introduction (lasciate ogni speranza, voi ch'entrate!)

unix weapons school

High-Level Goals

- Train formidably competent systems programmers.
- Ensure GT continues to graduate world-class hackers.
- Provide a fast-paced challenge for talented, driven students.
- Inculcate a profound appreciation of performance.

Low-Level Goals

- Introduce a collection of UNIX development tools.
- Introduce advanced algorithms and data structures.
- Demonstrate analysis-driven optimization.
- Survey the modern x86 instruction set and UNIX kernel.

Assumptions

- You are not required to take this class.
  - This assumption is also an answer to most complaints.
- You are dexterous with C.
  - You ought be able to pump out working C pretty quickly...
  - ... and be able to recognize any C construct.
  - A copy of *The C Programming Language* is recommended.
- You have a 64-bit x86 Linux box readily available.
  - I can't *require* you to install Linux, but...
  - You're CS majors. Why aren't you running open source?
  - We will regularly refer to and modify GNU/Linux source.

Relation to CoC I

UWS relies upon and extends:

- CS2110 (Bill Leahy) – C-based systems
- CS35xx (Dr. STAFF) – Algorithms
- CS325x (Dr. STAFF) – Networking

UWS borrows material from:

- CS6230 (Richard Vuduc) – HPC Tools and Methods
- CS6290 (Hyesoon Kim / Milos Prvulovic) – Comp Arch
- CS6251 (Santosh Pande) – Compiler Design
- Whatever Tom Conte's teaching at the moment

Relation to CoC II

but...
UWS focuses on use of these theories to produce high-performance, robust, practical system software solutions.

e.g.:
You will not gain experience implementing SSA (single static assignment, which if you don't know now, you'll know later), but you will understand the space of optimizations an SSA-driven compiler can effect on your code.

UWS will involve writing ~10,000 lines of code.

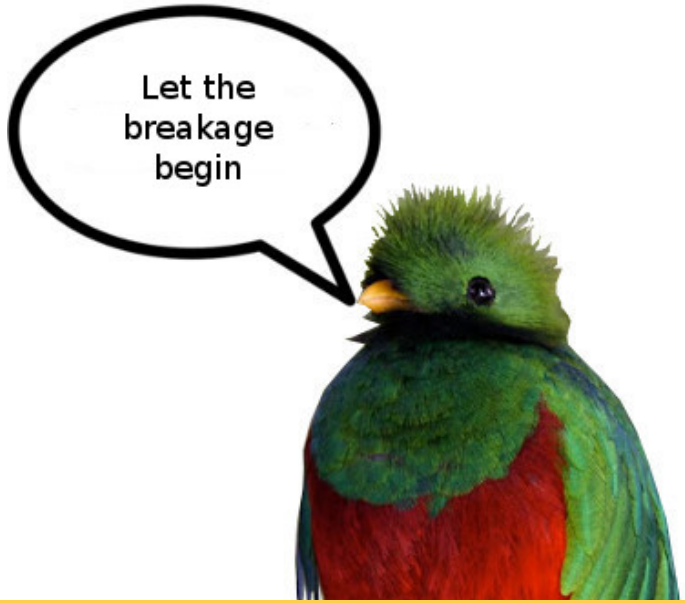
This is Sparta, young friends!

Projects I

Four projects compose the entirety of your grade:

- A high-performance, lock-free allocator
- A parallel DFA/regular expression engine
- A dynamic ELF binary instrumentation tool
- An event-driven compute transaction engine

This list is subject to change.



Projects II

- There are no teams, but you may discuss the project freely among yourselves.
- We will not discuss the projects in class, but they will be covered at length on the mailing list.
- Your internal design freedom is total[1];
  your external design freedom is nil.
- You will have access to reference platforms on which your code will be (automatically) tested.

[1] Save contact of off-system oracles, which is prohibited.

Cheating

If you cheat, I will set you on fire.

Project Grading

- I will prepare U, unit tests for each assignment $A_0 \ldots A_3$.
- For each assignment A_a, for each unit test U_{a_j}, I will:
  - Invoke turnin $T_{a_{i \in S}}$ upon U_{a_j} in a clean environment
  - Sort $|C_u|$ correct projects $0 \ldots n \geq 0$ by decreasing running time
  - If $|C_u| = 1$, award 1.5 points to turnin $T_{a_i} \in C_u$, otherwise

$$\forall i \forall j, T_{a_i} = C_{u_j} \implies \text{award } \frac{j}{|C_u| - 1} \text{ points to } T_{a_i}$$

  - NB: The slowest of multiple correct projects receives 0 points.
- Total points awarded = your score for the project.

Semester Grading

Project scores are totaled yielding $\{p_0, p_1, \ldots p_{|S|-1}\}$.

$G(p)$ maps p to $\{A > B > C > D \cong F\}$ such that[2]

$$\forall i \forall j, p_i > p_j \implies G(p_i) \geq G(p_j),$$

$$\forall i \forall j, p_i = p_j \implies G(p_i) = G(p_j),$$

$$\forall i \forall j, p_i \gg p_j \not\implies G(p_i) > G(p_j)$$

subject to:

$$\forall i, G(p_i) = \{A, D, F\} \implies S_i \text{ really deserved it}$$

with non-binding expectations:

$$|G_D| = 0 \leq |G_F| < |G_A| < \frac{|G_B|}{e} \simeq \frac{|G_C|}{e}$$

[2]Either the "grading function" or "Greenlee function".

Three self-assessment questions

If you know the correct answers, you're in a good position to take this class. You should probably at least know all the words.

- **Q1.** Can adding a NOP (no-operation) instruction accelerate a program?
- **Q2.** Will a program always incur fewer cache misses on a fully-associative cache than a directly-mapped cache of the same size?
- **Q3.** Which of these two programs is likely to complete first on a modern x86?

```
        xor bx,bx
        mov cx,50
erp:
        add bx,1
        add dx,dx
        add ax,ax
        cmp cx,bx
        jg erp
```

```
        xor bx,bx
        mov cx,50
erp:
        add bx,1
        add dx,dx
        add dx,dx
        cmp cx,bx
        jg erp
```

If you're thinking "what the hell's a cache?", you might want to reconsider UWS.

Self-assessment Question I

- **Q1.** Can adding a NOP (no-operation) instruction accelerate a program?
- **A1.** Yes. An example might be when it causes a branch target to be aligned.
- **NB:** This is generally the compiler's job.

Self-assessment Question II

- **Q2.** Will a program always incur fewer cache misses on a fully-associative cache than a directly-mapped cache of the same size?
- **A2.** No. A fully-associative cache will yield fewer hits when data that *is not* reused evicts data which *is* reused, and would otherwise have gone unevicted.
- **NB:** Under the classic Hill definition of a conflict miss as "a miss that would not have occurred in a fully associative cache", this implies that a cache can have a negative number of conflict misses!

Self-assessment Question III

- **Q3.** Which of these two programs is likely to complete first on a modern x86?

```
        xor bx,bx
        mov cx,50
erp:
        add bx,1
        add dx,dx
        add ax,ax
        cmp cx,bx
        jg erp
```

```
        xor bx,bx
        mov cx,50
erp:
        add bx,1
        add dx,dx
        add dx,dx
        cmp cx,bx
        jg erp
```

- **A3.** The program on the left, as it does not carry a long dependency chain as done by the program on the right. Given perfect store forwarding, the two programs might require the same number of cycles.
- **NB:** If your compiler generates either program, get a better compiler.

Introibimus ad altare Dei—our adventure begins!

Let us go then, you and I.

'Tis not too late to seek a newer world.