CS 5204
Operating Systems
Lecture 4

Godmar Back

Virginia Tech

Announcements

- Reminder: Paper Evaluation due Wed & Fri before class. Options include:
  - Hardcopy
  - Via email to xwensi@vt.edu (in PDF format, 1 pg)
- Send me your paper preferences if you haven't already

Recap: Reasons for Multithreading

- Overlap I/O and computation
  - Hide latency
- Reduce latency
  - If thread system supports preemption
- Exploit multiprocessors
  - CPU concurrency
- Software engineering reasons
  - Separation of concerns
- Non-reasons:
  - Performance as in reduction of execution time

Expressing Critical Sections

```
pthread_mutex_t m;
…
pthread_mutex_lock(&m);
/* in critical section */

if (*) {
   pthread_mutex_unlock(&m);
   return;
}

pthread_mutex_unlock(&m);
```

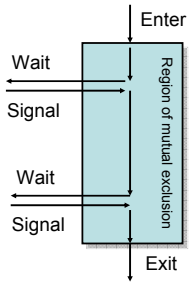
```
synchronized (object) {
   /* in critical section */

   if (*) {

      return;
   }

}
```

Pthreads/C vs Java

Note benefits of language support

Monitors (Hoare)

- Data Type:
  - internal, private data
  - public methods wrapped by Enter/Exit
  - wait/signal methods
- "Monitor Invariant"



Expressing Monitors

```
pthread_mutex_t m;
pthread_cond_t c;
…
 pthread_mutex_lock(&m);
 /* in critical section */

 while (somecond != true)
    pthread_cond_wait(&c, &m);

pthread_mutex_unlock(&m);
```

```
synchronized (object) {
   /* in critical section */

   while (somecond != true) {
      object.wait();
   }
}
```

```
pthread_mutex_lock(&m);
/* in critical section */
pthread_cond_signal(&c, &m);
pthread_mutex_unlock(&m);
```

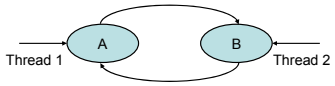
```
synchronized (object) {
   /* in critical section */
   object.notify();
}
```

See also *Java's insecure parallelism* [Per Brinch Hansen 1999]

Deadlock

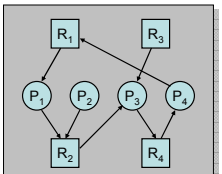
```
pthread_mutex_t A;
pthread_mutex_t B;
…
pthread_mutex_lock(&A);
pthread_mutex_lock(&B);
…
pthread_mutex_unlock(&B);
pthread_mutex_unlock(&A);
```

```
pthread_mutex_lock(&B);
pthread_mutex_lock(&A);
…
pthread_mutex_unlock(&A);
pthread_mutex_unlock(&B);
```



Deadlocks, more formally

- 4 necessary conditions
  - Mutual Exclusion
  - Hold and Wait
  - No Preemption
  - Circular Wait
- Q.: what are strategies to detect/break/avoid deadlocks?



Resource Allocation Graph
R → P Assignment
P → R Request

Implementing Threads

- Issues:
  - Who maintains thread state/stack space?
  - How are threads mapped onto CPUs?
  - How is coordination/synchronization implemented?
  - How do threads interact with I/O?
  - How do threads interact with existing APIs such as signals?
  - How do threads interact with language runtimes (e.g., GCs)?