Understanding Monitors: A Report on Hoare's Concept

Jeff Brandon

March 21, 2014

CPS 470 Operating Systems

This paper is designed to show understanding of the concepts proposed by C.A.R. Hoare in his paper *Monitors: An Operating System Structuring Concept* published 1974. This is important to solving problems of scheduling, mutual exclusion, synchronization, and multiprogramming. Proposed concepts, given examples, and the ramifications they carry will be explained. Monitors, while difficult to implement in some programming languages, provide a clean solution to many multiprogramming problems and therefore should be understood by those who study computer science.

Hoare proposed the concept of a monitor as a scheduler for a specific class of resource. Scheduling is important because an operating system must implement some form of scheduling to share the resources it manages. A monitor handles several traditional scheduling problems using condition variables which have wait and signal processes associated with them. Hoare's solutions for the bounded buffer problem and the readers and writers problem will be explained along with the priority scheme used to implement the solution to the readers and writers problem.

Monitors are schedulers for a specific class of resource used to simplify the scheduling process. A monitor holds some data pertinent to the resource it manages and procedures that

may be called by programs that wish to use the resource(s). Monitors manage the resource they are assigned to using data only accessible by the monitor. The procedures contained in a monitor can only manipulate data local to the procedure or the monitor itself. By design, only one procedure of a monitor may be executed at one time. This means that the monitor enforces mutual exclusion. Monitors handle critical sections of programs using the compiler so that there is less room for user error when writing a program that deals with a shared resource, reducing the chance of encountering a deadlock. One necessary component of a monitor is a condition variable.

Condition variables are dissimilar to Boolean variables in that no stored value is accessible to the program. Each condition variable represents a reason why a process is unable to proceed with execution and needs to wait. Each condition is also associated with wait and signal operations. Wait is used to put the executing program on hold until a signal on the condition is sent. It is important that when a wait operation is issued, the program relinquishes its mutual exclusion on the resource so that other monitor procedures may execute. If the process were to maintain mutual exclusion of the monitor, the signal it is waiting on could not be sent and the solution would not work. Signal is used on a condition to let waiting processes know that the condition has been met. Because more than one process may be waiting on a signal, when a signal is sent one process that is currently waiting is given access to the resource. The program given access next can depend on implementation and the resource being managed, one simple example is the process that has waited longest gets access next.

The condition variables' wait and signal operations are different from P and V operations that are used with semaphores. Inside a procedure of a monitor a Boolean value is checked, if the condition specified by that Boolean value is not met, the wait operation of a condition variable is run and mutual exclusion is released. When a signal for that condition is received or if the Boolean value evaluates to true, the procedure continues execution. When P is called on a semaphore, a request for access is made to the semaphore and the program halts until access is granted. In this situation mutual exclusion is not released on the semaphore until the program may continue. In essence, the wait operation works similarly to the P operation on a semaphore but the major difference is that wait does not maintain mutual exclusion of the resource like P does. Likewise signal and V perform similar tasks. Signal lets a waiting process know that it is able to continue because the condition it was waiting for has been met. V on the other hand releases resource managed by a semaphore and changes its value. The major difference between signal and V is that V will affect the semaphore regardless of when it is called whereas signal will do nothing when no processes are waiting on the condition signaled.

In his paper Hoare gives a solution for the bounded buffer problem using monitors. The bounded buffer problem involves a producer, a consumer and a buffer of size N. The producer creates items to be placed in the buffer, the consumer takes items from the buffer and processes them. It is important that the producer does not write to a full buffer and that the consumer does not read from an empty buffer. To solve the problem the producer and consumer are both procedures in a monitor that manages the bounded buffer. This ensures that the producer and consumer will have exclusive access to the buffer whenever they affect it. When the producer is ready to produce an item it first checks to see if the buffer is full, and if it is it waits on the nonfull condition. When the buffer is not full it places the item in the buffer at a position

specified by a pointer which points to the next location the producer should insert into the buffer. After the item is inserted into the buffer, pointer is incremented modulo N-1 and a signal is sent on the nonempty condition. Likewise the consumer, when it is ready to consume an item, first checks if the buffer contains at least one item. If it does not (the buffer is empty) the consumer waits on the nonempty condition. When the buffer has an item the consumer consumes the item at (pointer minus count) modulo N-1 where count is the number of items in the buffer. This specifies the item to be consumed next. Once item consumption has finished a signal is sent on the nonfull condition because the buffer is not full. The initial value of pointer is 0 the first element in the buffer. The number of items in the buffer is maintained in the count variable, it is initialized to 0 and incremented when an item is produced and decremented when an item is consumed.

Another solution Hoare gives using monitors is the readers and writers problem. The situation occurs when a file is being written to and read from by multiple readers and writers. It stands to reason that when one writer has access to the file it needs mutual exclusion. When a reader is reading a file it also makes sense that any other number of readers should be allowed to read the file because no changes are being made to it. To implement the solution the monitor has four procedures: startread, startwrite, endread, and endwrite. Condition variables OKtoread and OKtowrite are needed as well. A count of readers currently reading the file and a busy Boolean value are also necessary. The startread procedure first checks for busy being true or the OKtowrite condition having a queue, meaning there is a writer currently writing or a writer waiting to write. If the Boolean expression evaluates to true then the OKtoread condition is waited on. By not allowing a reader to begin reading when there is a writer waiting it ensures

that a writer will never be blocked by a continuous stream of readers reading a file. When OKtoread is signaled readercount is incremented by one and OKtoread is signaled again, this allows all currently waiting readers to enter. The endread procedure first decrements the readercount by 1 and then if the readercount is 0 is sends the OKtowrite signal letting any waiting writers know the coast is clear for them to modify the file. The startwrite procedure first checks to see if the readercount is nonzero or if busy is true, if so then the procedure waits on the OKtowrite condition. When OKtowrite is signaled startwrite sets busy to true. Finally the endwrite procedure sets busy to false. Then it checks to see if there are any readers waiting to read the document, if there are it sends the OKtoread signal otherwise it sends the OKtowrite signal. This implementation ensures that writers do not exclude readers from accessing the document even if it means the readers are reading stale information.