GNU Debugger (GDB)

Computing Laboratory

http://www.isical.ac.in/~dfslab/2023

Getting started

- Use `-g` or `-g3` to add debugging information when compiling
  ```
  $ gcc -g3 -Wall -o progx progx.c
  ```
- Starting gdb
  ```
  $ gdb ./progx
  ```
  ⟵ no command line arguments
- Starting the program (with command line arguments, if necessary)
  ```
  (gdb) run argument1 argument2 ...
  ```
- Lookup in-built documentation
  ```
  (gdb) help <command_name>
  ```
- Exiting gdb
  ```
  (gdb) quit
  ```

Example I

```c
#include<stdio.h>
#include<string.h>
int main() {
    char *str;
    printf("Size of the string = %lu", strlen(str));
    return 0;
}
```

```
$ gdb a.out
...
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...done.
(gdb) r
Starting program: ...

Program received signal SIGSEGV, Segmentation fault.
__strlen_avx2 () at ../sysdeps/x86_64/multiarch/strlen-avx2.S:65
65      ../sysdeps/x86_64/multiarch/strlen-avx2.S: No such file or directory.
```

Example II

```
(gdb) where
#0  __strlen_avx2 () at ../sysdeps/x86_64/multiarch/strlen-avx2.S:65
#1  0x00005555555469e in main () at gdb-basic.c:5
(gdb) l
60      in ../sysdeps/x86_64/multiarch/strlen-avx2.S
(gdb) up
#1  0x00005555555469e in main () at gdb-basic.c:5
5           printf("Size of the string = %lu", strlen(str));
(gdb) p str
$1 = 0x0
(gdb)
```

Breakpoints

- `b(reak) main` ← function name (beginning of function)
  `b 5` ← line number
  `b gdb-basic.c:5` ← line number in specified file
  `b +<N>` ← `N` lines after current line
  `b 5 if (str==0)`
- `cond(ition) bpnumber [condition]`
  Set a new condition for the breakpoint, an expression which must evaluate to true before the breakpoint is honored. If condition is absent, any existing condition is removed; i.e., the breakpoint is made unconditional.
- `dis(able) bpnumber`
  `en(able) bpnumber`
- Deleting breakpoints
  `cl(ear) filename:lineno`
  `d(elete) bpnumber1 bpnumber2 ...`
- `tb(reak)` : set temporary breakpoint

Viewing code, expressions

- `l(ist) [first [,last] | .]`
  List source code for the current file. Without arguments, list 11 lines around the current line or continue the previous listing.
  With `.` as argument, list 11 lines around the current line. With one argument, list 11 lines starting at that line. With two arguments, list the given range; if the second argument is less than the first, it is a count.
- `p(rint) <expr>`
  Any valid C expression can be passed.
- `disp(lay) [expression]`
  `undisplay [expression]`
  Display the value of expression *if it changed*, each time execution stops in the current frame. Without expression, list all display expressions for the current frame.
  Do not display expression any more in the current frame. Without expression, clear all display expressions for the current frame.

watchpoints

Executing statements

- `c(ontinue)`
- `n(ext)` (step over)
  Continue execution until the next line in the current function is reached or it returns.
- `s(tep)` (step into)
  Execute the current line and stop at the first possible occasion (either in a function that is called or in the current function).
- Run upto the end of current function and display return value
  `(gdb) finish` or `(gdb) f`
- `unt(il) [lineno]` *(very useful for loops)*
  Without lineno, continue execution until the line with a number greater than the current one is reached. With lineno, continue execution until a line with a number greater or equal to that is reached. In both cases, also stop when the current frame returns.

Examining / navigating the stack

- `w(here)` OR `bt`
  Print a stack trace, with the most recent frame at the bottom. An arrow indicates the current frame, which determines the context of most commands.
- `u(p) [count]`
  Move the current frame count (default one) levels up in the stack trace (to an older frame).
- `d(own) [count]`
  Move the current frame count (default one) levels down in the stack trace (to a newer frame).

Miscellaneous

- Set the value of a locally used variable or argument inside the current function

  `(gdb) set variable <variable_name> = <value>`
- List the locally stored variables for the current function

  `(gdb) info local` or `(gdb) i local`
- List all breakpoints present in the program

  `(gdb) info break` or `(gdb) info b` or
  `(gdb) i b` `(gdb) info args` or `(gdb) i args`
- `<Enter>` : repeat last command
- `q(uit)`