CS 33

Multithreaded Programming V

Alternatives to Mutexes:
Atomic Instructions

- Read-modify-write performed atomically
- *Lock prefix* may be used with certain IA32 and x86-64 instructions to make this happen
  - lock incr x
  - lock add $2, x
- It's expensive
- It's not portable
  - no POSIX-threads way of doing it
  - Windows supports
    - » InterlockedIncrement
    - » InterlockedDecrement

Alternatives to Mutexes:
Spin Locks

- **Consider**

```
pthread_mutex_lock(&mutex);
new->next = list_ele->next;
list_ele->next = new;
pthread_mutex_unlock(&mutex);
```

- **A lot of overhead is required to put thread to sleep, then wake it up**
- **Rather than do that, repeatedly test mutex until it's unlocked, then lock it**
  - **makes sense only on multiprocessor system**

Compare and Exchange

`cmpxchg src, dest`

- **compare contents of *%rax* with contents of *dest***
  - » **if equal, then *dest* = *src* (and *ZF* = *1*)**
  - » **otherwise *%rax* = *dest* (and *ZF* = *0*)**

Spin Lock

- **the spin lock is pointed to by the first arg (*%rdi*)**
  - **locked is 1, unlocked is 0**

```
        .text
        .globl slock, sunlock
slock:
loop:
        movq $0, %rax
        movq $1, %r10
        lock cmpxchg %r10, 0(%rdi)
        jne loop
        ret
sunlock:
        movq $0, 0(%rdi)
        ret
```


Improved Spin Lock

```
        .text
        .globl slock, sunlock
slock:
loop:
        cmp $0, 0(%rdi)  # compare using normal instructions
        jne loop
        movq $0, %rax
        movq $1, %r10
        lock cmpxchg %r10, 0(%rdi) # verify w/ cmpxchg
        jne loop
        ret
sunlock:
        movq $0, 0(%rdi)
        ret
```

Yet More From POSIX …

```
int pthread_spin_init(pthread_spin_t *s,
      int pshared);
int pthread_spin_destroy(pthread_spin_t *s);
int pthread_spin_lock(pthread_spin_t *s);
int pthread_spin_trylock(pthread_spin_t *s);
int pthread_spin_unlock(pthread_spin_t *s);
```


A Problem ...

- **In thread 1:**

```
if ((ret = open(path,
     O_RDWR) == -1) {
   if (errno == EINTR) {
      ...
   }
   ...
}
```

- **In thread 2:**

```
if ((ret = socket(AF_INET,
     SOCK_STREAM, 0)) {
   if (errno == ENOMEM) {
      ...
   }
```

There's only one errno!

However, somehow it works.

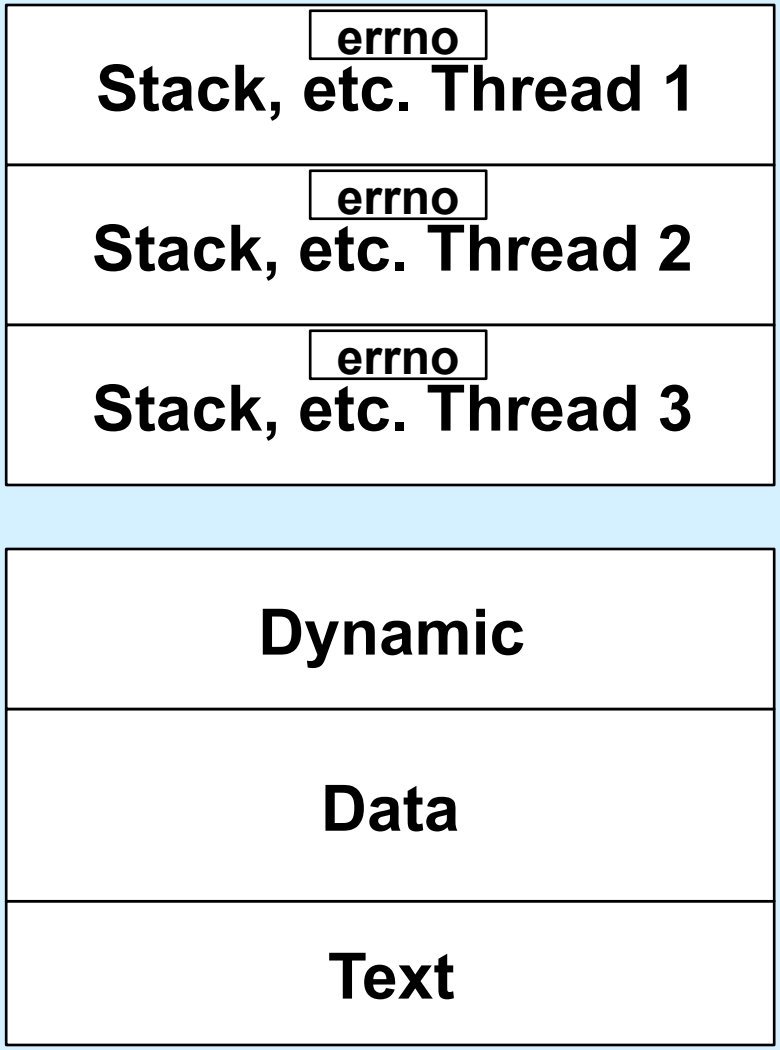
What's done???

A Solution ...

```
#define errno (*__errno_location())
```

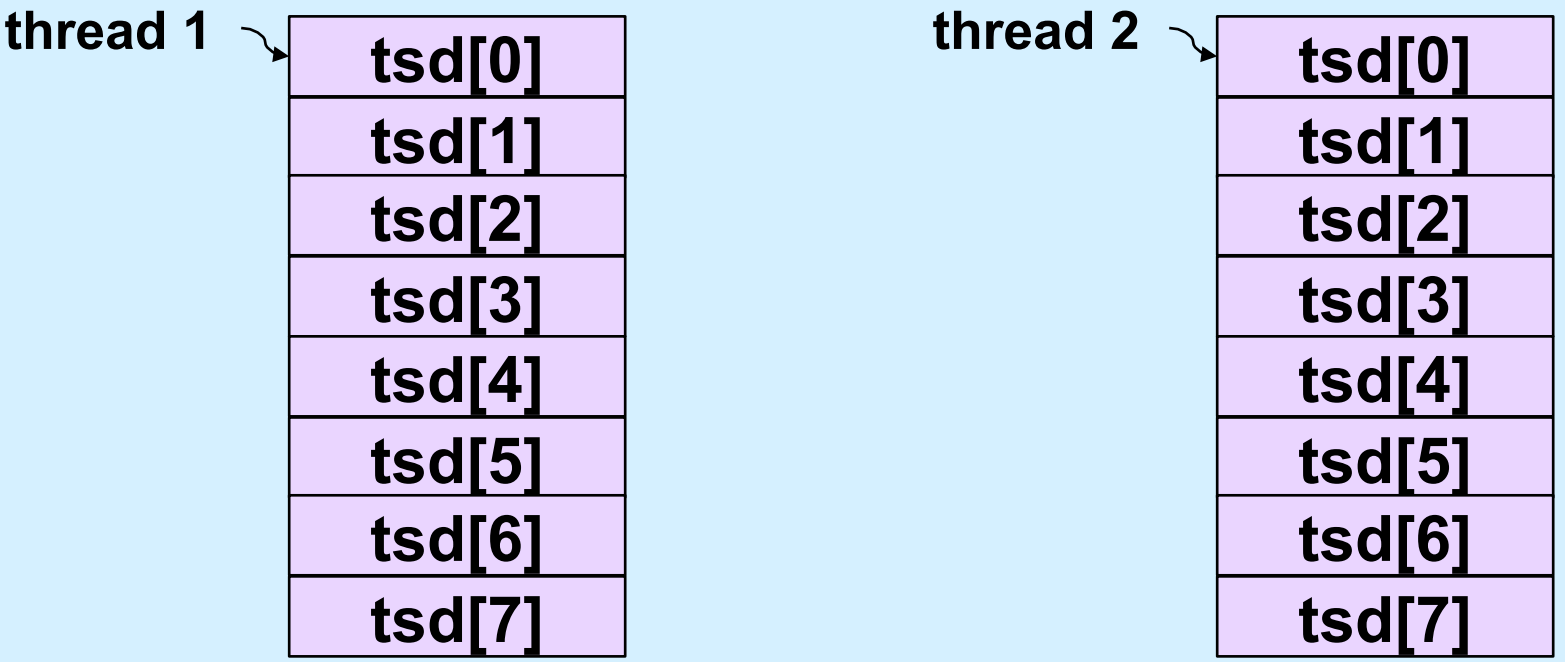
- **__errno_location returns an int * that's different for each thread**
  - **thus each thread has, effectively, its own copy of errno**

Process Address Space



Generalizing

- ***Thread-specific data*** **(sometimes called *thread-local storage*)**
  - **data that's referred to by global variables, but each thread has its own private copy**



Some Machinery

- `pthread_key_create(&key, cleanup_routine)`
  - **allocates a slot in the TSD arrays**
  - **provides a function to cleanup when threads terminate**
- `value = pthread_getspecific(key)`
  - **fetches from the calling thread's array**
- `pthread_setspecific(key, value)`
  - **stores into the calling thread's array**

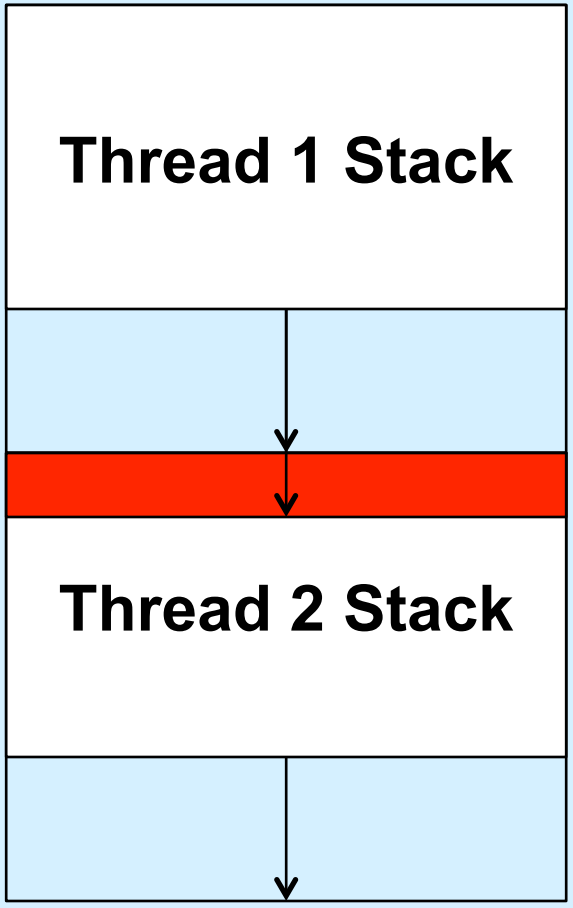
Beyond POSIX
TLS Extensions for ELF and gcc

- **Thread Local Storage (TLS)**

```
__thread int x=6;
  // Each thread has its own copy of x,
  // each initialized to 6.
  // Linker and compiler do the setup.
  // May be combined with static or extern.
  // Doesn't make sense for local variables!
```


Stacks

- Stack overflow
  - generate C code to check for it
    - not done by gcc
  - rely on OS to detect

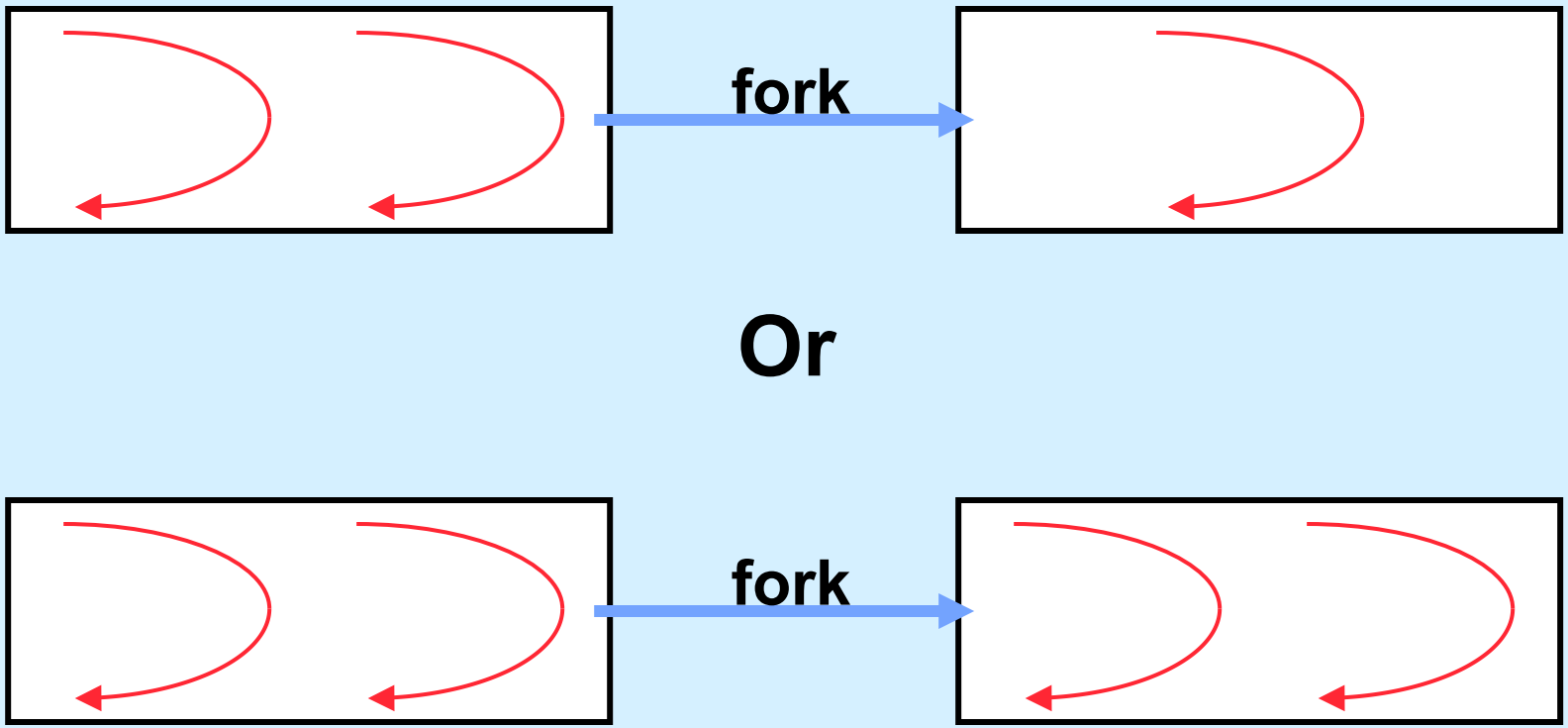


Last Quiz!

- With respect to stack overflow, the OS

  a) can't do anything
  b) can detect it in most cases
  c) can detect it in all cases

Fork and Threads



You'll Soon Finish CS 33 …

- You might
  - celebrate

  - take another systems course
    - » 32
    - » 138
    - » 166
    - » 167

  - become a 33 TA

Systems Courses Next Semester

- CS 32
  - you've mastered low-level systems programming
  - now do things at a higher level
  - learn software-engineering techniques using Java, XML, etc.
- CS 138
  - you now know how things work on one computer
  - what if you've got lots of computers?
  - some may have crashed, others may have been taken over by your worst (and brightest) enemy
- CS 166
  - liked buffer?
  - you'll really like 166
- CS 167/169
  - still mystified about what the OS does?
  - write your own!

The End

Well, not quite …

Database is due on 12/16.

Most malloc rubrics will be released tonight.

Happy coding and happy holidays!

