ASSIGNMENT 3

Software Engineering Lab

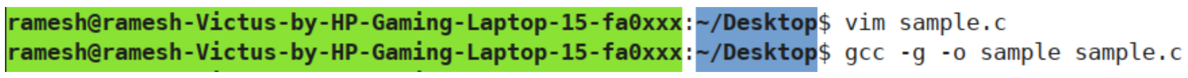
Member	Enrollment No
Ramesh Chandra Soren	2022CSB086
Moodu Roopa	2022CSB087
Jitendra Kumar Dodwadiya	2022CSB089
Deep Sutariya	2022CSB090

1. Running a Program

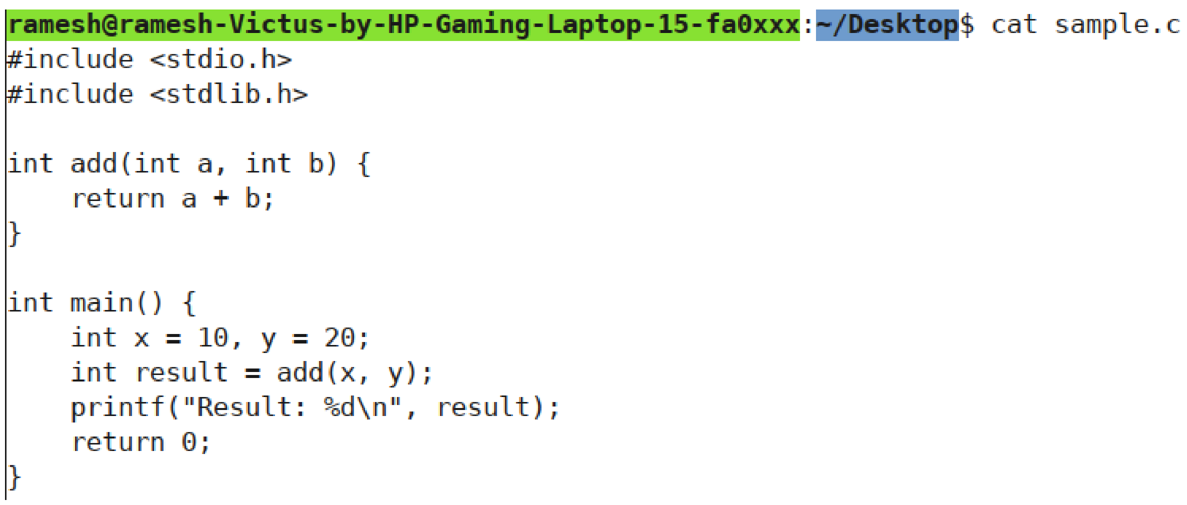
Syntax: `(gdb) run`

Explanation:

Starts the program within the GDB environment. You can pass command-line arguments as needed.



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ vim sample.c
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gcc -g -o sample sample.c
```



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ cat sample.c
#include <stdio.h>
#include <stdlib.h>

int add(int a, int b) {
    return a + b;
}

int main() {
    int x = 10, y = 20;
    int result = add(x, y);
    printf("Result: %d\n", result);
    return 0;
}
```

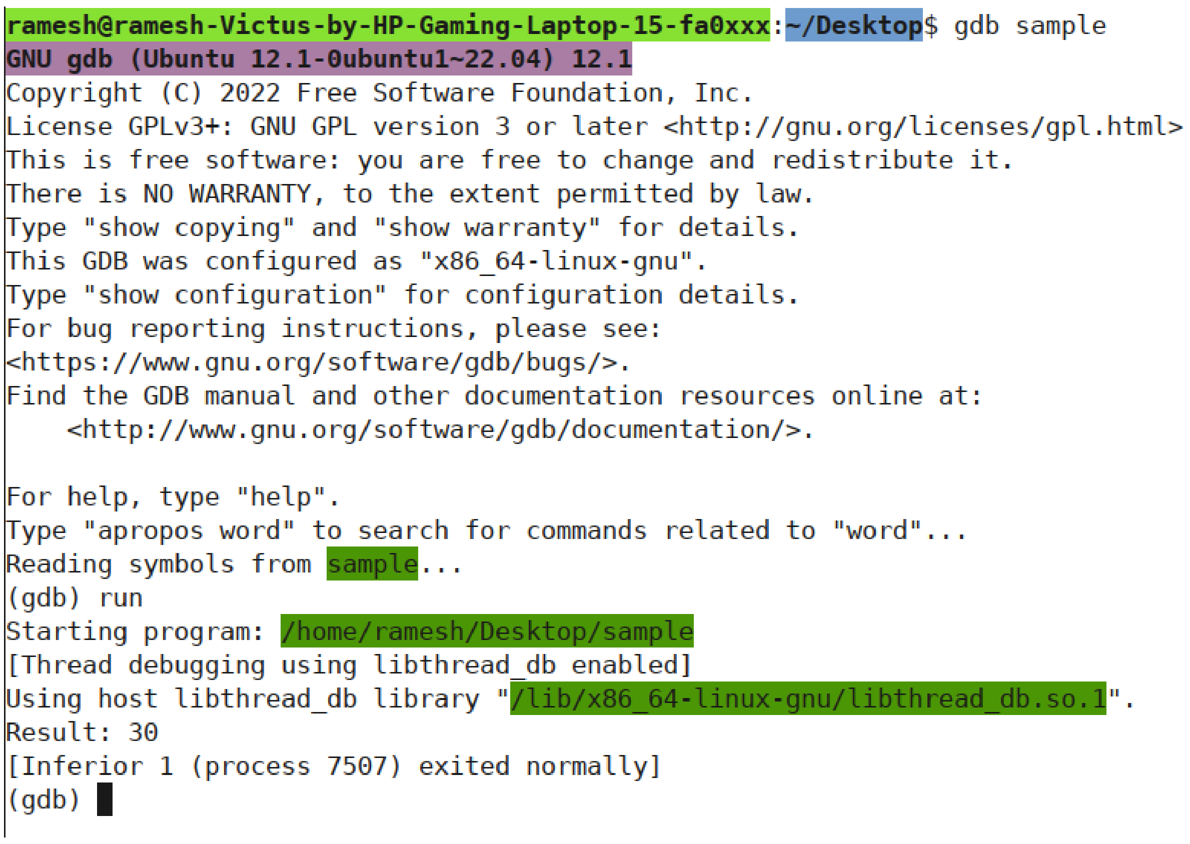
- Start GDB:
  At the shell prompt, start GDB with your executable. For example, if your executable is named `sample`, type: gdb sample.

  This will drop you into the GDB environment, and you'll see the prompt change to `(gdb)`.

- Run the Program:

  Now that you're in GDB, type the command: run

  GDB will then execute your program under its control.



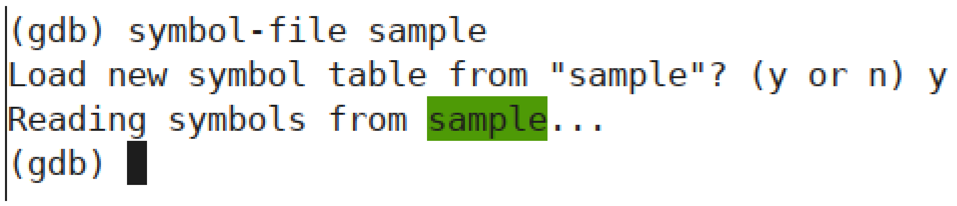
```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gdb sample
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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"...
Reading symbols from sample...
(gdb) run
Starting program: /home/ramesh/Desktop/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Result: 30
[Inferior 1 (process 7507) exited normally]
(gdb) 
```

2. Loading Symbol Table

Syntax: `(gdb) symbol-file sample`

Explanation:

Loads the symbol table for the executable sample. This ensures that GDB can correlate executable code with source code, allowing you to inspect variables and function names.



```
(gdb) symbol-file sample
Load new symbol table from "sample"? (y or n) y
Reading symbols from sample...
(gdb) 
```

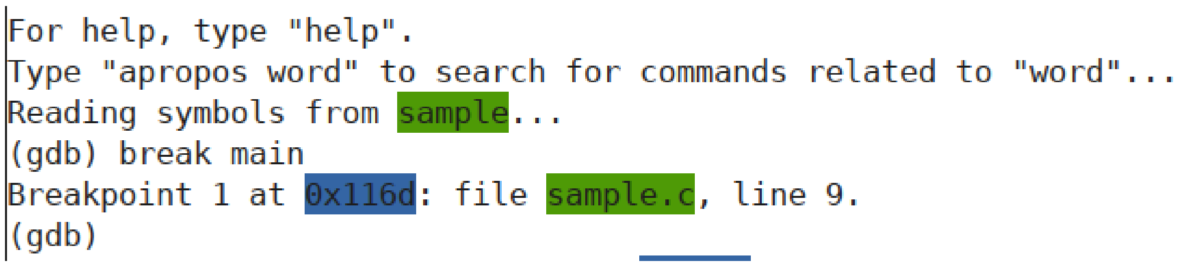
3. Setting a Breakpoint

Syntax: `(gdb) break main`

Explanation:

Sets a breakpoint at the start of the `main` function.

You can also set breakpoints at other functions or specific line numbers.



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample...
(gdb) break main
Breakpoint 1 at 0x116d: file sample.c, line 9.
(gdb)
```

4. Listing Variables and Examining Their Values

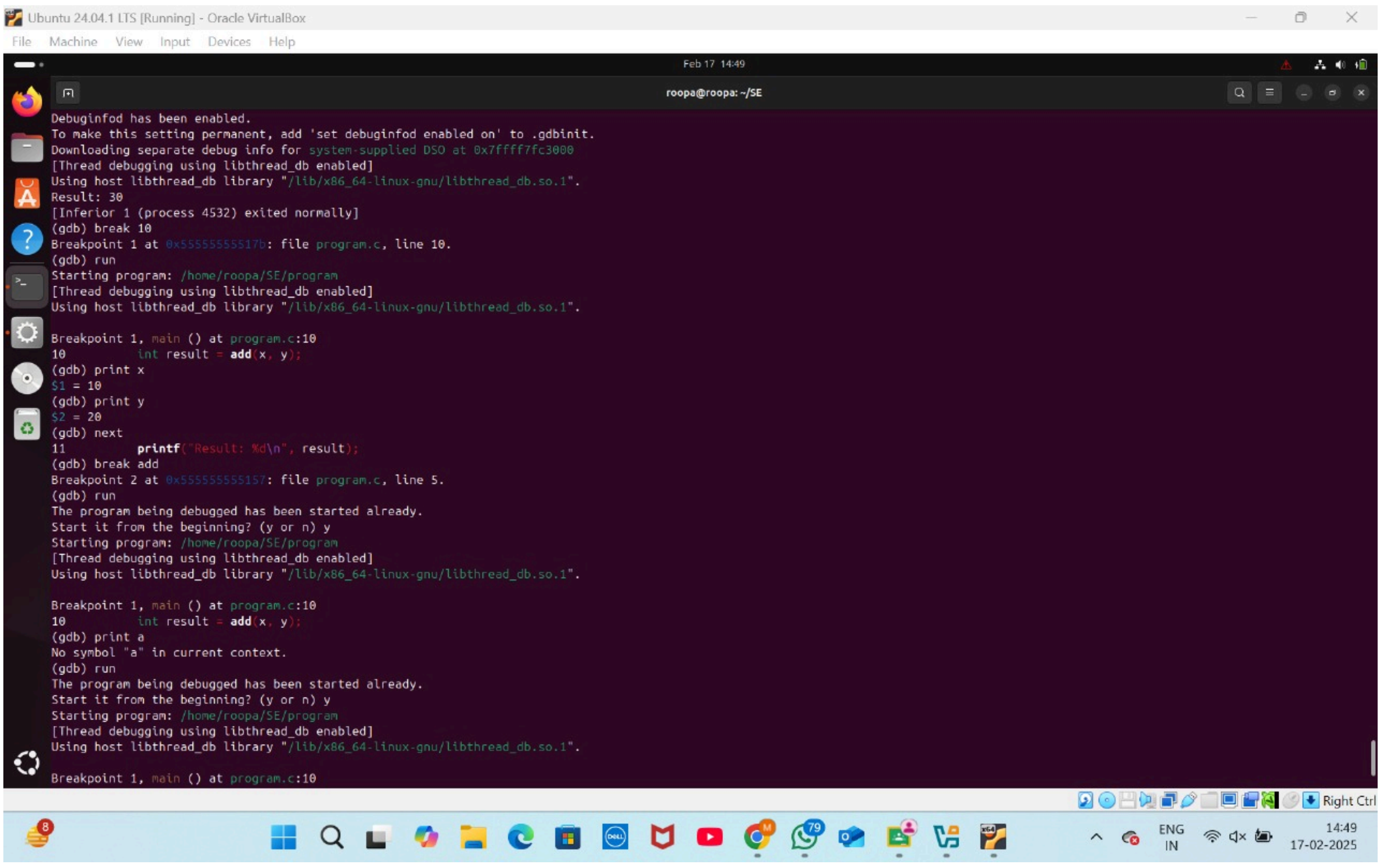
- ☐ Set a Breakpoint Before Running: Place a breakpoint at a point in your code where variables (like `x`) are in scope—for example, inside the `main` function before the variable is used or modified.
- ☐ Run the Program Again: After setting the breakpoint, run the program. When execution stops at the breakpoint, you'll have an active frame.
- ☐ Inspect Variables in the Active Frame: Now that you're paused in `main` (or any other function where `x` is defined), you can use:

Syntax:

```
(gdb) info locals

(gdb) print x
```

Explanation:

- `info locals` lists all local variables in the current stack frame.
- `print x` displays the current value of the variable `x`.



Ubuntu 24.04.1 LTS [Running] - Oracle VirtualBox

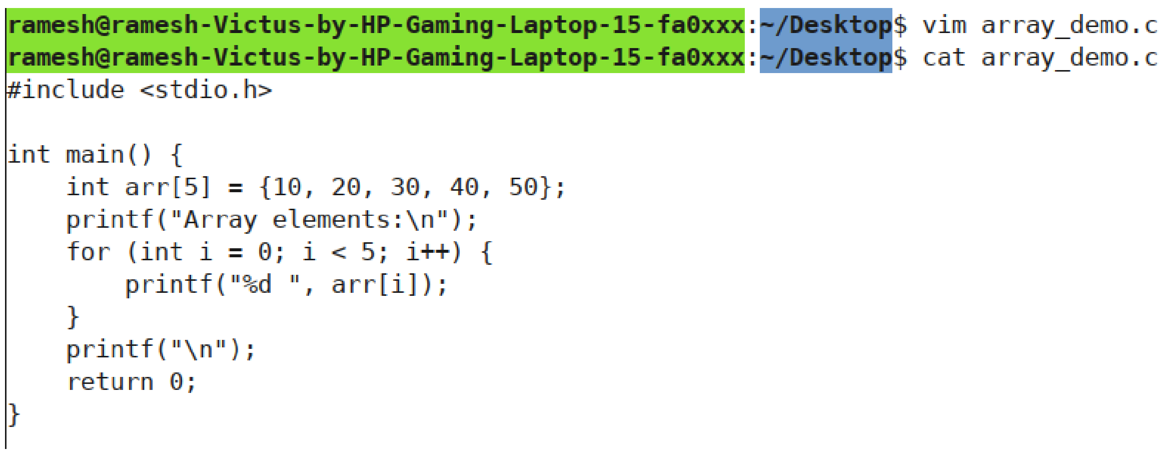
```
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc3000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Result: 30
[Inferior 1 (process 4532) exited normally]
(gdb) break 10
Breakpoint 1 at 0x55555555517b: file program.c, line 10.
(gdb) run
Starting program: /home/roopa/SE/program
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at program.c:10
10          int result = add(x, y);
(gdb) print x
$1 = 10
(gdb) print y
$2 = 20
(gdb) next
11          printf("Result: %d\n", result);
(gdb) break add
Breakpoint 2 at 0x555555555157: file program.c, line 5.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/roopa/SE/program
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at program.c:10
10          int result = add(x, y);
(gdb) print a
No symbol "a" in current context.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/roopa/SE/program
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at program.c:10
```

5. Printing Content of an Array or Contiguous Memory

Syntax: `(gdb) print array[0]@5`

Explanation:

This command prints 5 consecutive elements of the array starting at `array[0]`. (For demonstration, assume we have declared an array in our code.)



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ vim array_demo.c
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ cat array_demo.c
#include <stdio.h>

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    printf("Array elements:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

- ☐ Compile the Program with Debug Symbols: Use the `-g` flag to compile the code so that debugging information is included:

  `gcc -g array_demo.c -o array_demo`

- ☐ Start GDB: Launch GDB with the compiled executable:

  `gcc ./array_demo`

- ☐ Set a Breakpoint After the Array Initialization:

  If you set a breakpoint on the very line where the array is declared (line 4 in this case), the initialization may not have executed yet.

  Instead, set a breakpoint on a later line (e.g., on the `printf` line) to ensure the array has been initialized: `(gdb) break array_demo.c:5`

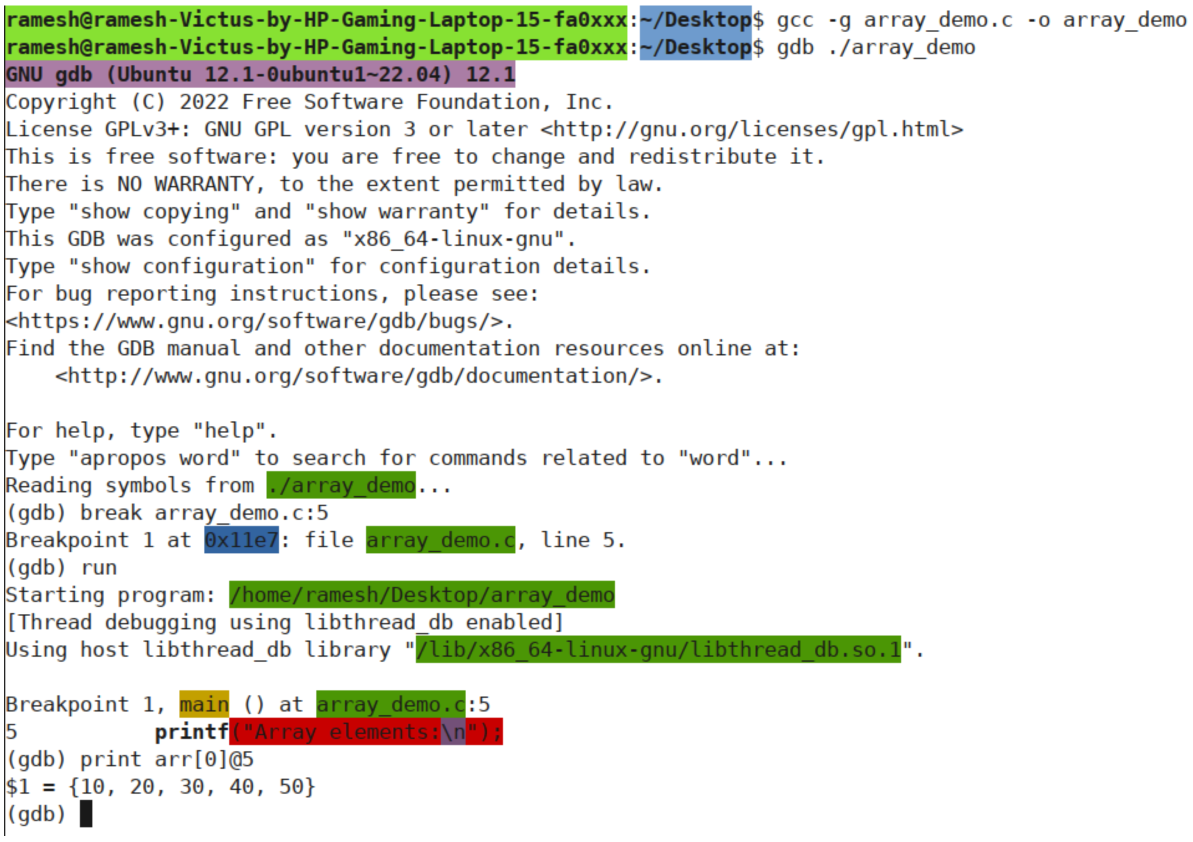
- ☐ Run the Program: Start execution with: `(gdb) run`

  At this point, the array arr has already been initialized.

- [ ] Print the Array Contents: Use the following GDB command to display the contents of the array: `(gdb) print arr[0]@5`

Explanation:

- arr[0]@5: This syntax tells GDB to print 5 consecutive elements starting at arr[0]. It effectively displays arr[0], arr[1], arr[2], arr[3], and arr[4].



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gcc -g array_demo.c -o array_demo
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gdb ./array_demo
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 "x86_64-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"...
Reading symbols from ./array_demo...
(gdb) break array_demo.c:5
Breakpoint 1 at 0x11e7: file array_demo.c, line 5.
(gdb) run
Starting program: /home/ramesh/Desktop/array_demo
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at array_demo.c:5
5           printf("Array elements:\n");
(gdb) print arr[0]@5
$1 = {10, 20, 30, 40, 50}
(gdb) 
```

6. Printing Function Arguments

Syntax: `(gdb) info args`

Explanation:

Lists the arguments passed to the current function. This is useful when you step into functions to verify that the parameters are as expected.

- [ ] Source Code (`array_demo.c`)
- [ ] Compile the Program with Debug Symbols

  Use the `-g` flag so that GDB can read the debugging information

- [ ] Start GDB
- [ ] Set a Breakpoint at the Start of `main()`

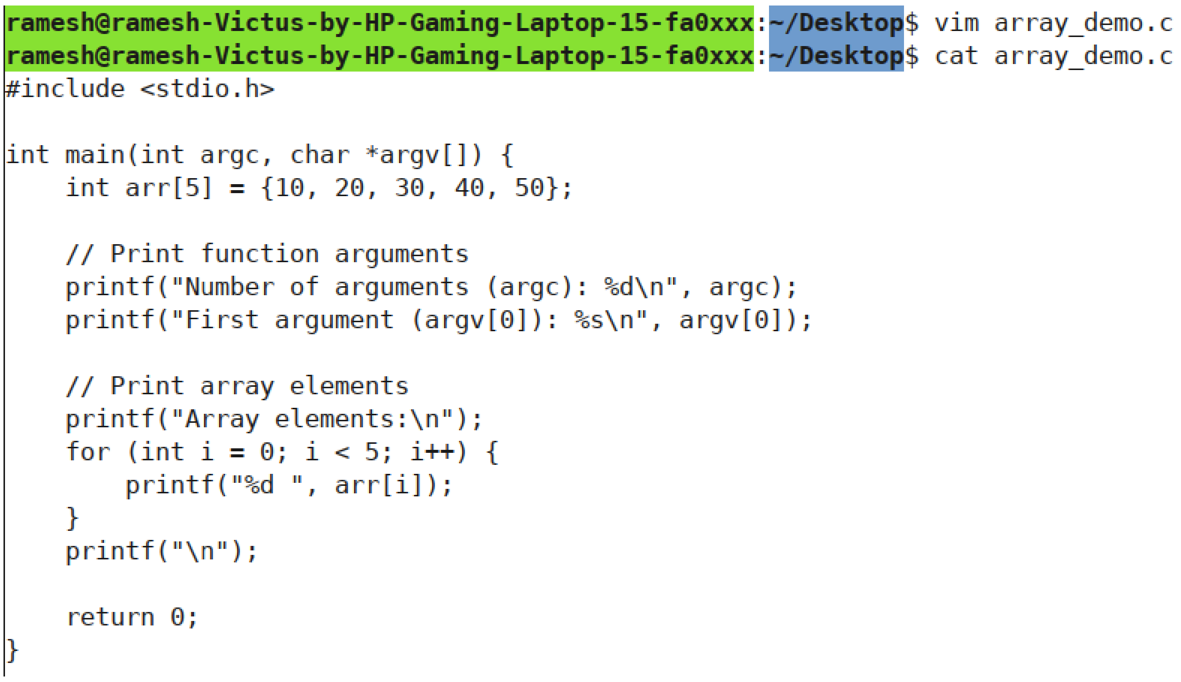
- ☐ Run the Program: GDB will stop at the beginning of `main()`
- ☐ Inspect the Function Arguments:

  Now that you're at the start of `main()`, use `info args` to inspect the arguments

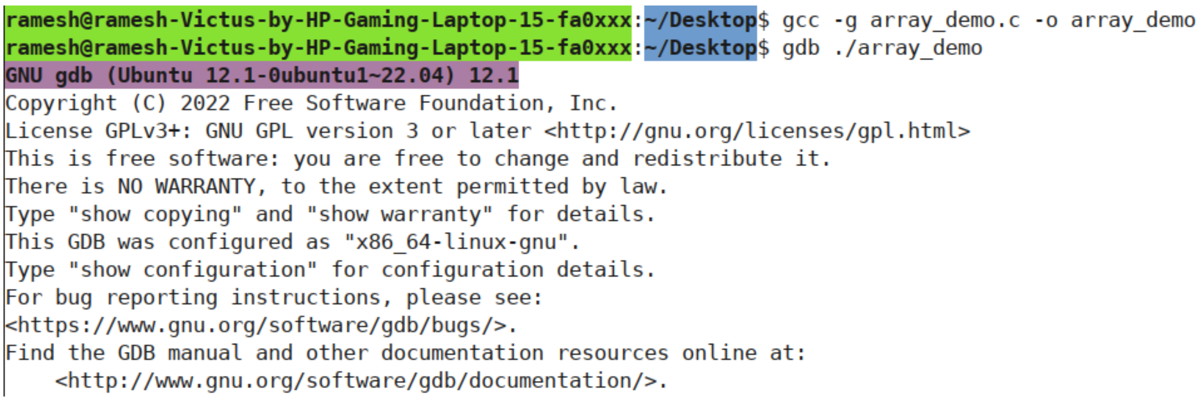
- ☐ This output indicates that `argc` is 1 (meaning one argument, usually the program name) and shows the memory address where `argv` is stored.
- ☐ Print the Array Contents:

  Now that the array is fully initialized, use the following command to print its contents. `(gdb) print arr[0]@5`

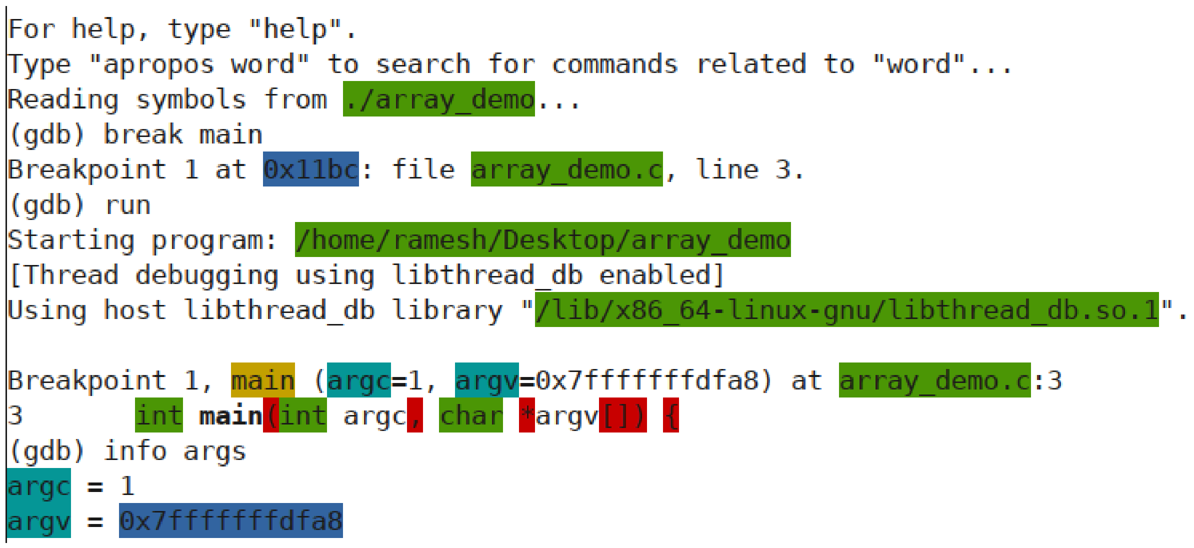
  This command prints 5 consecutive elements starting from `arr[0]`.



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ vim array_demo.c
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ cat array_demo.c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int arr[5] = {10, 20, 30, 40, 50};

    // Print function arguments
    printf("Number of arguments (argc): %d\n", argc);
    printf("First argument (argv[0]): %s\n", argv[0]);

    // Print array elements
    printf("Array elements:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gcc -g array_demo.c -o array_demo
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gdb ./array_demo
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 "x86_64-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"...
Reading symbols from ./array_demo...
(gdb) break main
Breakpoint 1 at 0x11bc: file array_demo.c, line 3.
(gdb) run
Starting program: /home/ramesh/Desktop/array_demo
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffdfa8) at array_demo.c:3
3         int main(int argc, char *argv[]) {
(gdb) info args
argc = 1
argv = 0x7fffffffdfa8
```

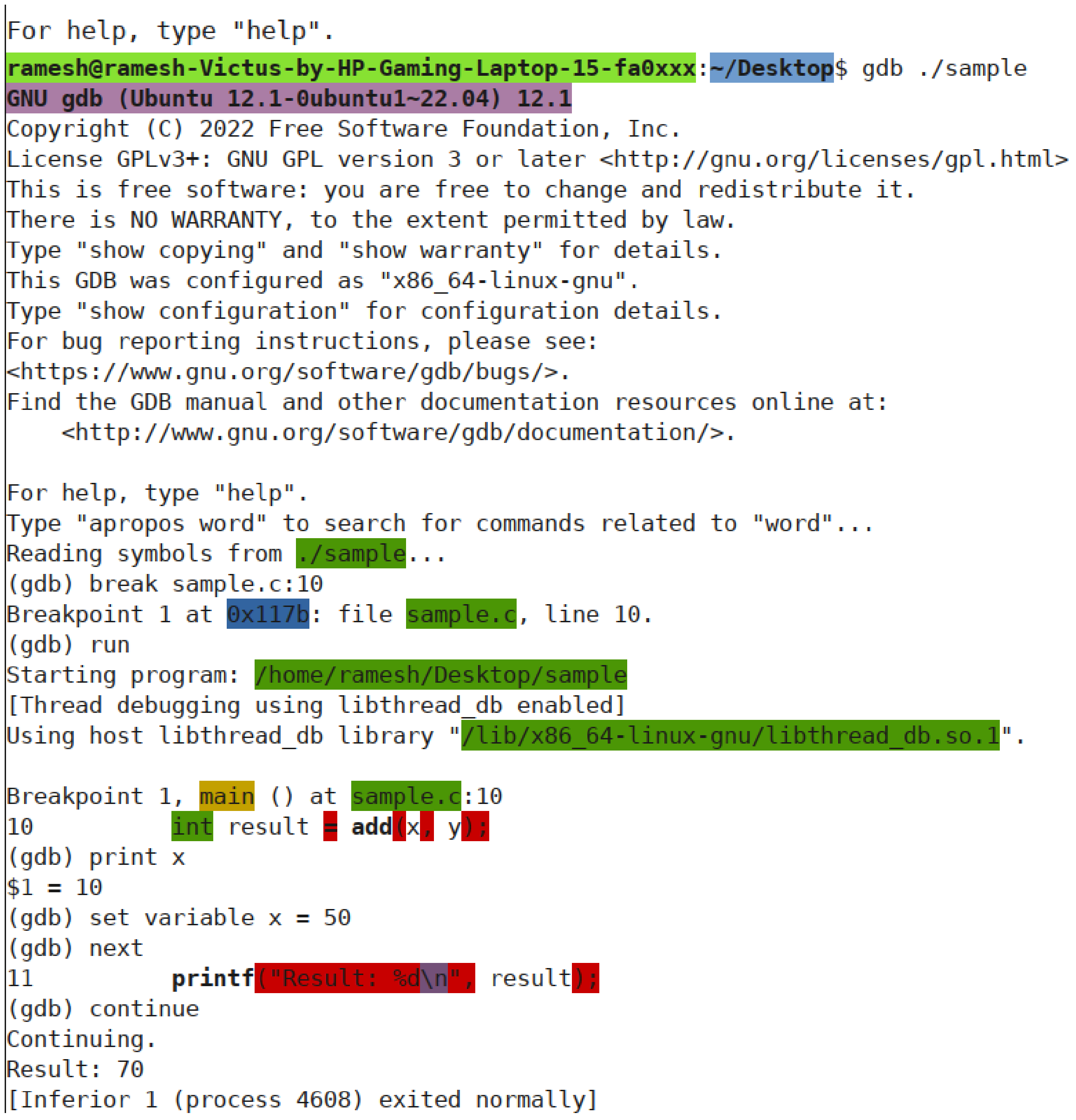
7. Next, Continue, and Set Variable Commands

Syntax:

- Next (step over): `(gdb) next`
- Continue (resume execution): `(gdb) continue`
- Set Variable: `(gdb) set variable x = 15`

Explanation:

- `next` executes the next line without stepping into functions.
- `continue` resumes execution until the next breakpoint is hit.
- `set variable` allows modification of a variable's value during runtime.



```
For help, type "help".
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gdb ./sample
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 "x86_64-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"...
Reading symbols from ./sample...
(gdb) break sample.c:10
Breakpoint 1 at 0x117b: file sample.c, line 10.
(gdb) run
Starting program: /home/ramesh/Desktop/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at sample.c:10
10          int result = add(x, y);
(gdb) print x
$1 = 10
(gdb) set variable x = 50
(gdb) next
11          printf("Result: %d\n", result);
(gdb) continue
Continuing.
Result: 70
[Inferior 1 (process 4608) exited normally]
```

8. Single Stepping Into a Function

Syntax: **`(gdb) step`**

Explanation:

The step command executes the next line of code and steps into any function calls. This is useful for line-by-line debugging inside a function.

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./sample...
(gdb) break main
Breakpoint 1 at 0x116d: file sample.c, line 9.
(gdb) run
Starting program: /home/ramesh/Desktop/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at sample.c:9
9           int x = 10, y = 20;
(gdb) step
10          int result = add(x, y);
(gdb) step
add (a=10, b=20) at sample.c:5
5           return a + b;
(gdb) step
6       }
(gdb) step
main () at sample.c:11
11          printf("Result: %d\n", result);
(gdb)
__printf (format=0x555555556004 "Result: %d\n") at ./stdio-common/printf.c:28
28      ./stdio-common/printf.c: No such file or directory.
(gdb) continue
Continuing.
Result: 30
[Inferior 1 (process 5761) exited normally]
```

9. Listing All Breakpoints

Syntax: **(gdb) info breakpoints**

Explanation:

Lists all breakpoints currently set in the session along with their status (enabled/disabled) and location.

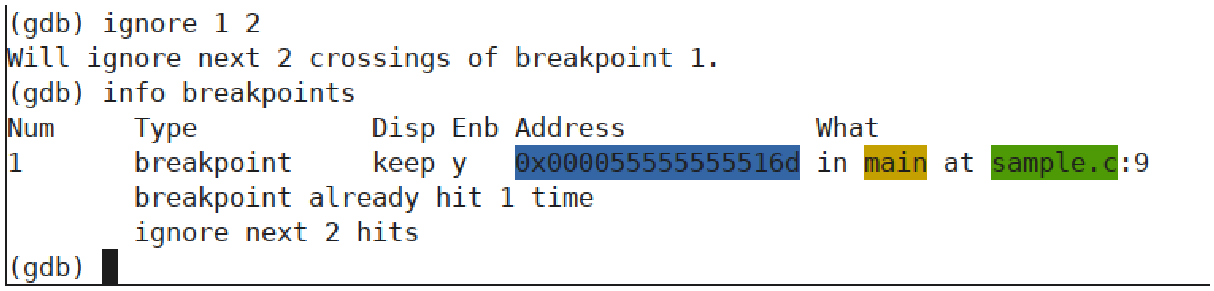
```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555516d in main at sample.c:9
        breakpoint already hit 1 time
```

10. Ignoring a Breakpoint for N Occurrences

Syntax: `(gdb) ignore 1 2`

Explanation:

Instructs GDB to ignore breakpoint number 1 for the next 2 occurrences. Useful for bypassing a frequently hit breakpoint.



```
(gdb) ignore 1 2
Will ignore next 2 crossings of breakpoint 1.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555516d in main at sample.c:9
        breakpoint already hit 1 time
        ignore next 2 hits
(gdb)
```

11. Enabling/Disabling a Breakpoint

Syntax:

- Disable a breakpoint: `(gdb) disable 1`
- Enable a breakpoint: `(gdb) enable 1`

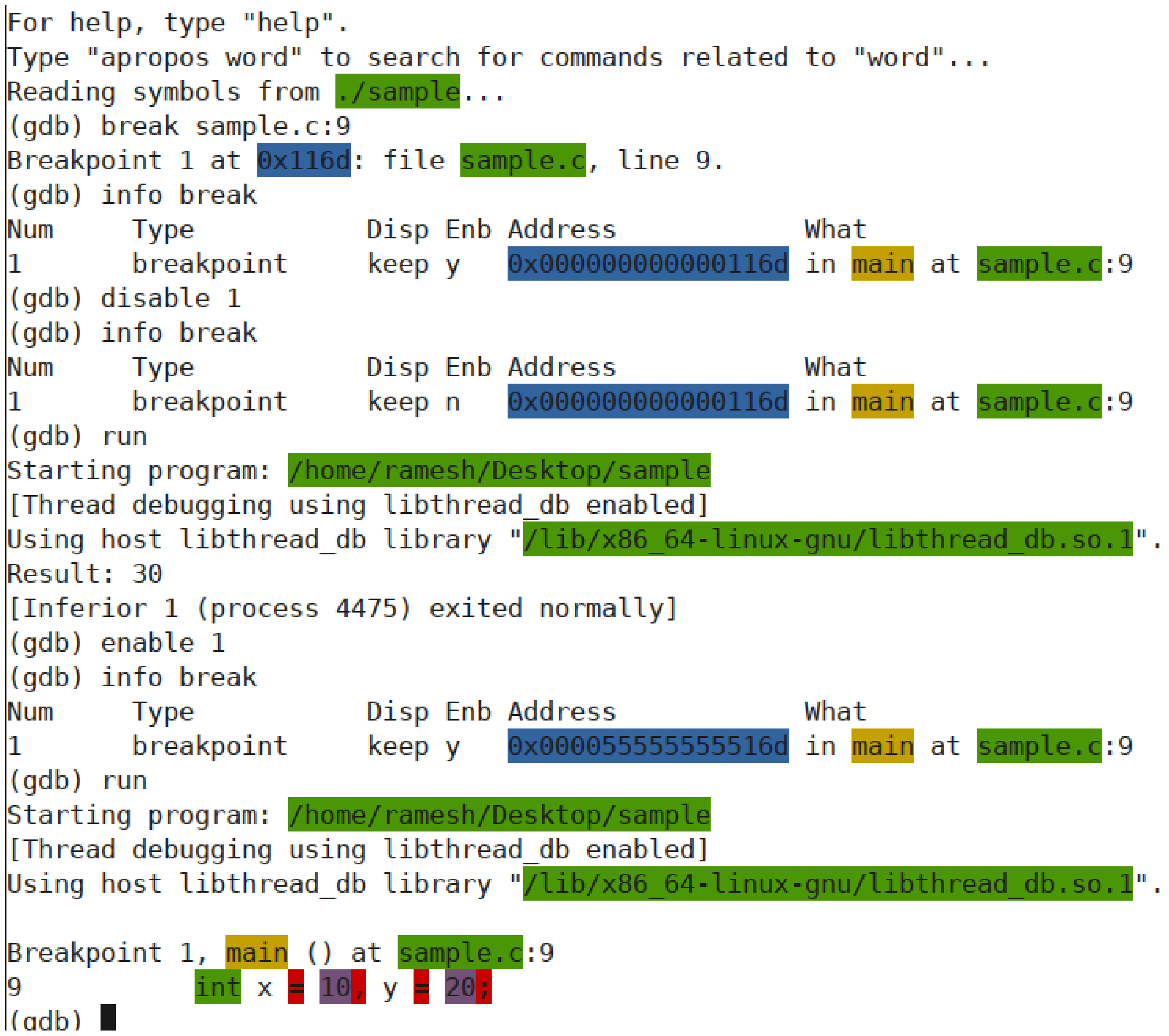
Explanation:

These commands temporarily turn off (or on) a breakpoint without deleting it.

If `info breakpoints` are giving you trouble, use `info break` instead. This should give you the desired output showing the breakpoint numbers, addresses, and enabled/disabled status.

This can sometimes happen because of version-specific quirks or if extra (invisible) characters are being interpreted as arguments. In many GDB versions, the command: `(gdb)info break`

(which is an alias for "info breakpoints") works just as well.



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./sample...
(gdb) break sample.c:9
Breakpoint 1 at 0x116d: file sample.c, line 9.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000000116d in main at sample.c:9
(gdb) disable 1
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep n   0x000000000000116d in main at sample.c:9
(gdb) run
Starting program: /home/ramesh/Desktop/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Result: 30
[Inferior 1 (process 4475) exited normally]
(gdb) enable 1
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555516d in main at sample.c:9
(gdb) run
Starting program: /home/ramesh/Desktop/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at sample.c:9
9           int x = 10, y = 20;
(gdb) 
```

12. Setting Break Conditions and Commands

Syntax:

- Conditional Breakpoint: `(gdb) break main if x == 15`
- Attaching Commands to a Breakpoint:

  (gdb) commands 1

  > print x

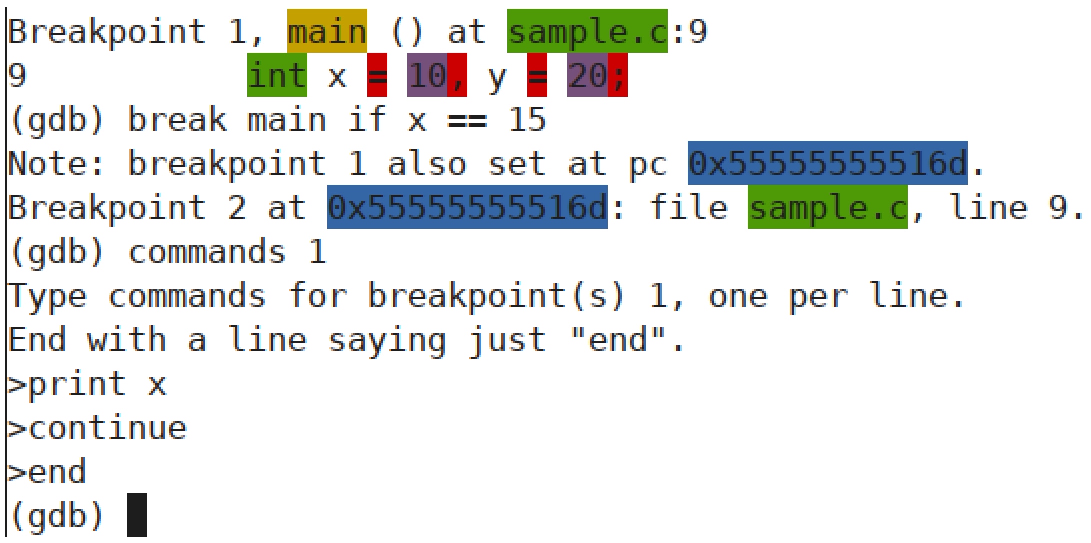
  > continue

  > end

Explanation:

- The conditional breakpoint stops only when `x` equals 15.

- The commands attached to a breakpoint will execute automatically when the breakpoint is hit. In this example, it prints the value of x and continues execution.



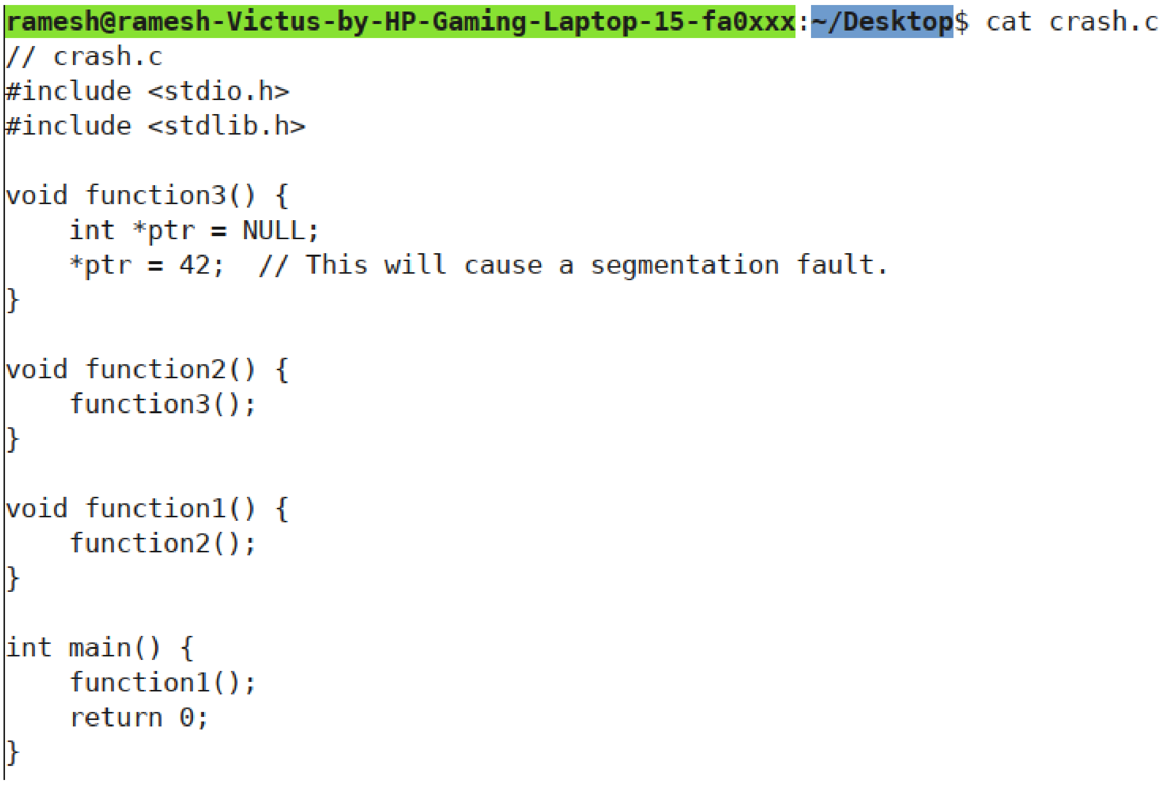
```
Breakpoint 1, main () at sample.c:9
9               int x = 10, y = 20;
(gdb) break main if x == 15
Note: breakpoint 1 also set at pc 0x55555555516d.
Breakpoint 2 at 0x55555555516d: file sample.c, line 9.
(gdb) commands 1
Type commands for breakpoint(s) 1, one per line.
End with a line saying just "end".
>print x
>continue
>end
(gdb) 
```

13. Examining Stack Trace

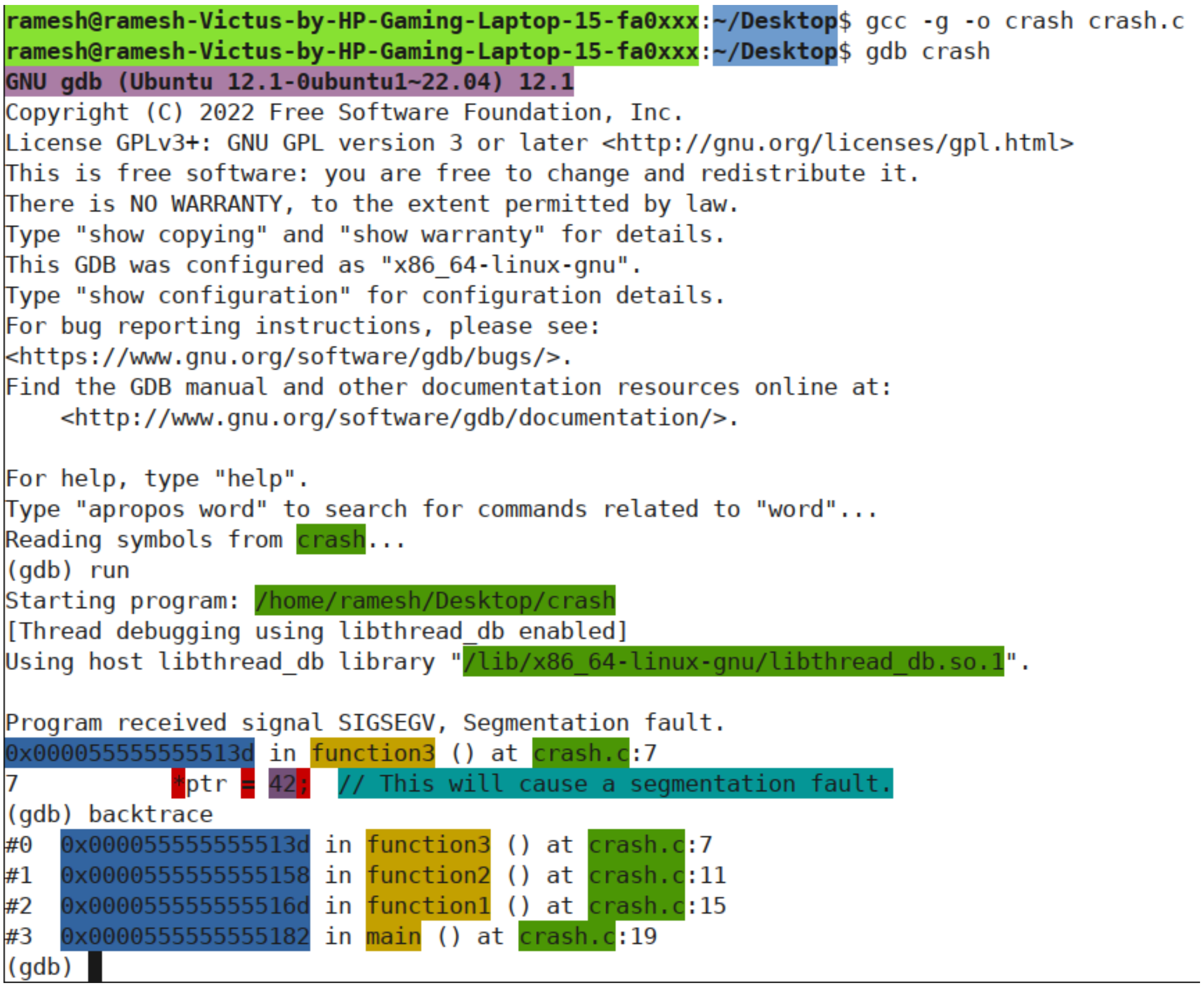
Syntax: `(gdb) backtrace`

Explanation:

Prints the current call stack, showing the chain of function calls leading to the current execution point.



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ cat crash.c
// crash.c
#include <stdio.h>
#include <stdlib.h>

void function3() {
    int *ptr = NULL;
    *ptr = 42;  // This will cause a segmentation fault.
}

void function2() {
    function3();
}

void function1() {
    function2();
}

int main() {
    function1();
    return 0;
}
```



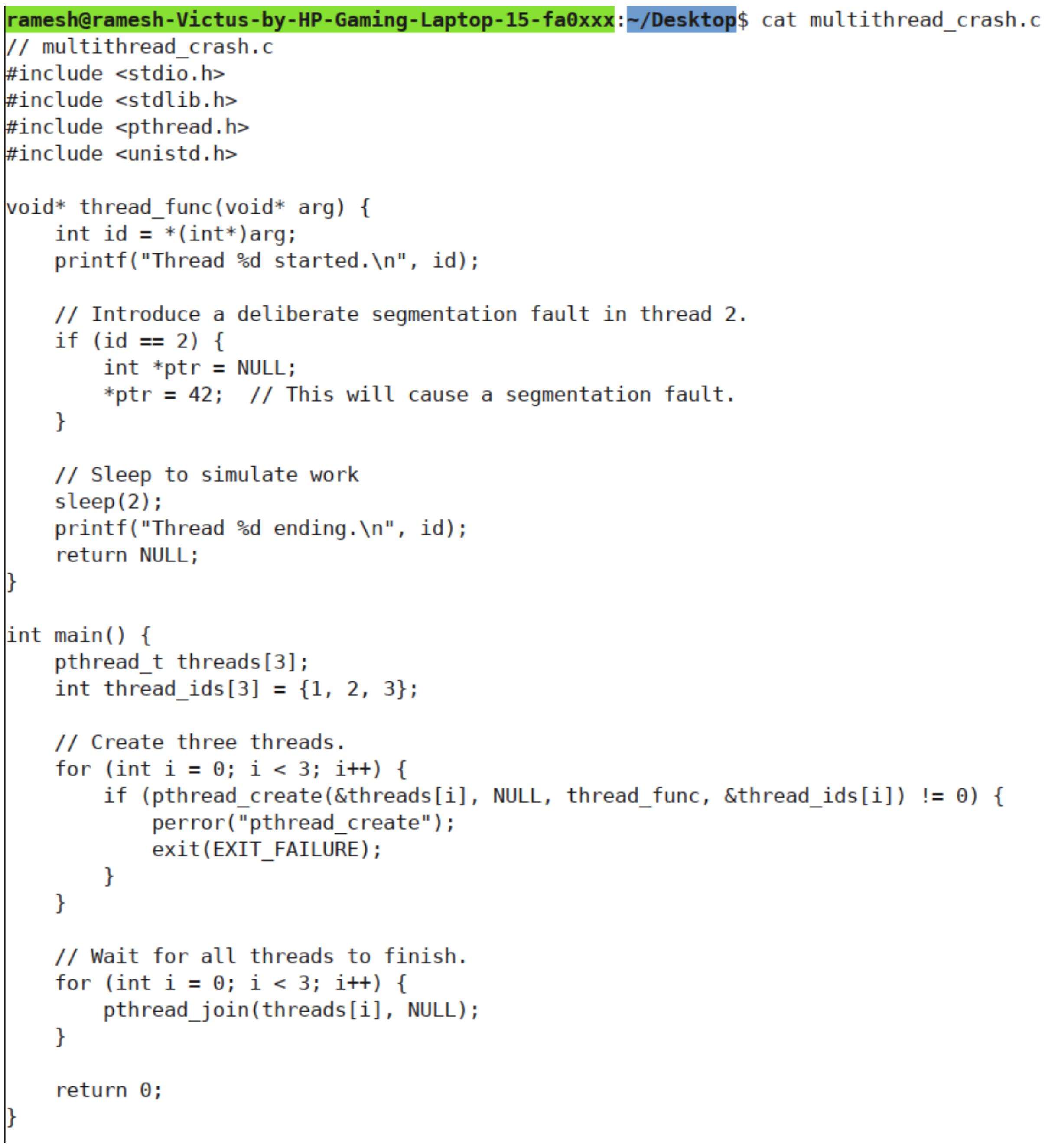
```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gcc -g -o crash crash.c
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gdb crash
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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 "x86_64-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"...
Reading symbols from crash...
(gdb) run
Starting program: /home/ramesh/Desktop/crash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x000055555555513d in function3 () at crash.c:7
7           *ptr = 42;  // This will cause a segmentation fault.
(gdb) backtrace
#0  0x000055555555513d in function3 () at crash.c:7
#1  0x0000555555555158 in function2 () at crash.c:11
#2  0x000055555555516d in function1 () at crash.c:15
#3  0x0000555555555182 in main () at crash.c:19
(gdb) 
```

14. Examining Stack Trace for a Multi-Threaded Program

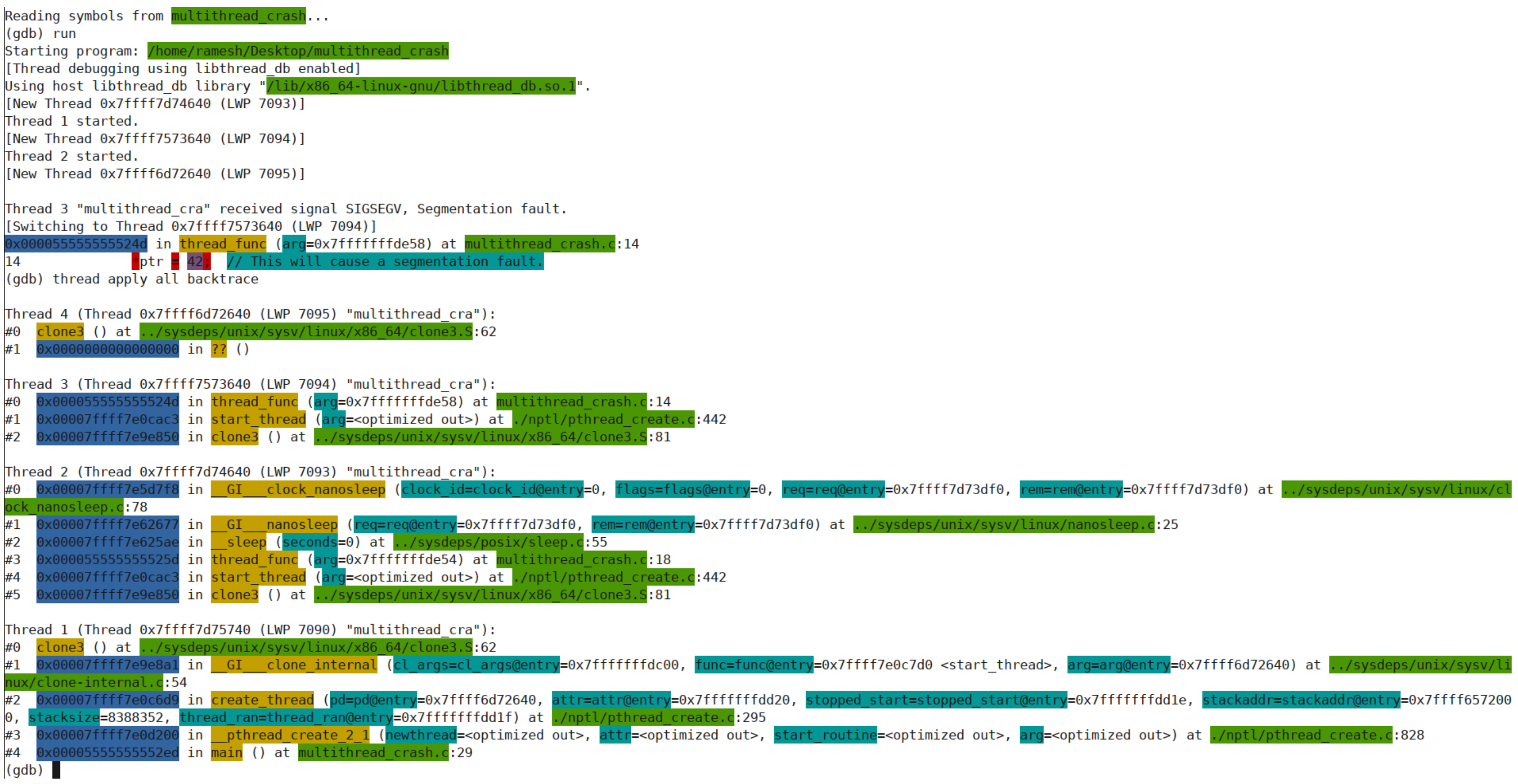
Syntax: `(gdb) thread apply all backtrace`

Explanation:

This command prints the stack trace for all threads in a multi-threaded program, which is essential for debugging concurrent applications.



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ cat multithread_crash.c
// multithread_crash.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

void* thread_func(void* arg) {
    int id = *(int*)arg;
    printf("Thread %d started.\n", id);

    // Introduce a deliberate segmentation fault in thread 2.
    if (id == 2) {
        int *ptr = NULL;
        *ptr = 42;  // This will cause a segmentation fault.
    }

    // Sleep to simulate work
    sleep(2);
    printf("Thread %d ending.\n", id);
    return NULL;
}

int main() {
    pthread_t threads[3];
    int thread_ids[3] = {1, 2, 3};

    // Create three threads.
    for (int i = 0; i < 3; i++) {
        if (pthread_create(&threads[i], NULL, thread_func, &thread_ids[i]) != 0) {
            perror("pthread_create");
            exit(EXIT_FAILURE);
        }
    }

    // Wait for all threads to finish.
    for (int i = 0; i < 3; i++) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```



```
Reading symbols from multithread_crash...
(gdb) run
Starting program: /home/ramesh/Desktop/multithread_crash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff7d74640 (LWP 7093)]
Thread 1 started.
[New Thread 0x7ffff7573640 (LWP 7094)]
Thread 2 started.
[New Thread 0x7ffff6d72640 (LWP 7095)]

Thread 3 "multithread_cra" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7ffff7573640 (LWP 7094)]
0x000055555555524d in thread_func (arg=0x7fffffffde58) at multithread_crash.c:14
14              *ptr = 42;  // This will cause a segmentation fault.
(gdb) thread apply all backtrace

Thread 4 (Thread 0x7ffff6d72640 (LWP 7095) "multithread_cra"):
#0  clone3 () at ../sysdeps/unix/sysv/linux/x86_64/clone3.S:62
#1  0x0000000000000000 in ?? ()

Thread 3 (Thread 0x7ffff7573640 (LWP 7094) "multithread_cra"):
#0  0x000055555555524d in thread_func (arg=0x7fffffffde58) at multithread_crash.c:14
#1  0x00007ffff7e0cac3 in start_thread (arg=<optimized out>) at ./nptl/pthread_create.c:442
#2  0x00007ffff7e9e850 in clone3 () at ../sysdeps/unix/sysv/linux/x86_64/clone3.S:81

Thread 2 (Thread 0x7ffff7d74640 (LWP 7093) "multithread_cra"):
#0  0x00007ffff7e5d7f8 in __GI___clock_nanosleep (clock_id=clock_id@entry=0, flags=flags@entry=0, req=req@entry=0x7ffff7d73df0, rem=rem@entry=0x7ffff7d73df0) at ../sysdeps/unix/sysv/linux/cl
ock_nanosleep.c:78
#1  0x00007ffff7e62677 in __GI___nanosleep (req=req@entry=0x7ffff7d73df0, rem=rem@entry=0x7ffff7d73df0) at ../sysdeps/unix/sysv/linux/nanosleep.c:25
#2  0x00007ffff7e625ae in __sleep (seconds=0) at ../sysdeps/posix/sleep.c:55
#3  0x000055555555525d in thread_func (arg=0x7fffffffde54) at multithread_crash.c:18
#4  0x00007ffff7e0cac3 in start_thread (arg=<optimized out>) at ./nptl/pthread_create.c:442
#5  0x00007ffff7e9e850 in clone3 () at ../sysdeps/unix/sysv/linux/x86_64/clone3.S:81

Thread 1 (Thread 0x7ffff7d75740 (LWP 7090) "multithread_cra"):
#0  clone3 () at ../sysdeps/unix/sysv/linux/x86_64/clone3.S:62
#1  0x00007ffff7e9e8a1 in __GI___clone_internal (cl_args=cl_args@entry=0x7fffffffdc00, func=func@entry=0x7ffff7e0c7d0 <start_thread>, arg=arg@entry=0x7ffff6d72640) at ../sysdeps/unix/sysv/li
nux/clone-internal.c:54
#2  0x00007ffff7e0c6d9 in create_thread (pd=pd@entry=0x7ffff6d72640, attr=attr@entry=0x7fffffffdd20, stopped_start=stopped_start@entry=0x7fffffffdd1e, stackaddr=stackaddr@entry=0x7ffff657200
0, stacksize=8388352, thread_ran=thread_ran@entry=0x7fffffffdd1f) at ./nptl/pthread_create.c:295
#3  0x00007ffff7e0d200 in __pthread_create_2_1 (newthread=<optimized out>, attr=<optimized out>, start_routine=<optimized out>, arg=<optimized out>) at ./nptl/pthread_create.c:828
#4  0x00005555555552ed in main () at multithread_crash.c:29
(gdb) 
```

Explanation

- `thread apply all backtrace`: This command tells gdb to run the `backtrace` command for every thread. This is extremely helpful in multi-threaded applications to see the call stacks for all active threads at the moment of the crash.
- Interpreting the Output:
  - Each block corresponds to a thread.
  - For example, Thread 2 shows the call stack leading up to the segmentation fault in `thread_func`.
  - Other threads show what they were doing (for example, sleeping or waiting).

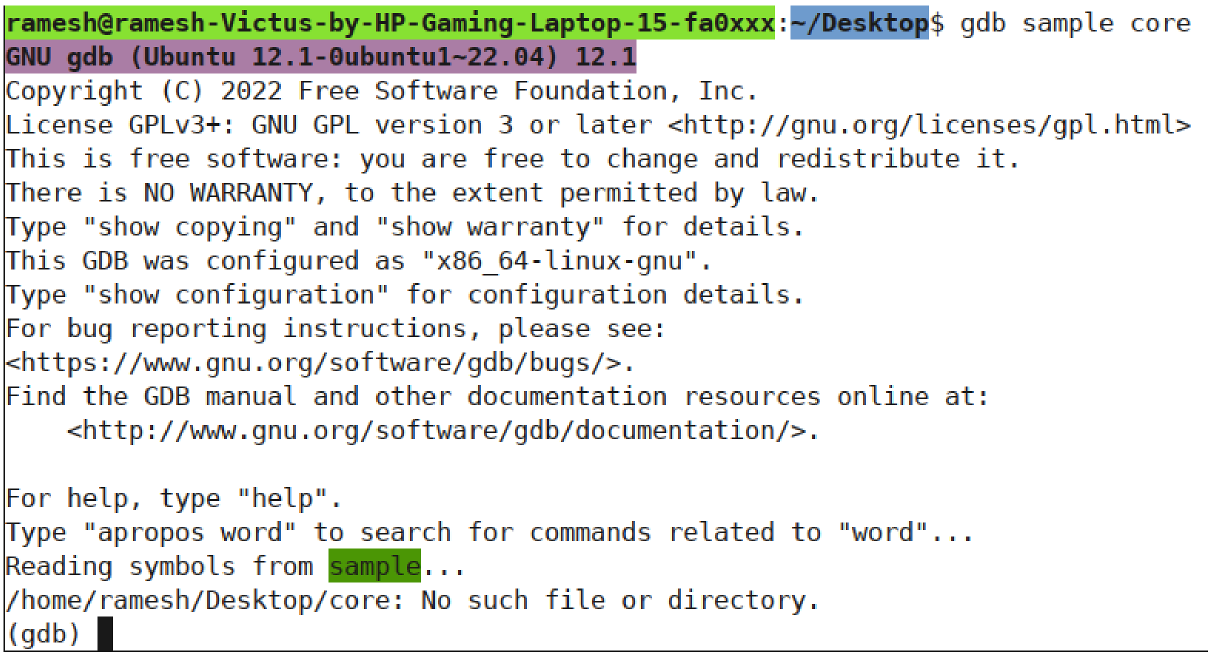
This demonstration illustrates how to use `thread apply all backtrace` to effectively debug multi-threaded applications by obtaining backtraces for all threads simultaneously.

15. Core File Debugging

Syntax: `gdb sample core`

Explanation:

This command starts GDB with the executable sample and a core dump file (core). It allows you to inspect the state of the program at the time of a crash.



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gdb sample core
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 "x86_64-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"...
Reading symbols from sample...
/home/ramesh/Desktop/core: No such file or directory.
(gdb) 
```

16. Debugging an Already Running Program

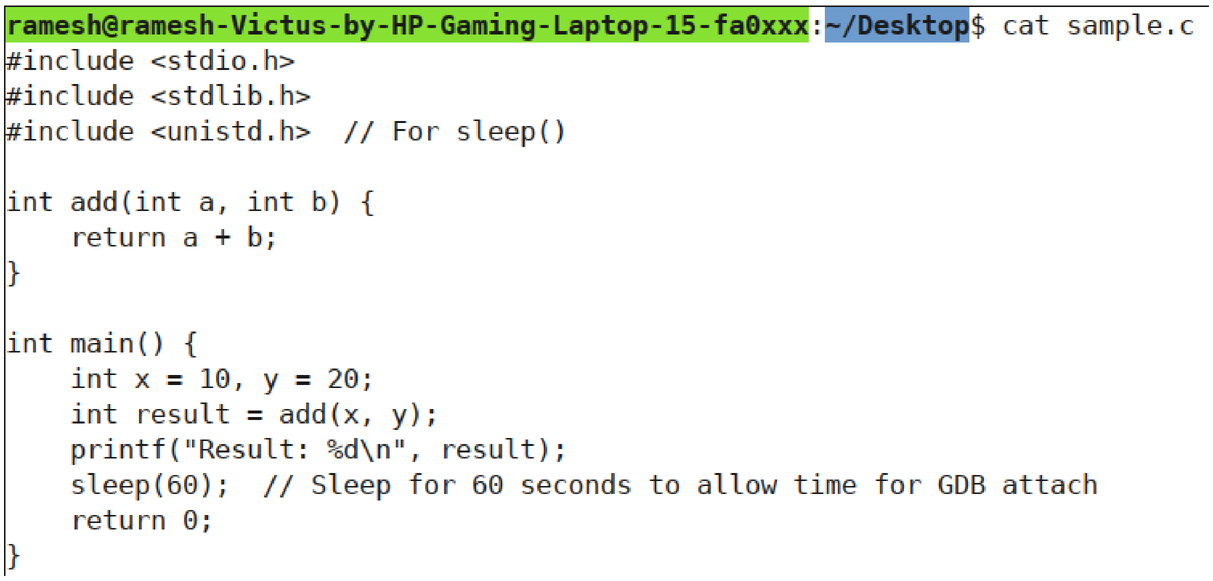
Syntax: `(gdb) attach <process-id>`

Explanation:

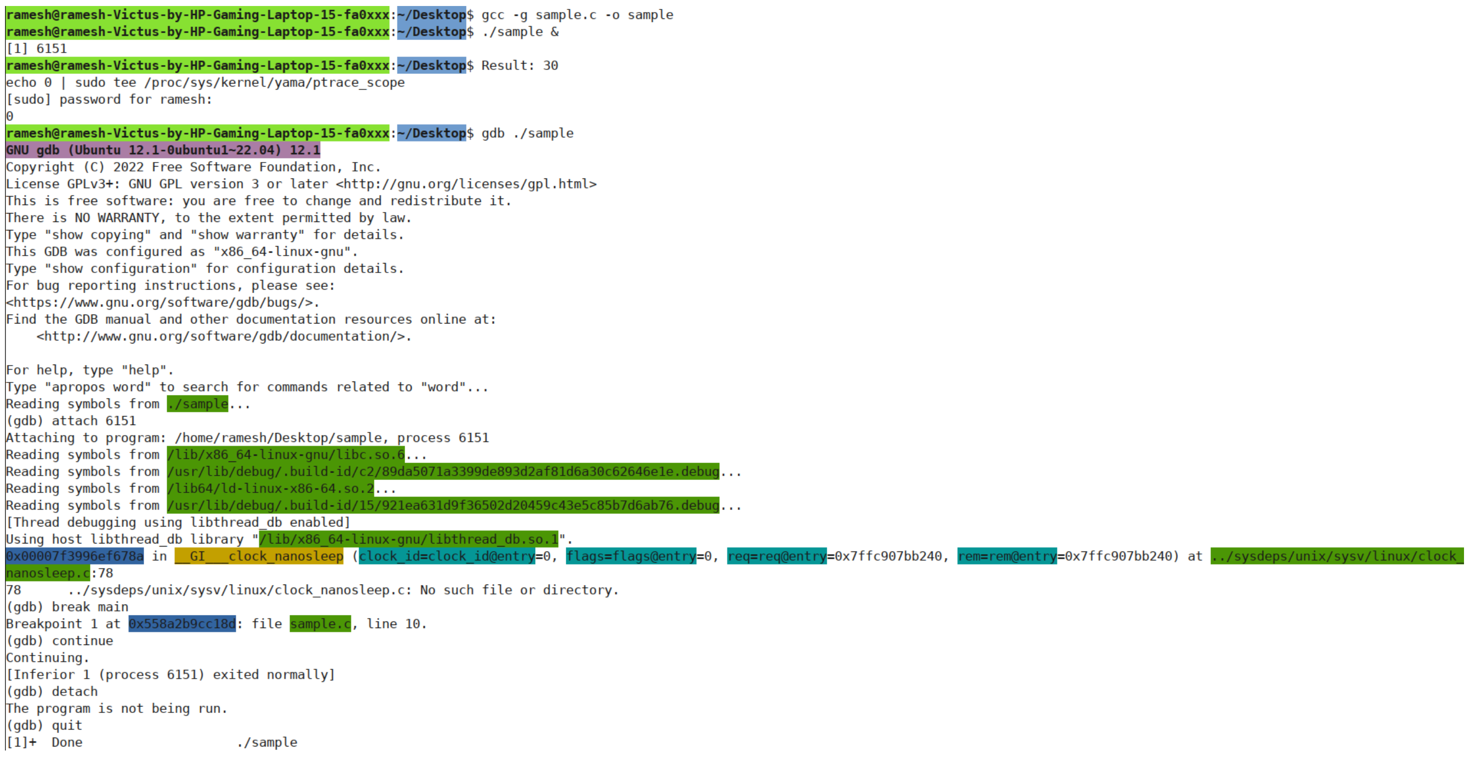
Attaches GDB to a running process identified by its process ID (PID). This is useful for live debugging without restarting the program.

Notes:

- Ensure you attach before the `sleep(60)` ends.
- The `ptrace_scope` change requires `sudo` privileges. If unavailable, consult your system administrator.



```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ cat sample.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  // For sleep()

int add(int a, int b) {
    return a + b;
}

int main() {
    int x = 10, y = 20;
    int result = add(x, y);
    printf("Result: %d\n", result);
    sleep(60);  // Sleep for 60 seconds to allow time for GDB attach
    return 0;
}
```



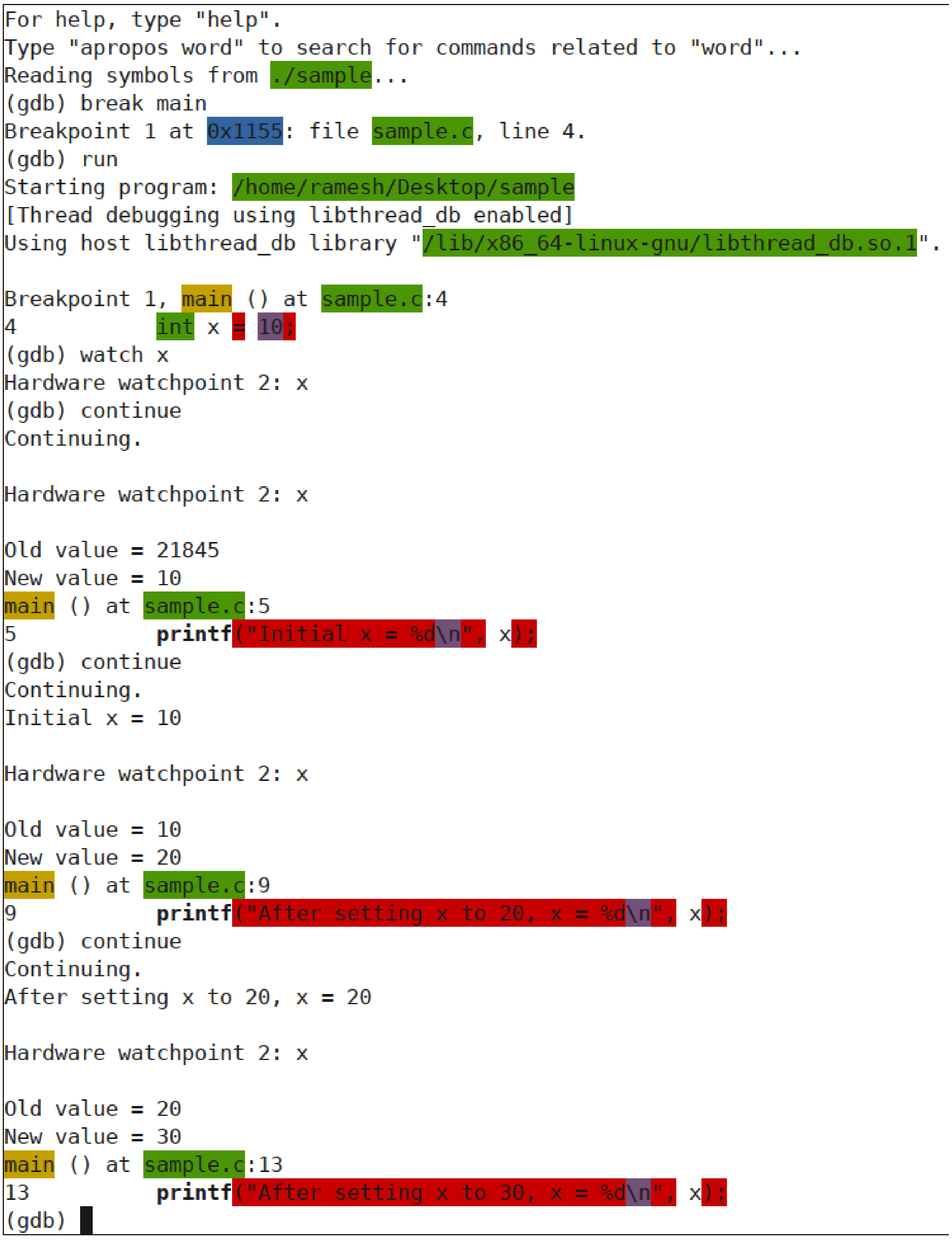
```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gcc -g sample.c -o sample
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ ./sample &
[1] 6151
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ Result: 30
echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope
[sudo] password for ramesh:
0
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ gdb ./sample
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 "x86_64-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"...
Reading symbols from ./sample...
(gdb) attach 6151
Attaching to program: /home/ramesh/Desktop/sample, process 6151
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".
0x00007f3996ef678a in __GI___clock_nanosleep (clock_id=clock_id@entry=0, flags=flags@entry=0, req=req@entry=0x7ffc907bb240, rem=rem@entry=0x7ffc907bb240) at ../sysdeps/unix/sysv/linux/clock_
nanosleep.c:78
78      ../sysdeps/unix/sysv/linux/clock_nanosleep.c: No such file or directory.
(gdb) break main
Breakpoint 1 at 0x558a2b9cc18d: file sample.c, line 10.
(gdb) continue
Continuing.
[Inferior 1 (process 6151) exited normally]
(gdb) detach
The program is not being run.
(gdb) quit
[1]+  Done                    ./sample
```

17. Advanced Concept: Watchpoints

Syntax: `(gdb) watch x`

Explanation:

Sets a watchpoint on variable x. GDB will halt execution whenever the value of x changes, helping to locate the exact point of modification.



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./sample...
(gdb) break main
Breakpoint 1 at 0x1155: file sample.c, line 4.
(gdb) run
Starting program: /home/ramesh/Desktop/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at sample.c:4
4            int x = 10;
(gdb) watch x
Hardware watchpoint 2: x
(gdb) continue
Continuing.

Hardware watchpoint 2: x

Old value = 21845
New value = 10
main () at sample.c:5
5            printf("Initial x = %d\n", x);
(gdb) continue
Continuing.
Initial x = 10

Hardware watchpoint 2: x

Old value = 10
New value = 20
main () at sample.c:9
9            printf("After setting x to 20, x = %d\n", x);
(gdb) continue
Continuing.
After setting x to 20, x = 20

Hardware watchpoint 2: x

Old value = 20
New value = 30
main () at sample.c:13
13           printf("After setting x to 30, x = %d\n", x);
(gdb)
```

Conclusion

This document provided a detailed walkthrough of key GDB commands with practical examples using a simple C program.

By following these steps, you can debug your programs effectively using GDB.

Remember to capture real screenshots during your debugging session and update the document accordingly before submitting your assignment.