Advanced Programming in the UNIX Environment

Week 05, Segment 11: Unix Development Tools: Using `gdb(1)`, Part III

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

```
Quit anyway? (y or n) y
apue$ vim main.c
apue$ cc -g main.c buf.c
main.c: In function 'main':
main.c:8:3: warning: implicit declaration of function 'fprintf' [-Wimplicit-func
tion-declaration]
   fprintf(stderr, "Usage: %s num\n", argv[0]);
   ^~~~~~~
main.c:8:3: warning: incompatible implicit declaration of built-in function 'fpr
intf'
main.c:8:3: note: include '<stdio.h>' or provide a declaration of 'fprintf'
main.c:8:11: error: 'stderr' undeclared (first use in this function)
   fprintf(stderr, "Usage: %s num\n", argv[0]);
           ^~~~~~
main.c:8:11: note: each undeclared identifier is reported only once for each fun
ction it appears in
apue$ vim main.c
apue$ cc -g main.c buf.c
ld: /tmp//ccr1wQBW.o: in function `printBufs':
/home/jschauma/apue-code/05/gdb-examples/buf.c:16: warning: warning: this progra
m uses gets(), which is unsafe.
apue$ ./a.out
Usage: ./a.out num
apue$
```

Terminal — 80×24

```
10              char *buf = malloc(n);
11              char *buf2 = malloc(8);
12              char *buf3 = malloc(8);
13
14              strcpy(buf2, DATA2);
15              strcpy(buf3, DATA3);
16              gets(buf);
17
(gdb) quit
A debugging session is active.

        Inferior 1 [process 5459] will be killed.

Quit anyway? (y or n) y
apue$ vim buf.c
apue$ cc -g main.c buf.c
ld: /tmp//ccX8vyph.o: in function `printBufs':
/home/jschauma/apue-code/05/gdb-examples/buf.c:25: warning: warning: this progra
m uses gets(), which is unsafe.
apue$ ./a.out 1024000000000000
a.out: malloc: Cannot allocate memory
apue$ ./a.out -1
a.out: malloc: Cannot allocate memory
apue$
```

Using a debugger

The purpose of a debugger such as `gdb(1)` is to allow you to see what is going on "inside" another program while it executes or what it was doing at the moment it crashed.

- `gdb(1)` can display and track code across multiple source files
- detailed errors can only be provided if debugging symbols are present
- we are able to *change* variables *while the program is executing*