CS 415
Operating Systems Practicum
Spring 2007

Ari Rabkin

Based on slides from Oliver Kennedy

Who am I?

- *Ari Rabkin (asr32@cornell.edu)*
  - *Cornell CS Major/MEng student*
  - *So I've been through 415...*
- *Office Hours: See course web page*
  - *... or by appt: feel free to email*

What do We Expect?

- *You should know some C (or learn quickly)*
- *Six projects turned in on time*
  - *Each project builds on the previous ones*
- *Code meets specification*
  - *Works correctly and efficiently*
- *Don't be afraid to ask questions!!*

Where are we going?

- *Six Projects*
  - *1) Cooperative Multitasking (Thread basics)*
  - *2) Preemptive Multitasking (Preemption)*
  - *3) Unreliable Networking (Datagrams)*
  - *4) Filesystems*
  - *5) Reliable Networking (Streams)*
  - *6) Routing (Path vector protocols)*

Design Document

- *At least a week before project due date, you should meet with course staff, and show them a "design document."*
- *1-2 pages*
- *Describe design choices, data structures, etc.*
- *This is for your benefit, not ours.*

Design doc, etc

- *Include revised design doc with final submission.*
- *Name it Design.pdf or Design.txt, put it in project folder.*
- *If you made significant change since original design, explain why.*
- *Also give test strategy...*

Test strategy

- *Give us a short description (1 paragraph) of how you tested your project, and why we should believe it works.*
- *No credit for "I ran the given test progs."*
- *Most systems bugs are hard to find; do stress/endurance tests. Lots of data, long running.*

Testing

- *Include your test programs with your submission.*
- *Describe them (succinctly) in final design doc/test strategy*
- *Should be well thought out; need not be long or time-consuming to write.*

Grading projects

- *Grade made up of several parts: test results, code review, design document, test strategy.*
  - *5% for test strategy, 10% for design*
  - *Rest is a mix of code review and testing*

Random Tidbits

- *Class uses CMS.*
  - *http://cms.csuglab.cornell.edu/*
- *2 Class formats (on alternating weeks)*
  - *Project assigned*
  - *Project questions (You grill me)*
- *Grading*
  - *20% per lab OR 12%/16%/20%/24%/28%*

C for Java Programmers

Ari Rabkin

based on lecture slides by Tom Roeder and Oliver Kennedy

Why use C?

- *Prettier than assembly, but close match*
  - *“What you see is what you get”*
    - *Nothing happens behind your back*
  - *Grants low-level access to hardware*
- *You probably know most of it already*
  - *Java inherited a lot of C’s syntax*

Primitives

- *Integer Types: int, short, long*
  - *short(2 bytes) <= int(2/4) <= long(4/8)*
- *Floating Point Types: float, double*
  - *float(4?) <= double(8?)*
- *Character Type: char [signed or unsigned]*
  - *String = character array (ends with '\0')*
  - *You manage storage!*

Control Flow

- *Mostly same as Java*
- *Except that there's no boolean type.*
  - *Loop condition is true if integer expression is nonzero*
- *No exception handling; functions return an error code instead.*
  - *Be sure to check return values*

Control Flow

- *if(...) { ... } else { ... }*
- *while(...) { ... }*
- *for(... ; ... ; ...) { ... }*
- *Functions*
  - *int myFunc(int myVar) { return myVar; }*
  - *myVar = myFunc(4);*
- *Programs start at int main()*

The Enum/Typedef

- *enum maps text in the code to an integer*
  - *enum foo { bar, baz, bat };*
  - *enum foo myVar = bar;*
  - *enum color { blue = 7, green = 137};*
- *typedef creates a new name for a type*
  - *typedef int foo;*
  - *foo myVar = 3;*

The Struct

- *Structures are like mini-classes*
  - *No methods, no inheritance, just variables*
- *struct foo { int bar; int baz; };*
  - *struct foo myVar;*
  - *myVar.bar = 2*
- *typedef struct foo {int bar;} baz;*
  - *baz myVar;*

The Union

- *Syntax is like structs, but only one of the members is defined at a time; member storage overlaps.*
- struct foo { int type; union {int bar; float baz;} };
  - *Typically use unions inside structs*
  - *can refer to either* bar *or* baz, *but not both at same time. Use* type *to find out.*

Arrays

- *Arrays work like they do in java*
  - *... if you know how big the array will be in advance*
  - *...and no .length variable*
  - *Be careful with array lengths*
- *Static Array Sizes:* int myArray[20]
- *Dynamic Array sizes: see malloc*

Pointers

- &var *yields the address of variable var*
- * *dereferences or declares a pointer*
  - int *myPointer = &myIntVar;
  - *myPointer++;
- myPointer = (int *)malloc(sizeof(int))
- free(myPointer)

Pointers (continued)

- *You must call free() on each pointer you get from malloc after you're done!*
- *You can allocate arrays with malloc()*
  - malloc(sizeof(struct foo) * n)
  - *These work like normal arrays.*

Example: Memory

C

```c
int main(int argc, char **argv){
    struct TrackPoint *myTrack = malloc(sizeof(struct TrackPoint));
    updatePoint(myTrack);
    printf("%d, %s\n", myTrack.color, argv[0]);
    free(myTrack);
}
```

Java

```java
public TrackPoint(){
    lastPoint = new MyPoint(0, 0);
}
```

Example: Pointer Usage

```
struct TrackPoint *makeTrackPoint(){
    struct TrackPoint *lastPoint = malloc(sizeof(struct TrackPoint));
    (*lastPoint).x = 0;
    lastPoint->y = 0;
}
```

Special Pointers

- *Anonymous pointers*
  - *void **
  - *Analogous to Java's Object; weak type*
- *Function pointers*
  - int call_me(float a) { return (int)a; }
  - int (*fp)(float) = &call_me;
  - (*fp)(3.0); ..... *or*..... fp(3.0) ;

Parameter Passing

- *Consider:* b = 3; foo(b); printf("%d", b);
- void foo(int a) { a += 2; } // outputs 3
- void foo(int *a) { (*a) += 2; } //outputs 5
- *In Java Objects/Arrays behave like case 2*
- *In C Pointers/Arrays behave like case 2*

Some gotchas

- *Declare all variables at top of function.*
- *Free what you malloc, but only once*
- *Be careful with strings...the library string functions don't manage storage for you.*

Some references

- *The comp.lang.c FAQ [http://c-faq.com/]*
- C Traps and Pitfalls, *by Andrew Koenig*
- The C Programming Language, *by Kernighan and Ritchie (slightly dated)*
- *Many other books...*

Careful...

- *No garbage collection, free what you take*
- *Don't free things that didn't get* malloc*ed*
- *Arrays aren't bounds checked (and no .length)*
- *Variables are initially undefined. (Set pointers to NULL, ints to 0 or whatever)*
- *Check for NULL pointers before each use!*
- *VC2005 is pretty smart. Listen to it.*

The Preprocessor

- *#define FOO 42*
- *#define foo(a,b) (a+b)*
- *#include "myheader.h"*
- *#ifdef / #else / #endif*
  - *#ifdef foo means that if foo is not #defined, everything between that and #else or #endif will be removed by the preprocessor*

Example: Preprocessor

```
#include <stdio.h>
#include "myheader.h"
```

```
//comment the following line out to use #defines for colors
#define USE_ENUM

#ifdef USE_ENUM
enum e_color { red = 0xf00, green = 0x0f0, blue = 0x00f };
typedef enum e_color color;
#else
#define red 0xf00
#define green 0x0f0
#define blue 0x00f
typedef int color;
#end
```

Why don’t more people use C?

- *Explicit memory management is a pain*
  - *Leaks, Accessing freed memory...*
- *Language features dependent on platform*
  - *Size of primitives, Library availability*
- *Limited typechecking*
- *Pointers can be error-prone*

Assignment 1
First part: Queues

Ari Rabkin

Part 1: A Queue

- *Objectives*
  - *Implement a queue with prepend*
  - *Should support Append/Prepend in O(1)*
    - *Linked Lists are ideal for this*
  - *The queue need not be threadsafe...*
    - *... but the rest of the project needs to be aware of this.*

Part 1: A Queue

- *Fill in the blanks:* queue.c/queue.h
- *Define one or more structures in* queue.c
- *The world sees a* queue_t
  - *Just an anonymous pointer*
  - *Use coercion to operate on* queue_t
    - (struct myqueue *)q->last

Memory leaks

- *C has no garbage collector.*
- *Won't reuse memory unless you say* free.
- *Program will use too much memory and crash if you don't.*
- *Run a stress test, use Windows task manager to make sure memory usage is bounded.*

More next week...

- Next week, I'll tell you about the rest of assignment one.
- Let us know if you have questions...
  - See webpage for office hours
- Due Feb 8