## CS 415 Operating Systems Practicum Spring 2007

Ari Rabkin

Based on slides from Oliver Kennedy

#### Who am I?

Ari Rabkin (<u>asr32@cornell.edu</u>)
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) o 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?

- o Six Projects
  - o 1) Cooperative Multitasking (Thread basics)
  - o 2) Preemptive Multitasking (Preemption)
  - o 3) Unreliable Networking (Datagrams)
  - o 4) Filesystems
  - o 5) Reliable Networking (Streams)
  - o 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."

o 1-2 pages

 Describe design choices, data structures, etc.

o 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.

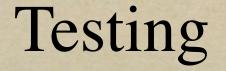
o Also give test strategy...

#### Test strategy

 Give us a short description (1 paragraph) of how you tested your project, and why we should believe it works.

o 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.



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

o Class uses CMS.

o http://cms.csuglab.cornell.edu/ o 2 Class formats (on alternating weeks) • Project assigned • Project questions (You grill me) o Grading o 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?

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

#### Primitives

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

#### **Control Flow**

o 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.

o Be sure to check return values

#### **Control Flow**

and a stand

the top and the manufactor provide the second and the transmithe the second provide the

#### The Enum/Typedef

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

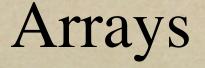
#### The Struct

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

#### The Union

- Syntax is like structs, but only one of the members is defined at a time; member storage overlaps.
- o 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 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

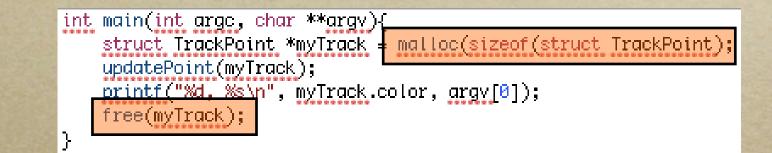
#### Pointers

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



States of p. . Ast to

Java

the stor wasn's

public TrackPoi <u>nt()</u> {			
lastPoint =	new	MyPoint(0,	0);
}			

#### Example: Pointer Usage

Freite.

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

#### **Special Pointers**

o Anonymous pointers o void \* • Analogous to Java's Object; weak type o Function pointers o int call\_me(float a) { return (int)a; }  $\circ$  int (\*fp)(float) = &call\_me; o (\*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 [<u>http://c-faq.com/</u>]
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 malloced
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

o #define FOO 42  $\circ$  #define foo(a,b) (a+b) o #include "myheader.h" o #ifdef / #else / #endif o #ifdef foo means that if foo is not

#ifdef foo means that if foo is not #defined, everything between that and #else or #endif will be removed by the preprocessor

#### Example: Preprocessor

The second of the second profile with the state of the state of

# #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?

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

## Assignment 1 First part: Queues

Ari Rabkin

#### Part 1: A Queue

o Objectives

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

#### Part 1: A Queue

o Fill in the blanks: queue.c/queue.h o Define one or more structures in queue.c o The world sees a queue\_t o Just an anonymous pointer • Use coercion to operate on queue\_t o (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