GNU Debugger (GDB)

Computing Lab

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

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 ...
  ```
  - input redirection works
    ```
    (gdb) run < input-file.txt
    ```
- Lookup in-built documentation
  ```
  (gdb) help <command_name>
  ```
- Exiting gdb
  ```
  (gdb) quit
  ```

Example I

```
#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  0x000055555555469e in main () at gdb-basic.c:5
(gdb) l
60      in ../sysdeps/x86_64/multiarch/strlen-avx2.S
(gdb) up
#1  0x000055555555469e 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 breakpoint
    (execution stops at breakpoint iff expression evaluates to true)
  - condition is absent ⇒ any existing condition is removed
    (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

Printing values

`p(rint) <expr>`

- Any valid C expression can be passed.
- C structures and constant-length arrays can be printed as a whole.

```
(gdb) p <name of struct type variable>
(gdb) p <name of constant-length array>
```

- To print multiple elements of an array

```
(gdb) p *array_name@length
```

`array_name` → pointer to start of array
`length` → number of elements to be displayed

- Formatted printing: `(gdb) p/<format character> <expr>`
  - `/x`: hexadecimal
  - `/d`: signed decimal
  - `/u`: unsigned decimal
  - `/c`: character
  - `/f`: Floating-point.

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).
- `f(inish)` (complete executing the current function)
  Run upto the end of current function and display return value
- `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.

Viewing source code

`l(ist) [first [,last] | .]`

- 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.

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).

More specialised commands

Watchpoints / data breakpoints

`watch <expr>`

- Like breakpoints, but execution stops *when value of* `<expr>` *changes*
- `rwatch <variable-name>`
  Set a *read watchpoint* on a particular variable
- `awatch <variable-name>`
  Set a *read/write watchpoint* on a particular variable
- Managing (enabling, disabling, deleting) watchpoints: as for breakpoints

  `delete <watchpoint-number>`

  `disable <watchpoint-number>`

  `enable <watchpoint-number>`

Automatically printing values

```
disp(lay) <expr>
```

- Add expression to *automatic display list*
  (values printed *each time the program stops*)
- If no expression is provided, list *all* expressions (serial number and current value) on automatic display list
- Managing the automatic display list: as for breakpoints

```
undisplay dnumber1 dnumber2 ...
delete display dnumber1 dnumber2 ...
disable display dnumber1 dnumber2 ...
enable display dnumber1 dnumber2 ...
```

Printing type information

- `whatis <expr or type-name>`
  - for expressions, prints type as defined in source code
    - details of `struct` not displayed, `typedefs` not unrolled
    - expression is **not** evaluated ($\Rightarrow$ no side-effects)
  - for type-names, at most *one* level of `typedef` unrolled
- `ptype <expr or type-name>`
  like `whatis`, but prints more detailed description of the type
- `explore <expr or type-name>`
  like `whatis, ptype`, but prints full details interactively

Miscellaneous

- Set the value of a locally used variable or argument inside the current function
  `(gdb) set variable <variable_name> = <value>`
- List all breakpoints (and watchpoints) present in the program
  `(gdb) info break` OR `(gdb) info b` OR `(gdb) i b`
- List the locally stored variables for the current function
  `(gdb) info local` OR `(gdb) i local`
- List the arguments of the current function
  `(gdb) info args` OR `(gdb) i args`
- List expressions that are displayed automatically
  `(gdb) info display` OR `(gdb) i display`
- `<Enter>` : repeat last command
- `q(uit)`

Additional resources, cheatsheets, etc.

- https://ucsb-cs24.github.io/m19/lectures/GDB-cheatsheet.pdf
- http://www.yolinux.com/TUTORIALS/GDB-Commands.html
- https://github.com/reveng007/GDB-Cheat-Sheet
- https://bytes.usc.edu/cs104/wiki/gdb