Thread Interaction

COMP450 Operating Systems
Thread Interaction

• Mutual Exclusion
  – Creates critical sections.
  – Limits access to shared data to a single thread

• Synchronization
  – Allows one thread to wait until another thread completes some action.
  – May be combined with mutual exclusion.
Synchronization

• In multithreaded applications, one thread may need to wait until another thread completes some activity.

• To synchronize, a thread may need to wait until another thread:
  – terminates
  – performs some activity
Notification of Termination

- A thread can wait until another thread terminates.
- In Java and C# this is done using the join method of the Thread class:
  ```java
  public void join();
  public void join(int maxMillisec);
  ```
Java Thread Termination

// start a new thread
SubThread xyz = new SubThread();
xyz.start();

// Wait for thread to terminate
xyz.join();
pthread join

```c
int pthread_join(
    pthread_t threadObj,
    void **exit_value);
```

where:

- `threadObj` is the thread object used when the thread was created.
- `exit_value` is the value the terminating thread passed to `pthread_exit`.

pthread join

pthread_t threadObj;
int result;
pthread_create(  // create thread
    &threadObj, NULL, myfunc, NULL);

// wait for thread to terminate
pthread_join(threadObj, &result);
Java Wait and Notify

• In a synchronized method the wait method will cause a thread to be suspended until started again by the notify or notifyAll methods.

    object.wait();
    object.wait(int millisec);
    object.notify();
    object.notifyAll();
Many multiple thread programs can be viewed as instances of the producer / consumer problem. Many producer threads create objects and put them on a shared queue. Many consumer threads take objects from the queue and use them. Consumers wait if there are no objects on the queue.
public class prodcon {
    private int numOnQ = 0;

    synchronized thing getFromQ() {
        if (numOnQ == 0) wait();
        numOnQ--;
        return next on Q;
    }

    synchronized putOnQ(thing x) {
        numOnQ++;
        put x on Q;
        notify();
    }
}
Consumer/Producer Semaphores

semaphore numInQ = 0;  mutex = 1;
void putOnQ( object x) {
    p(mutex);  // lock queue access
    put object on queue;
    v(mutex)   // unlock queue
    v(numInQ); // increment queue size
}

object getFromQ() {
    p(numInQ);   // dec q, wait if q is empty
    p(mutex);    // lock queue access
    put object on queue;
    v(mutex)     // unlock queue
}
pthreads in C on Unix

• The pthread functions provide mutual exclusion in a manner similar but slightly different from semaphores.

• The pthread mutex object can be locked and unlocked.

• Locking an already locked mutex object causes the thread to block.
pthread mutex methods

```c
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_init(&mutex);
pthread_mutex_lock(&mutex);
pthread_mutex_unlock(&mutex);
```
.NET Mutual Exclusion

• You can create a Mutex object by:

HANDLE CreateMutex(
    LPSECURITY_ATTRIBUTES secure,   // SD
    BOOL owner,                    // initial owner
    LPCTSTR name                   // object name
);

CreateMutex parameters

- security descriptor or null
- If owner is true then the creator initially owns the Mutex object. If false, the object is not initially owned by any thread.
- If the name is not null, it represents a name for the Mutex object.
Single Wait function

DWORD WaitForSingleObject(
    HANDLE object,       // handle to object
    DWORD dwMilliseconds // time-out interval
);

• object can be a Mutex, semaphore, process or thread
Multiple Object Wait

DWORD WaitForMultipleObjects(
    DWORD nCount,          // num handles in array
    CONST HANDLE *lpHandles, // handle array
    BOOL bWaitAll,          // wait option
    DWORD dwMilliseconds    // time-out interval
);

• if bWaitAll is true then wait for all
• if false then wait for any