

Class Notes Sept 8 2006.

Readers and Writers.

A more complicated problem is readers and writers. Any number of readers can read simultaneously, but only one writer can write at a time. Furthermore, to preserve consistency, writers must exclude readers.

Here's how it works

Writer:

```
Wrt.Wait( );
```

```
    Write
```

```
Wrt.Signal( );
```

Reader:

```
Rdr.Wait( );
```

```
    ReadCount++;
```

```
    if (ReadCount == 1)
```

```
    {
```

```
        Wrt.Wait( );
```

```
    }
```

```
Rdr.Signal( );
```

```
Read;
```

```
Rdr.Wait( );
```

```
    ReadCount--;
```

```
    if (ReadCount == 0)
```

```
    {
```

```
        Wrt.Signal( );
```

```
    }
```

```
Rdr.Signal( );
```

Monitors

A monitor is a class. The syntax in the book is based on Pascal, and does not use constructors properly. If monitors actually were implemented in C, the initialization section would be replaced by a constructor. At any given time, only one copy of a monitor exists in the system. The monitor's data is shared by all processes. Only one process can be executing a monitor function at a time. There is mutual exclusion between invocations of the same function, and between invocations of different functions of the same monitor. Wait and signal work differently from semaphores. Wait and signal must

be executed inside the monitor. Signal does nothing if there is nothing waiting. Wait breaks mutual exclusion and suspends the current process. Signal resumes a waiting process once the current function exits. (Or immediately?)

Threads

A thread is a cheap process – Usually.