Logistics

• Any Questions???
Lock Contention.

- When multiple threads are all “banging” on the same lock.
- One solution is to allocate an array of local_counts for use by each thread.
  - As I did in the pi example I handed out.
- This will reduce the lock contention but will not perform optimally.
  - Why?
False Sharing

• **Occurs when 2 or more threads modify different data elements which happen to be in the same cache line!!**

  ➢ Each processor has it’s own cache.
    ▪ Cache coherence is usually a hardware aware operation on SMPs.
  ➢ Thread one modifies local_count[0]
    ▪ Invalidates all copies of local_count cache line
  ➢ Thread two modifies local_count[1]
    ▪ Invalidates all copies of local_count cache line.
  ➢ This will work but it thrashes cache and thus degrades performance.
Solution is to introduce padding

typedef struct {
    int count;
    int_pad[128];
} shared_count;

shared_count *count;

count = (shared_count *)malloc(…);

count[0].count++;
Pthreads

- There is another problem with mutexes
  - Deadlock where a lock will never be obtained because of the code design.
  - A 1-800-OOPS-BUG kind of thing.
  - Deadlock is also possible in message passing so keep this in your brain even when we are done with pthreads 😊
Deadlock with Mutexes

Thread 1

Pthread_mutex_lock(l1);
Pthread_mutex_lock(l2);

Thread 2

Pthread_mutex_lock(l2);
Pthread_mutex_lock(l1);

This is a bad thing; careful planning is needed for lock hierarchies.
Testing if locked

```c
int pthread_mutex_trylock(pthread_mutex_t *mp);
```

- `pthread_mutex_trylock()` is the same as `pthread_mutex_lock()`, except that if the mutex object referenced by `mp` is locked the call returns immediately with an error.

- If `pthread_mutex_trylock()` fails and returns the corresponding value to “EBUSY” if the mutex pointed to by `mp` was already locked.

- Allows you to do other useful work while waiting for a mutex lock.
Destroying a Mutex

\[ \text{int pthread_mutex_destroy(pthread_mutex_t *mp);} \]

- destroys the mutex object referenced by mp
- the mutex object becomes uninitialized.
- The space used by the destroyed mutex variable is not freed.
  - Makes intuitive sense. Only "free()" should be used to free memory space.
- It needs to be explicitly reclaimed.
Pthreads

- There are other synchronization mechanisms that are more complex.
- Condition variables that are user defined.
  - Where do you want to go today?
- Condition variables in reality
  - Waiting on some event (e.g., condition)!
  - Event programming is the CORE of the Windows OS, X-windows applications.
  - How to really stress your brain today!
Condition Variables

• **Are always protected with a mutex.**
  - Why?
  - **Race Conditions 😊**
  - One thread might be testing the condition while another is updating the condition!!!

• **There are implicit and explicit**
  - Locks of mutexes
  - Unlocks of mutexes

• **Explicit**
  - Those you code

• **Implicit**
  - Those the Pthreads library does for you 😊
A diagram of condition variable use

Thread waiting on condition

Thread that sets condition

LOCK

Update Condition

UNLOCK

Wakeup/Signal

Condition?

(lock)

(sleep)

UNLOCK

Continue

Continue

2/21/2005

ComS 425

Spring 2005

12 of 35: Lecture 16
Condition Variable Initialization

```c
int pthread_cond_init(pthread_cond_t *cond,
             const pthread_condattr_t *attr);

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

- **Attributes**
  - PTHREAD_PROCESS_PRIVATE
    - The condition variable can synchronize threads only in this process.
  - PTHREAD_PROCESS_SHARED
    - The condition variable can synchronize threads in this process and other processes.
      - Only one process should initialize the condition variable.
      - The object initialized with this attribute must be allocated in memory shared between processes (e.g., System V shared memory)
      - It is illegal to initialize the object this way and to not allocate it in such shared memory.
        - ♠ Pthreads Cops arrest you
Standard vs. Implementation

• **Warning!!!**
  - Pthreads is a relatively new standard
    - 1990’s
  - All Implementations do not support the full standard
  - Your system man pages are your friend.
  - We talked about PTHREAD_SCOPE_?????
    - ???? Is SYSTEM or PROCESS
    - Linux and Solaris were different defaults.
  - PTHREAD_PROCESS_SHARED/PRIVATE
    - Fine for Solaris
    - Ignored by Linux
Waiting on the Event

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);

• *cond is the condition variable to wait for
• *mutex is a LOCKED mutex that protects the condition variables.
• The wait call implicitly
  ➢ Unlocks the mutex
  ➢ Sleeps until awakened
  ➢ Locks the mutex once awakened.
wait example loop.

- `pthread_cond_wait()` is normally used in a loop testing some condition, as follows:

```c
(void) pthread_mutex_lock(mp);
while (cond == FALSE) {
    (void) pthread_cond_wait(cvp, mp);
}
(void) pthread_mutex_unlock(mp);
```
Timed wait on the Event

```c
int pthread_cond_timedwait(
    pthread_cond_t *cond,
    pthread_mutex_t *mutex,
    const struct timespec *abstime);
```

- `pthread_cond_timedwait()` is the same as `pthread_cond_wait()` except
  - an error is returned if
    - the system time equals or exceeds the time specified by `abstime` before the condition `cond` is signaled or broadcasted or
    - if the absolute time specified by `abstime` has already passed at the time of the call.
  - `ETIMEDOUT` is the error if the time specified by `abstime` has passed.
Timed wait example loop.

- `pthread_cond_timedwait()` is also normally used in a loop testing in some conditions. It uses an absolute timeout value as follows:
  ```c
  timestruc_t to;
  (void) pthread_mutex_lock(mp);
  to.tv_sec = time(NULL) + TIMEOUT;
  to.tv_nsec = 0;
  while (cond == FALSE) {
    err = pthread_cond_timedwait(cvp, mp, &to);
    if (err == ETIMEDOUT) {
      /* timeout, do something */              break;
    }          }
  (void) pthread_mutex_unlock(mp);
  ```

- This example sets a bound on the total wait time even though `pthread_cond_timedwait()` may return several times due to the condition being signaled or the wait being interrupted.
Signaling Waiting Threads

• A condition signal allows a thread to unblock the next thread waiting on the condition variable, whereas, a condition broadcast allows a thread to unblock all threads waiting on the condition variable.

  • int pthread_cond_signal(pthread_cond_t *cond);
    ➢ pthread_cond_signal() unblocks at least one thread blocked on the specified condition variable cond, if any threads are blocked on cond.

  • int pthread_cond_broadcast(pthread_cond_t *cond);
    ➢ pthread_cond_broadcast() unblocks all threads blocked on the condition variable cond.
Signaling Waiting Threads [2]

- Both functions have no effect if there are no threads blocked on cond.
- `pthread_cond_signal()` or `pthread_cond_broadcast()` may be called by a thread regardless of whether it owns the mutex which threads calling a wait function have associated with the condition variable during their waits.

  ➢ However, if predictable scheduling behavior is required, then that mutex should be locked by the thread calling `pthread_cond_signal()` or `pthread_cond_broadcast()`.
Destroying a Condition Variable.

- int pthread_cond_destroy(
  pthread_cond_t *cond);

- pthread_cond_destroy() destroys the condition variable specified by cond.

  ▶ The space for destroying the condition variable is not freed.
Handout

• Take a copy and pass the stack 😊
Simplest Example

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define NUM_THREADS 3
#define TCOUNT 10
#define COUNT_THRES 12
int    count = 0;
int    thread_ids[3] = {0,1,2};
pthread_mutex_t
    count_lock=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t count_hit_threshold=
    PTHREAD_COND_INITIALIZER;
```
Simplest Example [2]

- `int main(void)
  {
    int i;  
    pthread_t threads[3];
    
    pthread_create(&threads[0], NULL, inc_count, (void *)&thread_ids[0]);
    pthread_create(&threads[1], NULL, inc_count, (void *)&thread_ids[1]);
    pthread_create(&threads[2], NULL, watch_count, (void *)&thread_ids[2]);
    
    for (i = 0; i < NUM_THREADS; i++)
      pthread_join(threads[i], NULL);
    return 0;
  }`
void *inc_count(void *idp)
{
    int i=0;  int *my_id = idp;
    for (i=0; i<TCOUNT; i++) {
        pthread_mutex_lock(&count_lock);
        count++;
        printf("inc_counter(): thread \%d, count = \%d, unlocking mutex\n", *my_id, count);
        if (count == COUNT_THRES) {
            printf("inc_count(): Thread \%d, count \%d\n", *my_id, count);
            pthread_cond_signal(&count_hit_threshold);
        }
        pthread_mutex_unlock(&count_lock);
        usleep(1000*(my_id+1));
    }
    return(NULL);
}
Simplest Example [4]

```c
void *watch_count(void *idp) {
    int *my_id = idp;
    printf("watch_count(): thread %d\n", *my_id);
    pthread_mutex_lock(&count_lock);
    while (count < COUNT_THRES) {
        pthread_cond_wait(&count_hit_threshold,
                           &count_lock);
        printf("watch_count(): thread %d, count %d\n", *my_id, count);
    }
    pthread_mutex_unlock(&count_lock);
    return(NULL);
}
```
Simplest Example Output

inc_counter(): thread 0, count = 1, unlocking mutex
watch_count(): thread 2
inc_counter(): thread 1, count = 2, unlocking mutex
inc_counter(): thread 1, count = 3, unlocking mutex
inc_counter(): thread 0, count = 4, unlocking mutex
inc_counter(): thread 0, count = 5, unlocking mutex
inc_counter(): thread 1, count = 6, unlocking mutex
inc_counter(): thread 0, count = 7, unlocking mutex
inc_counter(): thread 1, count = 8, unlocking mutex
inc_counter(): thread 0, count = 9, unlocking mutex
inc_counter(): thread 1, count = 10, unlocking mutex
inc_counter(): thread 0, count = 11, unlocking mutex
inc_counter(): thread 1, count = 12, unlocking mutex
inc_count(): Thread 1, count 12
watch_count(): thread 1, count 12
inc_counter(): thread 0, count = 13, unlocking mutex
inc_counter(): thread 1, count = 14, unlocking mutex
inc_counter(): thread 0, count = 15, unlocking mutex
inc_counter(): thread 1, count = 16, unlocking mutex
inc_counter(): thread 0, count = 17, unlocking mutex
inc_counter(): thread 1, count = 18, unlocking mutex
inc_counter(): thread 0, count = 19, unlocking mutex
inc_counter(): thread 1, count = 20, unlocking mutex
Condition Variable Example

```c
#include <stdio.h>
#include <pthread.h>
define NUM_THREADS 6
define TCOUNT 18
define COUNT_THRES 18

int we_are_done = 0; int count = 0;
int thread_ids[NUM_THREADS];

pthread_mutex_t done_lock=PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t count_lock=PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t count_hit_threshold=PTHREAD_COND_INITIALIZER;
```
int main(void) {
    int i;  pthread_t threads[NUM_THREADS];
    for(i=0;i<NUM_THREADS;i++) thread_ids[i] = i;
    for(i=0;i<(NUM_THREADS-1);i++)
        pthread_create(&threads[i], NULL, inc_count, (void *)&thread_ids[i]);
    pthread_create(&threads[(NUM_THREADS -1)], NULL, watch_count,
                   (void *)&thread_ids[(NUM_THREADS -1)]);
    for (i = 0; i < NUM_THREADS; i++)
        pthread_join(threads[i], NULL);
    return 0;}

Condition Variable Example [3]

```c
void *inc_count(void *idp) {
    int i=0, save_state, save_type; int *my_id = idp;
    for (i=0; i<TCOUNT; i++) {
        pthread_mutex_lock(&done_lock);
        if (we_are_done) {
            printf("inc_count(): thread %d  we are done
                        shutting down (0:%d)\n", *my_id,i);
            pthread_mutex_unlock(&done_lock);return(NULL);    }
        else      pthread_mutex_unlock(&done_lock);
        pthread_mutex_lock(&count_lock);
        count++;
        printf("inc_count(): thread %d, count = %3d, pass=%2d unlocking mutex\n",    *my_id, count,i);
        if (count == COUNT_THRES) {
            printf("inc_count(): Thread %d, count %d\n", *my_id, count);
            pthread_cond_signal(&count_hit_threshold);    }
        pthread_mutex_unlock(&count_lock);
        sleep((unsigned) (NUM_THREADS + 1 - *my_id));
    }   return(NULL); }   
```
Condition Variable Example [4]

```c
void *watch_count(void *idp) {
    int i=0, save_state, save_type;  int *my_id = idp;
    printf("watch_count(): thread %d\n", *my_id);
    pthread_mutex_lock(&count_lock);
    while (count < COUNT_THRES) {
        pthread_cond_wait(&count_hit_threshold, &count_lock);
        printf("watch_count(): thread %d, count %d\n", *my_id, count);
    }
    pthread_mutex_lock(&done_lock);
    we_are_done++;
    printf("watch_count(): shut down in progress\n");
    pthread_mutex_unlock(&done_lock);
    pthread_mutex_unlock(&count_lock);
    return(NULL);}
```
Output with shutdown [1]

inc_count(): thread 0, count = 1, pass= 0 unlocking mutex
inc_count(): thread 1, count = 2, pass= 0 unlocking mutex
inc_count(): thread 2, count = 3, pass= 0 unlocking mutex
inc_count(): thread 3, count = 4, pass= 0 unlocking mutex
inc_count(): thread 4, count = 5, pass= 0 unlocking mutex
watch_count(): thread 5
inc_count(): thread 4, count = 6, pass= 1 unlocking mutex
inc_count(): thread 3, count = 7, pass= 1 unlocking mutex
inc_count(): thread 2, count = 8, pass= 1 unlocking mutex
inc_count(): thread 1, count = 9, pass= 1 unlocking mutex
inc_count(): thread 4, count = 10, pass= 2 unlocking mutex
inc_count(): thread 0, count = 11, pass= 1 unlocking mutex
inc_count(): thread 3, count = 12, pass= 2 unlocking mutex
inc_count(): thread 4, count = 13, pass= 3 unlocking mutex
inc_count(): thread 2, count = 14, pass= 2 unlocking mutex
inc_count(): thread 3, count = 15, pass= 3 unlocking mutex
Output with shutdown [2]

inc_count(): thread 4, count = 16, pass= 4 unlocking mutex
inc_count(): thread 1, count = 17, pass= 2 unlocking mutex
inc_count(): thread 0, count = 18, pass= 2 unlocking mutex
inc_count(): Thread 0, count 18
watch_count(): thread 5, count 18
watch_count(): shut down in progress
inc_count(): thread 2  we are done shutting down (0:3)
inc_count(): thread 4  we are done shutting down (0:5)
inc_count(): thread 3  we are done shutting down (0:4)
inc_count(): thread 1  we are done shutting down (0:3)
inc_count(): thread 0  we are done shutting down (0:3)
all threads done
Output without shutdown [1]

- inc_count(): thread 0, count = 1, pass= 0 unlocking mutex
- inc_count(): thread 1, count = 2, pass= 0 unlocking mutex
- inc_count(): thread 2, count = 3, pass= 0 unlocking mutex
- inc_count(): thread 3, count = 4, pass= 0 unlocking mutex
- inc_count(): thread 4, count = 5, pass= 0 unlocking mutex
- watch_count(): thread 5
- inc_count(): thread 4, count = 6, pass= 1 unlocking mutex
- inc_count(): thread 3, count = 7, pass= 1 unlocking mutex
- inc_count(): thread 2, count = 8, pass= 1 unlocking mutex
- inc_count(): thread 1, count = 9, pass= 1 unlocking mutex
- inc_count(): thread 4, count = 10, pass= 2 unlocking mutex
- inc_count(): thread 0, count = 11, pass= 1 unlocking mutex
- inc_count(): thread 3, count = 12, pass= 2 unlocking mutex
- inc_count(): thread 4, count = 13, pass= 3 unlocking mutex
- inc_count(): thread 2, count = 14, pass= 2 unlocking mutex
- inc_count(): thread 3, count = 15, pass= 3 unlocking mutex
Output without shutdown [2]

- inc_count(): thread 4, count = 16, pass= 4 unlocking mutex
- inc_count(): thread 1, count = 17, pass= 2 unlocking mutex
- inc_count(): thread 0, count = 18, pass= 2 unlocking mutex
- inc_count(): Thread 0, count 18
- watch_count(): thread 5, count 18
- inc_count(): thread 2, count = 19, pass= 3 unlocking mutex
- inc_count(): thread 4, count = 20, pass= 5 unlocking mutex
- inc_count(): thread 3, count = 21, pass= 4 unlocking mutex
- inc_count(): thread 4, count = 22, pass= 6 unlocking mutex
- ...
- inc_count(): thread 1, count = 85, pass=16 unlocking mutex
- inc_count(): thread 0, count = 86, pass=14 unlocking mutex
- inc_count(): thread 1, count = 87, pass=17 unlocking mutex
- inc_count(): thread 0, count = 88, pass=15 unlocking mutex
- inc_count(): thread 0, count = 89, pass=16 unlocking mutex
- inc_count(): thread 0, count = 90, pass=17 unlocking mutex
- all threads done