A Word on the Final Exam

• Disclaimer
  - Past performance is not a guarantee of future results

• The course will change
  - Up to now: basics
    • What you need for Project 3
  - Coming: advanced topics
    • Design issues
    • Things you won't implement
A Word on the Final Exam

• Examination will change to match
  – More design
  – Some things you won't have implemented
Basic Assumptions

• This is a **C** programming class!
  - `sizeof (char) == 1 /* 8 bits */`
  - `sizeof (int) == 4 /* 32 bits, mostly true now */`
  - You need to **really** understand pointers

• Semantics
  - "\0" isn't “just” an 1-byte zero – it's the zero `char`

• Other languages are excellent
  - ...but very few are ok for writing OS code
A&B exams: Q1

• Definitions
• Papers looked mostly ok
A:Q2: malloc() loop *in kernel*

```c
void foo(void)
{
    while (1)
       (void) malloc(1);
}
```

- What we're looking for
  - Vicious memory leak
  - Kernel memory is *limited*
  - When it's gone *no process* can get more
B:Q2: user-mode “summation” loop

```c
void bar(void) {
    int sum = 0, *ip; /*2 decl, 1 set*/
    ip = &sum;
    while (1) { sum += *ip; --ip; }
}
```

- What we're looking for
  - It's infinite stack growth
  - *Eventually* the kernel will kill the process
  - Other processes may suffer – *or be killed!*
A: Q3(a)

- Why *multiple* kernel stacks?
- Not: “protection”!
  - “Monolithic kernel” model: *all* kernel code is trusted
- Key issue: *preemption*
  - User process in kernel mode
  - Stack contains: trap frame, N procedure call frames
  - Switch to another process: copy all that around?
  - From/to where?? Would'n't that be a stack too?
A:Q3(b) - How to set kernel stack size

- Kernel virtual memory size typically fixed, \textit{small}
  - (Why small?)
  - Large stacks means fewer stacks, so fewer threads
- \textit{Not:} “run it and see what happens”
- Good: examine call chains, local variable usage
- Very good: factor in \textit{interrupt handlers}!
B:Q3(a) – How many page faults?

```c
void foo(void) {
    int i; char x[2000];
    for (i = 0; i < 2000; i++)
        x[i] = '\0';
}
```

- What we wanted to see
  - Stack *and code* are involved
  - Page *alignment* may not be ideal even if size fits

- Minimum “good” answer: 4 pages
- Competitive paging: easy to get ~4k/~6k faults
B:Q3(b) – Stack growth, wild access

• “stack growth due to one memory reference”

```c
void foo(void) {
    char x[65536];
    x[0] = '\0';
}
```

• How far is too far? **Not right:** “one page”
B:Q3(b) – Good vs. bad?

```c
void one(void) {
    double a[1024];
    int i;

    for (i = 1024; i >= 0; i--)
        a[i] = 0.0L;
}
```

```c
void one(void) {
    double a[1024];
    int i;

    for (i = 0; i < 1024; i++)
        a[i] = 0.0L;
}
```
B:Q3(b) - Stack growth, wild access

• Also not:
  – Let stack grow *all the way* to top of heap
  – That means *no such thing* as a wild pointer access!

• Reasonable
  – Some *large* size
  – Halfway through the void
Other questions

• Q4 – deadlock
• Q5 – broken mutual exclusion protocol