow.com/questions/7732983/core-dump-file-is-not-generated

ACKNOWLEDGEMENT

We would like to thank our Professors for providing this opportunity of exploring about GDB Debugger and improving our Presentation skills. We are forever indebted to the Lord, God Almighty, for without His marvellous divine creations nothing of this would be possible. Our heartfelt gratitude goes forward to all our elders, parents, teachers, guides and well-wishers for their support and blessings all throughout.

THANK YOU