21CS8056
Sayan Paul

Compiler Design Lab-1

Assignment 7: Working with GNU GDB (the GNU Debugger) (Assignments for 6th and 13th March 2024)

Suvrojit Das
•
Mar 6 (Edited Mar 6)
100 points

Find attached a very good introduction to using GDB by Dr. Abhijit Das and Dr. Arobindo Gupta of IIT Kharagpur.
A standard place to learn everything about GNU GDB is here: https://www.sourceware.org/gdb/
In case you don't have GDB installed in your system by default, you can download and build of your own: https://ftp.gnu.org/gnu/gdb/
And of course you have a very rich wiki page! :https://en.wikipedia.org/wiki/GNU_Debugger

---

Q1. Write a program with deliberate errors like accessing an uninitialized variable or division by zero. Use GDB to set breakpoints, examine variables and understand the cause of the error during program execution.

CODE :

```c
#include<stdio.h>

int main(){
   int a,b;
   a=10;
   b=0;
   a=a/b;
   printf("%d\n",a);
   return 0;
}
```

OUTPUT :

1. gcc -g divideByZero.c -o divzero
2. gdb ./divzero
3. break 7
4. run

```
Breakpoint 1, main () at divby0.c:7

7     a=a/b;

(gdb) s

Program received signal SIGFPE, Arithmetic exception.
```

Q2. Write a program using both recursive and non-recursive function calls. Use GDB to step through the function calls, understand the call stack and inspect arguments and return variables.
(Helper Links: Function Calls and the Call Stack AND Function Call Stack Examples)

CODE :

```c
#include <stdio.h>

int factorial(int n) {
     return ((n>0)? (n * factorial(n-1)) :1);
}

int fibbonacci(int n) {
  if(n == 0 || n == 1)
     return n;
  else
     return (fibbonacci(n-1) + fibbonacci(n-2));
}

int main() {
  int n = 5;
  int i;

  printf("Factorial of %d: %d\n" , n , factorial(n));
  printf("Fibbonacci of %d: " , n);

  for(i = 0;i<n;i++) {
     printf("%d ",fibbonacci(i));
  }
}
```

OUTPUT :

1. gcc -g callStack.c -o csk
2. gdb ./csk
3. break 23
4. run

```
--Type <RET> for more, q to quit, c to continue without paging--
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 callFibo...
(gdb) list
9           }
10      }
11
12      int fibbonacci(int n) {
13         if(n == 0){
14            return 0;
15         } else if(n == 1) {
16            return 1;
--Type <RET> for more, q to quit, c to continue without paging--
17         } else {
18            return (fibbonacci(n-1) + fibbonacci(n-2));
(gdb)
(gdb) break 23
Breakpoint 1 at 0x11d2: file callStack.c, line 23.
(gdb) run
Starting program:
/home/student/Desktop/sayan56/cdlab/gbd/callFibo
[Thread debugging using libthread_db enabled]
Using host libthread_db library
"/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at callStack.c:23
23         int n = 5;
(gdb) s
26         printf("Factorial of %d: %d\n" , n , factorial(n));
(gdb) s
factorial (n=5) at callStack.c:5
5          if(n == 0) {
(gdb) s
8             return n * factorial(n-1);
(gdb) s
factorial (n=4) at callStack.c:5
5          if(n == 0) {
(gdb) s
8             return n * factorial(n-1);
(gdb) s
factorial (n=3) at callStack.c:5
5          if(n == 0) {
(gdb) s
8             return n * factorial(n-1);
(gdb) s
factorial (n=2) at callStack.c:5
5          if(n == 0) {
```

```
(gdb) s
8               return n * factorial(n-1);
(gdb)
factorial (n=1) at callStack.c:5
5           if(n == 0) {
(gdb) s
8               return n * factorial(n-1);
(gdb) s
factorial (n=0) at callStack.c:5
5           if(n == 0) {
(gdb) s
6               return 1;
(gdb) s
10      }
(gdb) s
factorial (n=1) at callStack.c:8
8               return n * factorial(n-1);
(gdb) s
10      }
(gdb) s
factorial (n=2) at callStack.c:8
8               return n * factorial(n-1);
(gdb) s
10      }
(gdb) s
factorial (n=3) at callStack.c:8
8               return n * factorial(n-1);
(gdb) s
10      }
(gdb) s
factorial (n=4) at callStack.c:8
8               return n * factorial(n-1);
(gdb) s
10      }
(gdb) s
factorial (n=5) at callStack.c:8
8               return n * factorial(n-1);
(gdb) s
10      }
(gdb) s
__printf (format=0x555555556004 "Factorial of %d: %d\n") at
./stdio-common/printf.c:28
28
```

Q3. Write a program that allocates and manipulates dynamic memory (like malloc and free). Use GDB to check memory leaks and access violations (*eg: Segmentation faults*) by deliberately introducing errors in your memory management code.
(Helper Link: LINUX GDB: IDENTIFY MEMORY LEAKS)

CODE :

```c
#include<stdio.h>
int main(){
   int * ptr = (int *) malloc(1e9);
   free(ptr);
   *ptr =10;
   return 0;
}
```

OUTPUT :

1. gcc -g mem.c -o mem
2. gdb ./mem
3. break 5

```
main () at mem.c:5
5           *ptr =10;
(gdb)

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555193 in main () at mem.c:5
5           *ptr =10;
(gdb) s

Program terminated with signal SIGSEGV, Segmentation fault.
```