CS246
Unix:gdb
C:why free? generics, unions

April 8

midterm 2

- Overall Average 82.6

	Average	Std Dev
Q1	15.87	2.58
Q2	15.95	3.94
Q3	17.73	2.41
Q4	17.52	2.27
Q5	15.52	3.94

GDB

- “Gnu DeBugger”
- Allows you to inspect program while running
  - breakpoints
  - conditional breakpoints
- Another way to attack segmentation faults
  - arguably better
- Debuggers arguably give a lot more flexibility than print statements

A Program that breaks

- gdb loves line numbers
  - cat -n xxx.c

- Program has three issues

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void smashing() {
 5      int aa[10];
 6      for (int i = 0; i < 20; i++) {
 7          aa[i] = i;
 8      }
 9  }
10  int main(int argc, char const *argv[])
11  {
12      int strt = atoi(argv[1]);
13      int aa[strt];
14      smashing();
15      for (int i = 0; i < 1000; i++)
16      {
17          printf("%d %d\n", i, aa[i]);
18      }
19      return 0;
20  }
```

gbd usage

- gcc
  - compile with -g flag
    - like valgrind
- UNIX> gdb executable
  - Equivalently
    - UNIX> gdb
    - (gdb) file executable
  - like valgrind

- Does not start the program!

```
[gtowell@powerpuff L14]$ gcc -g broken.c
[gtowell@powerpuff L14]$ gdb a.out
GNU gdb (GDB) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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"...
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb)
```

gdb help

- gdb is interactive and runs its own shell-like thing
  - tab completion for commands
  - file name completion
  - help
- (gdb) help [command]
  - There are 100’s of gdb commands

gdb basic usage

- quit
  - exit gdb
- run
  - runs the program without args
- run arg1 arg2 …
  - exactly like UNIX> executable arg1 arg2 …

-

gdb breakpoints

- places where the program execution will stop
  - you can set as many as you want

- by line number
  - (gdb) break filename:linenumber
  - if only a single file can omit filename
  - gdb broken.c:12
- by function:
  - (gdb) break smashing
    - no filename since function names must be unique in C
      - at least in “local” files

gdb doing things at a pause

- (gdb) continue
  - resume program execution
- (gdb) step
  - advance one line in program
  - will go into called functions
- (gdb) next
  - does not go into called functions
    - other debuggers call this “step over”
- (gdb)<ENTER> repeat last command

gdb — inspecting when program paused

- to look at the value of a variable when program is paused
    - (gdb) print varName
- (gdb) watch varName
    - program pauses whenever named var changes!
- backtrace
    - how did I get here?

Conditional breakpoints, etc

- (gdb) break 12 if i>10

- gdb has LOTs more it can do
    - for instance, go backward!

Garbage Collection

- Why doesn't java need free?

- Reference counting
    - ls -l
        - mark and sweep

- Java popping items from a stack and garbage collection

- More details https://en.wikipedia.org/wiki/Reference_counting

Unions

- Are all about saving space.
    - If you do not care, then don't
- Similar to structs in definition
- BUT unions only store a **single thing**
- Idea
    - have a situation in which you store either a taxid or a social security number
    - do not need space for both, just one or the other
    - So only allocate space for the largest of the things.
    - At runtime interpret the bits as appropriate for the name

```
file: unioner.c


#include <stdio.h>

typedef union {
    int uA;
    long uB;
    char aa[16];
} aUnion;

int main(int argc, char const *argv[])
{
    aUnion a[4];
    for (int i = 0; i < 4; i++)
        printf("%d %ld\n", i, &a[i]);
    return 0;
}
```

Unions

```
file: union2.c

#define INTEGER 1
#define DOUBLE 2
#define CHAR 3
typedef unsigned
    char unionType;
typedef union
{
    int a;
    double b;
    char c;
} uuu;
typedef struct {
    unionType
        whichOne;
    uuu foo;
} sss;
```

```
sss putint(int ii) {
    sss ret;
    ret.whichOne = INTEGER;
    ret.foo.a = ii;
    return ret;
}

sss putdouble(double dd) {
    sss ret;
    ret.whichOne = DOUBLE;
    ret.foo.b = dd;
    return ret;
}
```

When I use unions I always put them inside a struct and have the struct contain info about what the union holds

```
void printStruct(sss str) {
    switch (str.whichOne) {
        case INTEGER:
            printf("INT %d\n", str.foo.a);
            break;
        case DOUBLE:
            printf("DOU %.2f\n", str.foo.b);
            break;
        }
}

int main(int argc, char const *argv[]) {
    sss aaa[10];
    for (int i = 0; i < 10; i++)
        if (i%2==0)
            aaa[i] = putdouble(i * 5.0);
        else
            aaa[i] = putint(i * 5);
    for (int i = 0; i < 10; i++) {
        printf("%d\n", i);
        printStruct(aaa[i]);
    } return 0; }
```

Generics in C (sort of)

- Recall from java
  - public class AClass<A> { … }
  - AClass aaa = new AClass<String>();

- C does not have that!

- Prior to Generics in java
  - You need to store something — just cast it to Object!
    - C has that!

Generics

- Key observation in C
  - EVERY POINTER IS THE SAME SIZE

- So char *data
  - says that data will contain a pointer to character
  - but the actual pointer is exactly the same thing thing as “int *data”.
  - char and int in the declaration say how to interpret the bits that are pointed at, they say nothing about the pointers.

void *

- void *a;
  - a pointer
    - to something. But it could be anything
    - Before use, it must be typed

  void * malloc(size_t size);

  - This is exactly what malloc returns!
- So to generify a struct just change "xxx *" to "void *"

```c
typedef struct DLLItem {
    char *payload;
    struct DLLItem *next;
    struct DLLItem *prev;
} DLLItem;
```

```c
typedef struct DLLItem {
    void *payload;
    struct DLLItem *next;
    struct DLLItem *prev;
} DLLItem;
```

void *

- One trick.
  - The program still has to know what is being stored so it can interpret the bits correctly.
    - Incorrect casting problems will only show up at runtime
      - GDB!!!
  - Contrast with generics in java that will catch lots of problems in compiler.

- void * automatically casts to (and from) whatever it is set to
  - `char * aaa = ~~(char *)~~malloc(110 * sizeof(char));`

```
int main(int argc, char const *argv[]) {
    int *pi;
    *pi = 5;
    void *vi = pi;
    int *npi = vi;
    printf("%d %d %d\n", *pi, *vi, *npi);
    return 0;
}
```

Will not compile. Cannot get the value of a void pointer

```
int main(int argc, char const *argv[]) {
    int *pi;
    *pi = 5;
    void *vi = pi;
    double *npi = vi;
    printf("%d %.2lf\n", *pi, *npi);
    return 0;
}
```

Seg Fault, why? Print?

LAB

- Get my program broken.c from class web site
- Use GDB to help you find each of the 3 issues.
  - (Yes you can probably find them by inspection)
- Submit screenshot / picture /copy&paste of gdb at a breakpoint around one of these issues.
- More generally, use gdb.