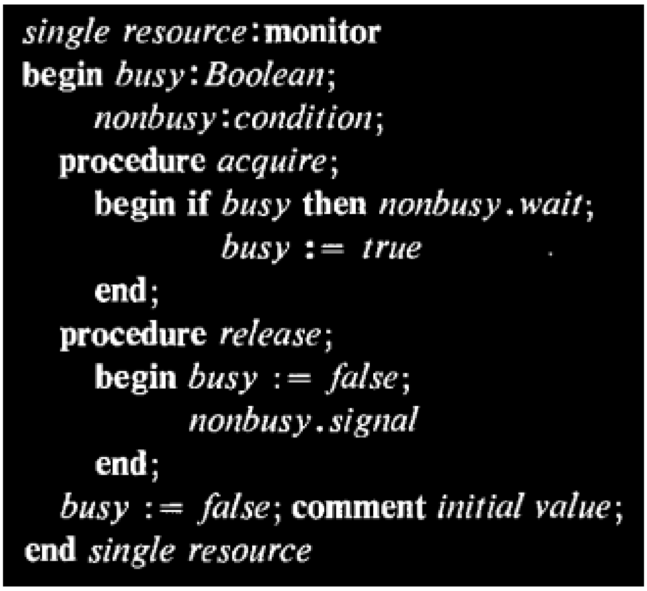
CSE221 Lecture 5

Aronya Baksy

October 2024

1 Monitors: An OS Structuring Concept

- Hoare defines a monitor as a **collection** of **programs, subroutines and administrative data** used to co-ordinate resource alloc/de-alloc (for a particular type of resource)
- Calls to monitor routines have to be serialized (only one process in the monitor at a time)
- Monitors scheduling similar resources may be grouped into **classes**
- This needs a **wait** operation (to prevent concurrent access) and a **signal** operation (for one waiting process to get ready to access the monitor)
- The construct introduced to solve this problem is called the **condition variable**. A monitor routine declares one condvar for each reason for waiting
- A condvar has 2 operations:
  - **wait**: suspend the thread and release the monitor lock
  - **signal**: wake up one waiting thread and acquire the monitor lock
- Hoare's vision is of very coarse-grained monitors (one per subsystem)



```
single resource:monitor
begin busy:Boolean;
      nonbusy:condition;
   procedure acquire;
      begin if busy then nonbusy.wait;
               busy := true
      end;
   procedure release;
      begin busy := false;
               nonbusy.signal
      end;
   busy := false; comment initial value;
end single resource
```

Figure 1: Design of simple monitor with `acquire()` and `release()` methods using a condition variable `nonbusy`

1.1 Condition Variables

- Implementation: queue of processes waiting, initially empty
- The code listing in 1 shows that condvars are identical to Dijkstra semaphores
- Semaphores can also be used to implement condition variables:

  - One semaphore per monitor to ensure mutual exclusion among the monitor routines, called `mutex`
  - One semaphore per monitor to maintain number of waiters called `urgent`
  - for each condition local to the monitor, a semaphore condsem initialized to 0 for process to suspend itself when calling wait, called `condsem`

- Possible optimizations:
  - Implement conditions in hardware
  - Abolish the integer variables `condcount` and `urgentcount` and instead inspect the semaphore (this is an atomic op)
  - Short monitors that don't call other monitors can just execute atomically (using h/w support) instead of using synchronization primitives

- The monitor invariant I is a condition on a monitor procedure's data that must be true whenever no thread is executing in the monitor i.e. the condition must be true before and after every proc call

- The monitor invariant is important to ensure the consistency of monitor data in a multiprocess environment

- The assertion B describes the condition under which a program waiting on a condition variable wishes to be resumed

- **Hoare semantics** can be explained as (which of I and B are true before/after wait/signal)

$$I\{b.wait\}I\&B \tag{1}$$

$$I\&B\{b.signal\}I \tag{2}$$

2 Process and Monitors in Mesa

2.1 Goals

- Resolve problems with practical impl. of monitors in OS
  - Definition of a wait operation
  - Priority scheduling
  - Timeout, interrupt and exception handling
  - Interactions with process creation and destruction
  - monitoring large numbers of small objects

- Mesa monitors deal with practical challenges not explored by Hoare's paper

- Designed for application programs heavy on concurrency

- Following facilites provided:
  - **Local concurrent programming**: every app. is represented as a number of concurrent processes
  - **Global resource sharing** at the intra-app and inter-app level
  - **Replacing interrupts** by waking up appropriate processes instead of a forced branch

- Monitors for synchronization over message passing as authors found it easier than designing a message passing funcitonality integrated with the Mesa lang.

- Monitors for sync. over pre-emptive scheduling as it allows multiprogramming, avoids multiple scheduling schemes (stuff like I/O interrupts are pre-emptive anyway), makes modularity possible and works well with virtual memory schemes

Hoare	**Mesa**
the signaler yields the monitor to the released thread	the signaling thread continues and the released thread yields the monitor
The signaler is suspended after it signals	The signaler continues to run after signaling
the signaler's monitor lock is taken away and given to the released thread, and it is suspended	the released thread does not get it's monitor lock back from the signaler, and must wait for the monitor to be empty
Use if to check condition before a wait	Use while to check condition as it may be false after wait

Table 1: Hoare vs Mesa Monitors

2.2 Hoare vs Mesa Monitors

2.2.1 Advantage of Mesa Monitor

- Allows very simple verification rules (The monitor invariant must be established just before a return from an entry procedure or a WAIT)
- Allows broadcast operations on multiple waiters (and each waiter then checks the specific condition they waiting on)

2.2.2 Deadlock Patterns

- Two processes both call wait and keep waiting on each other
- M, N are monitors that call entry procedures in each other and wait for each other to release the monitor lock (impose partial ordering to fix this)
- M calls N, and N then waits for a condition which can only occur when another process enters N through M

2.2.3 Naked Notify operation for hardware

- Shared memory area for passing commands to devices which can be read from/written to atomically
- Notify is used by device to wake up the listening process waiting on the condvar
- The race condition caused by lack of a monitor lock (hence the name naked) is solved by using a design pattern called the **wakeup-waiting switch**

3 Hoare vs Mesa vs Java Monitors

	Hoare	**Mesa**	**Java**
Type of cond-var	Explicit	Explicit	implicitly declared by compiler (explicit condvars recently added)
wait semantics	same	same	same
signal	yes	notify, broadcast	notify, notifyAll
granularity	coarse	fine (monitors compose modules that make up the OS)	code block or entire class, static/runtime
abort semantics	none	abort signal can be sent to a process that will resume immediately on the next wait and finish execution	Exception handlers
nesting	not handled	let first call to a lock work and let devs handle this case	same as mesa

Table 2: Hoare vs Mesa Monitors