CSCI 344-715
Lecturer: Dr. Simina Fluture

Lecture #8

Topics: Signal and Continue Monitors
Monitors vs. Semaphores

Readings: **Lecture on the web**
Class notes
[SH] 5.8

Implementing Monitors using Semaphores

Implementing mutual exclusion of monitor procedures
Each service method that is inside the monitor can only be executed by one thread at a time. By this the monitor implicitly enforces the mutual exclusion condition.

Implementing condition synchronization between monitor procedures.
(code that implements wait, signal and other operations on condition variables)

When a process does a **wait** on a condition, it will block on the queue of that condition.
For each condition we will use a semaphore, Semi.
We can consider **only one condition**.

T0 – waiting thread T1 – notifying thread

Operations on Condition Variables
Signal and Continue monitor

Wait(cond):

Signal(cond):

Signal and Exit

The notified thread has to be the next that will execute inside the monitor.
After the signal, the signaling thread immediately leaves the monitor.

Wait(cond):

Signal(cond):

Implementing Semaphores using Monitors

Binary Semaphores:
P(S): if (S == 1) { S = 0 };
Else add the thread to the semaphore queue; // Block

V(S): if (s.queue !empty) remove a process from the queue;
Else S = 1;

Using monitors (pseudocode):
P( ) and V( ) service methods. Sem will be the variable condition.

P( ) **V( )**

Counting Semaphores

P(S): S--;
if (S < 0) add the thread to semaphore queue;

V(S): S++;
if (S. queue not empty) release one thread from the queue;

Using monitors (pseudocode):

P( ) { **V( ) {**

} **}**

By modifying the value of the semaphore whenever an operation is done, we try to maintain the meaning of the semaphore value.
On [SH] page 160 you can see a simulation of operation on semaphores using a Signal and Continue (Java) monitor.