Synchronization V
Dmitri Loguinov
Texas A&M University
February 21, 2024
Chapter 5: Roadmap

5.1 Concurrency
5.2 Hardware mutex
5.3 Semaphores
5.4 Monitors
5.5 Messages
5.6 Reader-Writer
**Mutex**

- Windows kernel mutex has semantics close to a binary semaphore 2.0, with two exceptions:
  - Repeated mutex lock from the same thread does not block it
  - Mutex can only be unlocked by the thread that locked it
- Examples:

```c
Semaphore semaX = {1, 1}; // (s,max)
Thread () {
    semaX.Wait(); // P
    semaX.Wait(); // P
}
```

```c
Mutex m; // unlocked
Thread () {
    m.Lock();
    m.Lock();
}
```

deadlocks because it attempts to decrement s twice
works fine as this thread already owns the mutex
Mutex

Examples (cont’d):

Semaphore semaX = {1, 1}; // (s,max)
Thread1 () {
    semaX.Wait(); // P
    semaX.Wait(); // P
}

Semaphore semaX = {1, 1}; // (s,max)
Thread2 () {
    // some initialization
    semaX.Release(); // V
}

Mutex m;
Thread1 () {
    m.Unlock(); // does nothing
}

Mutex m; // initially unlocked
Thread2 () { // thread2 runs first
    m.Lock();
    // long critical section
}

thread₁ blocks temporarily, then gets unblocked by thread₂

thread₁ deadlocks if thread₂ runs first; how to fix this?

thread₁ fails to unlock mutex owned by thread₂
The last standard synchronization primitive is an **event**
- An event can be in two states: signaled (1) and non-signaled (0) just like a binary semaphore
- However, it also has two possible modes of operation
  - AUTO = binary semaphore
  - MANUAL = event stays signaled until manually reset

```cpp
class Event {
    int s; // state
    int mode;
    List blocked;
    Wait(); Set(); Reset();
}
```

```cpp
Event::Wait() {
    if (s == NOT_SIGNALED)
        // block current thread
    else if (mode == AUTO)
        s = NOT_SIGNALED;
}
```

```cpp
Event::Set() {
    if (blocked.size() > 0)
        if (mode == AUTO)
            // unblock 1 thread
        else
            // unblock all threads
            s = SIGNALED;
    else
        s = SIGNALED;
}
```

```cpp
Event::Reset() {
    s = NOT_SIGNALED;
}
```
Windows APIs

- **Semaphore**
  - Security is NULL as always
  - Name can be used when multiple processes need to open the same object

- **Wait (i.e., P)**
  - WaitForSingleObject()
    - Returns WAIT_OBJECT_0 when ready
    - WAIT_TIMEOUT if timeout
    - Otherwise, an error

- **Release (i.e., V)**
  - ReleaseSemaphore(N)

- **CreateMutex/CreateEvent**
  - Can specify if this thread initially owns the mutex and initial state for event

- **Locking done with WaitForSingleObject()**
  - Unlocking with ReleaseMutex() and signaling with SetEvent()

- **Resetting events**
  - ResetEvent()