CS221
C and Systems Programming

1

Error handling in C

Don't we all love **Exceptions**?

- Easy to catch or ignore
- Full of useful information
- Consolidate handling in 1 place

Well, C doesn't have that

Error handling in C

Many C system calls return a special value if there is an error

- `read()`
  - 0 means End Of Input
  - -1 means, well, lots of things

`errno` tells us more details about errors in system calls, when they occur

man –s2 read
man strtol – sad…

Error handling in C

How about other kinds of errors?

- `int a = 0; int b = 15/a;`
- `char* c = 19; char d = *c;`

Floating point exception (core dumped)

Segmentation fault (core dumped)

LOOK AT crash.c

Signals

Signals are messages sent our program

- May kill the program, pause the program, or do nothing
- May be raised by error conditions, may be sent by our program to itself, or may be sent from elsewhere

Like **Exceptions**, they are handled "out-of-band"

- Whatever our program was doing gets interrupted
- We can write **Signal Handlers** for most signals

See badmath.c

Signals

```
#include <signal.h>

// Our code to handle some signal
void mySpecialHandler(int sigNumber) { .. code here }

int main() {
    signal(SIGFPE, mySpecialHandler); // install handler
```


Look man –s7 signal for list of signals

Can install different (or same) signalHandler for different signals

ADD sigs.h to crash.c

Signals

Can send signal to our own program using `raise()`

Can send signal to other programs, if we know their **ProcessId**

Can send signal to a program from the command line
- **kill -QUIT 38992**

Can send **SIGABRT** to our program by calling `abort()`
- Kills program with no cleanups
- Usually generates a **corefile**

To use 'kill' we must know pgm's process ID, via "ps"

Signals

Keyboard shortcuts send signals to currently running program

- **ctrl-C: SIGINT**
- **ctrl-Z: SIGTSTP**

Interrupt, Terminal Stop

Is this as good as Exceptions?

No

Can't put handlers in different methods

Can't actually recover from internal errors

Can't handle errors from system calls

Don't have info about what caused the signal

So…when do we use them?

Do cleanups or report info before exiting, when we get **SIGINT**

Handle "alarms" from `alarm()`

Super-quick in-class Lab!

Use alarm() to interrupt ourselves every 5 seconds. Print running time, and restart alarm().
Infinite loop in main(). sleep() if you want.
Handle SIGINT to print "Ouch!"
REVIEW THE MAKEFILE!!

How to stop the program? kill -9 PID. Cannot catch KILL signal