

Contents

Preface

- 1 Introduction to Concurrent Programming**
 - 1.1 Processes and Threads: An Operating System's View
 - 1.2 Advantages of Multithreading
 - 1.3 Threads in Java
 - 1.4 Threads in Win32
 - 1.5 Pthreads
 - 1.6 A C++ *Thread* Class
 - 1.6.1 C++ Class *Thread* for Win32
 - 1.6.2 C++ Class *Thread* for Pthreads
 - 1.7 Thread Communication
 - 1.7.1 Non-Deterministic Execution Behavior
 - 1.7.2 Atomic Actions
 - 1.8 Testing and Debugging Multithreaded Programs
 - 1.8.1 Problems and Issues
 - 1.8.2 Class *TDThread* for Testing and Debugging
 - 1.8.3 Tracing and Replaying Executions with Class Template *sharedVariable<>*
 - 1.9 Thread Synchronization
 - Further Reading
 - References
 - Exercises
- 2 The Critical Section Problem**
 - 2.1 Software Solutions to the Two-Thread Critical Section Problem
 - 2.1.1 Incorrect Solution 1
 - 2.1.2 Incorrect Solution 2
 - 2.1.3 Incorrect Solution 3
 - 2.1.4 Peterson's Algorithm
 - 2.1.5 Using the `volatile` modifier
 - 2.2 Ticket-Based Solutions to the n-Thread Critical Section Problem
 - 2.2.1 Ticket Algorithm
 - 2.2.2 Bakery Algorithm
 - 2.3 Hardware Solutions to the n-Thread Critical Section Problem
 - 2.3.1 A Partial Solution
 - 2.3.2 A Complete Solution
 - 2.3.3 A Note on Busy-Waiting
 - 2.4 Deadlock, Livelock, and Starvation
 - 2.4.1 Deadlock
 - 2.4.2 Livelock
 - 2.4.3 Starvation
 - 2.5 Tracing and Replay for Shared Variables
 - 2.5.1 ReadWrite-sequences
 - 2.5.2 An Alternative Definition of ReadWrite-sequences
 - 2.5.3 Tracing and Replaying ReadWrite-sequences
 - 2.5.4 Class Template *sharedVariable<>*
 - 2.5.5 Putting it all Together
 - 2.5.6 A Note on Shared Memory Consistency
 - Further Reading
 - References
 - Exercises
- 3 Semaphores and Locks**
 - 3.1 Counting Semaphores
 - 3.2 Using Semaphores
 - 3.2.1 Resource Allocation

- 3.2.2 More Semaphore Patterns
- 3.3 Binary Semaphores and Locks
- 3.4 Implementing Semaphores
 - 3.4.1 Implementing P() and V()
 - 3.4.2 The VP() Operation
- 3.5 Semaphore-Based Solutions to Concurrent Programming Problems
 - 3.5.1 Event Ordering
 - 3.5.2 Bounded Buffer
 - 3.5.3 Dining Philosophers
 - 3.5.4 Readers and Writers
 - 3.5.5 Simulating Counting Semaphores
- 3.6 Semaphores and Locks in Java
 - 3.6.1 Class *countingSemaphore*
 - 3.6.2 Class *mutexLock*
 - 3.6.3 Class *Semaphore*
 - 3.6.4 Class *ReentrantLock*
 - 3.6.5 Example: Java Bounded Buffer
- 3.7 Semaphores and Locks in Win32
 - 3.7.1 CRITICAL_SECTION
 - 3.7.2 Mutex
 - 3.7.3 Semaphore
 - 3.7.4 Events
 - 3.7.5 Other Synchronization Functions
 - 3.7.6 Example: C++/Win32 Bounded Buffer
- 3.8 Semaphores and Locks in Pthreads
 - 3.8.1 Mutex
 - 3.8.2 Semaphore
- 3.9 Another Note on Shared Memory Consistency
- 3.10 Tracing, Testing, and Replay for Semaphores and Locks
 - 3.10.1 Non-Deterministic Testing with the Lockset Algorithm
 - 3.10.2 Simple SYN-sequences for Semaphores and Locks
 - 3.10.3 Tracing and Replaying Simple PV-sequences and LockUnlock-sequences
 - 3.10.4 Deadlock Detection
 - 3.10.5 Reachability Testing for Semaphores and Locks
 - 3.10.6 Putting it all Together
- Further Reading
- References
- Exercises

4 Monitors

- 4.1 Definition of Monitors
 - 4.1.1 Mutual Exclusion
 - 4.1.2 Condition Variables and SC Signaling
- 4.2 Monitor-Based Solutions to Concurrent Programming Problems
 - 4.2.1 Simulating Counting Semaphores
 - 4.2.2 Simulating Binary Semaphores
 - 4.2.3 Dining Philosophers
 - 4.2.4 Readers and Writers
- 4.3 Monitors in Java
 - 4.3.1 A Better *countingSemaphore*
 - 4.3.2 *notify* vs. *notifyAll*
 - 4.3.3 Simulating Multiple Condition Variables
- 4.4 Monitors in Pthreads
 - 4.4.1 Pthreads Condition Variables
 - 4.4.2 Condition Variables in J2SE 5.0
- 4.5 Signaling Disciplines
 - 4.5.1 Signal-and-Urgent-Wait (SU)
 - 4.5.2 Signal-and-Exit (SE)

- 4.5.3 Urgent-Signal-and-Continue (USC)
- 4.5.4 Comparing SU and SC signals
- 4.6 Using Semaphores to Implement Monitors
 - 4.6.1 SC Signaling
 - 4.6.2 SU Signaling
- 4.7 A Monitor Toolbox for Java
 - 4.7.1 A Toolbox for SC Signaling in Java
 - 4.7.2 A Toolbox for SU Signaling in Java
- 4.8 A Monitor Toolbox for Win32/C++/Pthreads
 - 4.8.1 A Toolbox for SC Signaling in C++/Win32/Pthreads
 - 4.8.2 A Toolbox for SU Signaling in C++/Win32/Pthreads
- 4.9 Nested Monitor Calls
- 4.10 Tracing and Replay for Monitors
 - 4.10.1 Simple M-sequences
 - 4.10.2 Tracing and Replaying Simple M-sequences
 - 4.10.3 Other Approaches to Program Replay
- 4.11 Testing Monitor-Based Programs
 - 4.11.1 M-sequences
 - 4.11.2 Determining the Feasibility of an M-sequence
 - 4.11.3 Determining the Feasibility of a Communication-sequence
 - 4.11.4 Reachability Testing for Monitors
 - 4.11.5 Putting it all Together

Further Reading

References

Exercises

5 **Message-Passing**

- 5.1 Channel Objects
 - 5.1.1 Channel Objects in Java
 - 5.1.2 Channel Objects in C++/Win32
- 5.2 Rendezvous
- 5.3 Selective Wait
- 5.4 Message-Based Solutions to Concurrent Programming Problems
 - 5.4.1 Readers and Writers
 - 5.4.2 Resource Allocation
 - 5.4.3 Simulating Counting Semaphores
- 5.5 Tracing, Testing, and Replay for Message Passing Programs
 - 5.5.1 SR-sequences
 - 5.5.2 Simple SR-sequences
 - 5.5.3 Determining the Feasibility of an SR-sequence
 - 5.5.4 Deterministic Testing
 - 5.5.5 Reachability Testing for Message-Passing Programs
 - 5.5.6 Putting it all Together

Further Reading

References

Exercises

6 **Message-Passing in Distributed Programs**

- 6.1 TCP Sockets
 - 6.1.1 Channel Reliability
 - 6.1.2 TCP Sockets in Java
- 6.2 Java TCP Channel Classes
 - 6.2.1 Classes *TCPSender* and *TCPMailbox*
 - 6.2.2 Classes *TCPsynchronousSender* and *TCPsynchronousMailbox*
 - 6.2.3 Class *TCPSelectableSynchronousMailbox*
- 6.3 Timestamps and Event Ordering
 - 6.3.1 Event Ordering Problems
 - 6.3.2 Local Real-Time Clocks

- 6.3.3 Global Real-Time Clocks
 - 6.3.4 Causality
 - 6.3.5 Integer Timestamps
 - 6.3.6 Vector Timestamps
 - 6.3.7 Timestamps for Programs using Messages and Shared Variables
 - 6.4 Message-Based Solutions to Distributed Programming Problems
 - 6.4.1 Distributed Mutual Exclusion
 - 6.4.2 Distributed Readers and Writers
 - 6.4.3 Alternating Bit Protocol
 - 6.5 Testing and Debugging Distributed Programs
 - 6.5.1 Object-Based Sequences
 - 6.5.2 Simple Sequences
 - 6.5.3 Tracing, Testing, and Replaying CARC-sequences and CSC-sequences
 - 6.5.4 Putting it all Together
 - 6.5.5 Other Approaches to Replaying Distributed Programs
- Further Reading
References
Exercises

- 7 Testing and Debugging Concurrent Programs**
- 7.1 Synchronization Sequences of Concurrent Programs
 - 7.1.1 Complete Events vs. Simple Events
 - 7.1.2 Total Ordering vs. Partial Ordering
 - 7.2 Paths of Concurrent Programs
 - 7.2.1 Defining a Path
 - 7.2.2 Path-based Testing and Coverage Criteria
 - 7.3 Definitions of Correctness and Faults for Concurrent Programs
 - 7.3.1 Defining Correctness for Concurrent Programs
 - 7.3.2 Failures and Faults in Concurrent Programs
 - 7.3.3 Deadlock, Livelock, and Starvation
 - 7.4 Approaches to Testing Concurrent Programs
 - 7.4.1 Non-Deterministic Testing
 - 7.4.2 Deterministic Testing
 - 7.4.3 Combinations of Deterministic and Non-Deterministic Testing
 - 7.5 Reachability Testing
 - 7.5.1 The Reachability Testing Process
 - 7.5.2 SYN-sequences for Reachability Testing
 - 7.5.3 Race Analysis of SYN-sequences
 - 7.5.4 Timestamp Assignment
 - 7.5.5 Computing Race Variants
 - 7.5.6 A Reachability Testing Algorithm
 - 7.5.7 Research Directions
- Further Reading
References
Exercises