CS 241
Data Organization
Using GDB

September 12, 2018

What is GDB?

- The GNU debugger
- Allows you to inspect the program during execution
- Works for several languages, including C and C++

Compiling for Debugging

When you are going to use the debugger, compile your code with the -g option to include debugging information in your executable.

```
gcc -g -o myprog myprog.c
```

Compiling with picky flags would look like:

```
gcc -Wall -ansi -pedantic -g -o myprog myprog.c
```

Makefiles will help save you typing.

Starting gdb

- Generally, you'll start gdb specifying the program to debug.

  ```
  > gdb myprog
  (gdb)
  ```

- Alternatively, you can specify the program after starting the debugger.

  ```
  > gdb
  (gdb) file myprog
  ```

- Use the `quit` command to exit.

Getting help with gdb commands

- gdb is an interactive shell, similar to the shell you use in a linux terminal.
    - Recall history with arrow keys
    - Auto-complete with TAB
    - Give short versions of commands
- If you need more information while using the debugger, use the `help` command.
- For information on a particular command, use `help` *`commandname`*

Running the program

- Run the program with the `run` command.
- You can give command line arguments to the program here.
- If program is runs normally outside of debugger, it should run fine here, too.
- If program crashes, you'll get useful information about where it crashed.

Segfault example

```c
 1  #include <stdio.h>
 2
 3  int main()
 4  {
 5    char *str = "value";
 6    int i;
 7
 8    str[3] = 'x';
 9
10    for(i = 0; i < 5; i++)
11    {
12      printf("%c\n", str[i]);
13    }
14    return 0;
15  }
```

```
Program received signal SIGSEGV, Segmentation fault.
0x000000000040050c in main () at str-broken.c:8
8    str[3] = 'x';
```

Breakpoints

- You can set a breakpoint at a given line or function with the `break` command.
  - `break 21`
  - `break myfile.c:32`
  - `break myfunction`
- You can set as many breakpoints as you want.
- If the program reaches a breakpoint while running, it will pause and prompt you for another command.

Reached a breakpoint, now what?

- Resume until next breakpoint with `continue`
- Use `step` to execute the next line of code, possibly entering another function.
- Use `next` to execute the next line of code, treating function call as single line.

Inspecting data

- The `print` command prints the value of an expression.
- Use to inspect value of variables.
- Can dereference pointers, access array elements, etc.

Where am I?

- Use `list` to display source code around the currently suspended line.
- Use `backtrace` to show the current stack.

Watchpoints

- Watchpoints pause the program whenever a watched variable's value is modified.
- Use `watch myvar` to start watching a `myvar`
- Whenever `myvar`'s value changes, the program will pause and print out the old and new values.

Conditional breakpoints

- Perhaps you know the problem only happens under a certain condition.
- You can create a `conditional breakpoint` that will only trigger a condition is true.
- `(gdb) break 6 if i == 10` will pause on line 6 only if the value of the variable i is equal to 10.

Wrong result example

```c
#include <stdio.h>

int factorial(int n)
{
  int result = 1;
  while(n--)
  {
    result *= n;
  }
  return result;
}

int main()
{
  int n = 5;
  int fact = factorial(5);
  printf("%d! = %d\n", n, fact);
  return 0;
}
```

Recursive example

```c
#include <stdio.h>

int fib(int n)
{
  if(n < 2) return 1;
  else return fib(n-1) + fib(n-2);
}

int main()
{
  int n = 5;
  printf("fib(%d) = %d\n", n, fib(n));
  return 0;
}
```