CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Monitor Implementation

To implement a monitor, we need to

- ensure only one process at a time has access
- implement condition variables

We know how to protect a critical region using semaphores. Implementing condition variables with semaphores is not too cumbersome due to the similarity between the two constructs

Our rules:

- only one process may be executing at once within the monitor
- a process that `wait`s on a condition variable immediately gives up control
- processes already in the monitor (`wait`ing) have scheduling priority over those that are not

Two semaphores:

- `mutex`: control access to monitor itself
- `next`: control which process already *inside* monitor can run

Additional variable:

- `next_count`: integer indicating how many processes are waiting inside monitor

One additional semaphore and counter per condition variable

Without condition variables, we already know how to control access

Replace each externally callable function with

```
wait(mutex);

...
body of function
...

signal(mutex);
```

With condition variables, we decided to allow waiting processes to run before processes not yet in the monitor

```
wait(mutex);

...
body of function
...

if (next count > 0)
    signal(next);
else
    signal(mutex);
```

Condition variable x in monitor associated with two variables in our implementation:

- `x_count` -- number of processes waiting on x (integer)
- `x_sem` -- used to implement waiting and signaling

Both initialized to 0

Implementing `x.wait()`

```
x_count++;
if (next_count > 0)
    signal(next);
else
    signal(mutex);
wait(x_sem);
x_count--;
```

```
x_count++;              // let others know we are
                        // interested

if (next_count > 0)     // these lines just give
    signal(next);       // control to the next
else                    // process (inside or
    signal(mutex);      // outside the monitor)

wait(x_sem);            // after giving up control,
                        // we need to wait

x_count--;              // control returned --
                        // no longer waiting on x
```

Implementing `x.signal()`

```
if (x_count > 0) {
    next_count++;
    signal(x_sem);
    wait(next);
    next_count--;
}
```

```
// signal in condition variables acts only if something is
// already waiting
if (x_count > 0) {

    // let others know we will be waiting inside monitor
    next_count++;

    // allow one waiting process to go
    signal(x_sem);

    // only one process can be active, so we must wait
    wait(next);

    // control returned -- no longer waiting
    next_count--;
}
```

With mutual exclusion and condition variables, we have everything we need for a monitor