Discussion 7: parallelism

Sen Zhou
POSIX threads (pthread)

- **POSIX**
  - "**Portable Operating System Interface [for Unix]**"
- **Basic pthread functions**
  - `#include <pthread.h>`
  - Type of handle of a thread
    - `pthread_t`
  - Creating a thread
    - `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);`
  - Suspending the calling thread until the target thread terminates
    - `int pthread_join(pthread_t thread, void **value_ptr);`
Mutexes

• Mutexes have two basic operations
  – Lock
  – Unlock
• If a mutex is unlocked and a thread calls lock, the mutex locks and the thread continues.
• If the mutex is locked, the thread blocks until the thread 'holding' the lock calls unlock.
Mutex functions

• Type of handle of mutex
  – pthread_mutex_t

• Create a mutex
  – int pthread_mutex_init (pthread_mutex_t *mut, const pthread_mutexattr_t *attr);

• Lock a mutex
  – int pthread_mutex_lock (pthread_mutex_t *mut);

• Unlock a mutex
  – int pthread_mutex_unlock (pthread_mutex_t *mut);

• Deallocates any memory or resource associated with the mutex.
  – int pthread_mutex_destroy (pthread_mutex_t *mut);
Examples of pthread and mutexes

• To use pthread, we need to add –lpthread when compiling using gcc

• Examples
  – hello.c
    • Using pthread
  – count.c
    • Using a mutex to ensure the atomicity(all or none) of adding operation
Compiler optimizations

• Compiler optimizations will keep the result of a single-thread program

• If there are multiple thread inside a program, compiler optimizations are not always correct:
  – Do not use optimization (not a good choice for bigger size programs)
  – Use mutexes to help regulate the behavior of parallel threads
Examples of compiler optimization

• Undesired compiler optimization
  – spin.c
    • while(flag==0) is considered as an endless loop and is optimized
  – badorder.c
    • The order of data=100; and flag=1; is reversed by the compiler optimization
    • Happens on ladera but not in newport
    • Examples where bad things happen are not necessarily robust to changes in compilers etc
      ----Brian Demsky

• Fixed compiler optimization
  – fixspin.c
Deadlocks

- A set of blocked threads, each holding a resource and waiting to acquire a resource held by another thread in the set
- Eg. Two threads: P0, P1
  - When P0 holds A and P1 holds B, deadlock

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock(A)</td>
<td>Lock(B)</td>
</tr>
<tr>
<td>Lock(B)</td>
<td>Lock(A)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Unlock(B)</td>
<td>Unlock(A)</td>
</tr>
<tr>
<td>Unlock(A)</td>
<td>Unlock(B)</td>
</tr>
</tbody>
</table>
Deadlocks

• 4 necessary conditions for deadlock to happen:
  – Mutual exclusion
    • One can access at a time
  – Hold and wait
  – No preemption
    • Otherwise one thread can ask the other one to unlock its locks
  – Circular wait
Example of deadlocks

• bank.c
  – Try to avoid deadlocks through the fourth way
  – Always have smaller number account wait for greater number account, thus no circular wait...