Monitors

• A monitor type resembles a class in a language like C++ or Java
  – Contains declarations of shared data
  – Its procedures encode operations that manipulate shared data and perform process synchronization through condition variables; compiler ensures that at most one process can be executing a monitor operation at any time
    • A condition is a situation of interest in a monitor
    • A condition variable is associated with a condition
    • A process executing a wait operation on condition variable is blocked until some process performs a signal operation

• In a concurrent system, processes share data by creating a monitor object
  – We call it simply a monitor

Example: Monitor Implementation of a Binary Semaphore

![Diagram of Binary Semaphore]

Figure 6.32 Monitor implementation of a binary semaphore.

Figure 6.33 A monitor implementing a binary semaphore.

Figure 6.34 A snapshot of the system of Example 6.5.
Example: Producers-Consumers Using Monitors

Type Bounded_buffer_type = monitor
const
n = \ldots;
type
item = \ldots;
var
buffer : array [0..n-1] of item;
full, prod_ptr, cons_ptr : integer;
buff_full : condition;
buff_empty : condition;
begin [initialization]
full := 0;
prod_ptr := 0;
cons_ptr := 0;
end

procedure produce (produced_info : item)
begin
if full = n, then buff_empty.wait;
buffer[prod_ptr] := produced_info;
prod_ptr := prod_ptr + 1 mod n;
full := full + 1;
buff_full.signal;
end;

procedure consume (for_consumption : item)
begin
if full = 0, then buff_full.wait;
for_consumption := buffer[cons_ptr];
cons_ptr := cons_ptr + 1 mod n;
full := full - 1;
buff_empty.signal;
end;

Figure 6.35 Producers–consumers using monitors.

Figure 6.36 Snapshots of the monitor of Example 6.7 with a single buffer.
Monitors in Java

• A Java class becomes a monitor type when the attribute *synchronized* is associated with one or more methods in the class
  – An object of such a class is a monitor
• The java virtual machine ensures mutual exclusion over the synchronized methods in a monitor
• Each monitor contains a single unnamed condition variable
  – Can lead to busy waits in an application that has many conditions

Summary

• Process synchronization is a generic term for data access synchronization and control synchronization
• A race condition occurs when actions of concurrent processes may have unexpected consequences
  – Avoided through *mutual exclusion*
• Avoidance of race conditions is a primary issue in process synchronization
• Critical section: section of code that accesses some shared data in a mutually exclusive manner
• Synchronization achieved through: *indivisible instructions, semaphores, monitors*