CS344/715
Lecturer: Simina Fluture

Lecture #3

Topics: Semaphores (review)
Monitors
- concept
- Signal and Exit monitor
- Signal and Continue monitor

Reading: Lecture on the web **Bibliography: [SH] ch4, ch5**
Class notes
Previous material related to semaphores
[SH] ch5 (5.1, 5.2, 5.3)

Semaphores

Semaphores provide a primitive yet powerful tool for enforcing mutual exclusion and for coordinating processes. Semaphores use the operating system support.

Apart from initialization, two atomic operations can be performed on semaphores:
wait and ***signal***.

Initial Definition of *wait* and *signal*

wait(S): while S ≤ 0 do no-op;
S = S-1;

signal(S): S = S+1;

This kind of semaphore is also called *spinlock.*

Overcoming the need for busy waiting:

Rather then busy waiting, the process can ***block*** itself.

Classical definition of *wait* and *signal*

We define a semaphore as a record:

```
type semaphore = record
                    value: integer;
                    queue: list of process;
                end;
```

Each semaphore has an integer value and a list of processes (queue).

wait(S): S .value= S.value-1;

```
if S.value < 0
then begin
        add this process to S.queue;
        block the process;
end;
```

signal(S):

```
S.value := S.value +1;
if S.value ≤ 0;
then begin
        remove a process P from  S.queue;
        wakeup(P);                          /* release */
end;
```

Mutual Exclusion implementation: *n* processes share a semaphore, named *mutex.*
Initially mutex:=1.

repeat
wait(mutex);
CS
signal(mutex);
remainder section
until false;

Producer/Consumer synchronization

Producer	Consumer
Wait(Empty)	Wait(Full)
……	…….
……	…….
Signal(Full)	**Signal(Empty)**

Monitors
Monitors provide a higher level construct than semaphores. Monitors and Semaphores are equal in power in that they solve the same types of problems.
Monitors are declared and implemented in different ways in different languages.

Fig. Monitor Service Methods, Condition Variables, and Sets

Each monitor object has a lock. Only one thread is allowed to be active (execute) inside the monitor at a given time. By this the Mutual Exclusion condition is implicitly enforced.
The data global to all service methods exist and retain their values as long as the monitor exists.

Only the service methods names are visible outside the monitor. Calls of monitor methods have the form:
call **mname.methodname(arguments)**

Two operations can be done on condition variables:
Wait (wait())
Signal (notify(), notifyAll())

Signaling Disciplines
When a process executes **signal**, it is executing inside the monitor and has control over the lock. If **signal** resumes another thread, then there are two threads that could execute: the one that executes signal (**P**) and the one that just was awakened (**Q**)

Signal and Wait: P waits until Q leaves the monitor or waits for another condition.

Signal and Exit:

Signal and Continue:

Fig. State Diagram for synchronization in monitors.