

# Monitors in Mesa

Josh Triplett

January 17, 2006

# Pilot Operating System

- Developed at Xerox PARC
- First GUI
- Written in Mesa language (Algol-like)
- Programs written in Mesa as well

# Pilot Operating System

- Developed at Xerox PARC
- **First GUI**
- Written in Mesa language (Algol-like)
- Programs written in Mesa as well

# Pilot Operating System

- Developed at Xerox PARC
- First GUI
- Written in Mesa language (Algol-like)
- Programs written in Mesa as well

# Pilot Operating System

- Developed at Xerox PARC
- First GUI
- Written in Mesa language (Algol-like)
- Programs written in Mesa as well

# Concurrency

- Take advantage of multiple processors
  - Hardware devices
  - Multitasking
  - Concurrent processing in one application

# Concurrency

- Take advantage of multiple processors
- **Hardware devices**
- Multitasking
- Concurrent processing in one application

# Concurrency

- Take advantage of multiple processors
- Hardware devices
- **Multitasking**
- Concurrent processing in one application

# Concurrency

- Take advantage of multiple processors
- Hardware devices
- Multitasking
- **Concurrent processing in one application**

# Interprocess Communication

- Message passing or shared memory?
- Message passing harder to integrate into Mesa type system
- Shared memory equivalent to message passing

# Interprocess Communication

- Message passing or shared memory?
- Message passing harder to integrate into Mesa type system
- Shared memory equivalent to message passing

# Interprocess Communication

- Message passing or shared memory?
- Message passing harder to integrate into Mesa type system
- Shared memory equivalent to message passing

# Synchronization

- Shared memory communication requires synchronization
  - Possibilities: non-preemption, semaphores
  - Non-preemption unacceptable
  - Explicit semaphores have too little structure

# Synchronization

- Shared memory communication requires synchronization
- Possibilities: non-preemption, semaphores
- Non-preemption unacceptable
- Explicit semaphores have too little structure

# Synchronization

- Shared memory communication requires synchronization
- Possibilities: non-preemption, semaphores
- **Non-preemption unacceptable**
- Explicit semaphores have too little structure

# Synchronization

- Shared memory communication requires synchronization
- Possibilities: non-preemption, semaphores
- Non-preemption unacceptable
- **Explicit semaphores have too little structure**

## Solution: Monitors

- **Manage shared memory**
- Provide both synchronization and communication
- Could be integrated into Mesa
- Separation of data and interface fit with modular Mesa
- Well-established technique

## Solution: Monitors

- Manage shared memory
- Provide both synchronization and communication
- Could be integrated into Mesa
- Separation of data and interface fit with modular Mesa
- Well-established technique

## Solution: Monitors

- Manage shared memory
- Provide both synchronization and communication
- **Could be integrated into Mesa**
- Separation of data and interface fit with modular Mesa
- Well-established technique

## Solution: Monitors

- Manage shared memory
- Provide both synchronization and communication
- Could be integrated into Mesa
- Separation of data and interface fit with modular Mesa
- Well-established technique

## Solution: Monitors

- Manage shared memory
- Provide both synchronization and communication
- Could be integrated into Mesa
- Separation of data and interface fit with modular Mesa
- Well-established technique

## Solution: Monitors

- Manage shared memory
- Provide both synchronization and communication
- Could be integrated into Mesa
- Separation of data and interface fit with modular Mesa
- Well-established technique ...?

# Hoare monitor semantics

- Condition variables associated with conditions stronger than the monitor invariant
- Signalling process must guarantee condition
- Waiting process assumes condition
- Signalling process gives control to waiting process

## Hoare monitor semantics

- Condition variables associated with conditions stronger than the monitor invariant
- Signalling process must guarantee condition
- Waiting process assumes condition
- Signalling process gives control to waiting process

## Hoare monitor semantics

- Condition variables associated with conditions stronger than the monitor invariant
- Signalling process must guarantee condition
- **Waiting process assumes condition**
- Signalling process gives control to waiting process

## Hoare monitor semantics

- Condition variables associated with conditions stronger than the monitor invariant
- Signalling process must guarantee condition
- Waiting process assumes condition
- Signalling process gives control to waiting process

# Consequences of Hoare semantics

- Signalling process must be in monitor
- Signalling process releases lock, lets waiting process run
- Control must transfer directly to waiting process
- Signalling process resumes when waiting process done
- Requires forced context switch, unlocks/locks
- Condition checking decentralized

# Consequences of Hoare semantics

- Signalling process must be in monitor
- Signalling process releases lock, lets waiting process run
- Control must transfer directly to waiting process
- Signalling process resumes when waiting process done
- Requires forced context switch, unlocks/locks
- Condition checking decentralized

## Consequences of Hoare semantics

- Signalling process must be in monitor
- Signalling process releases lock, lets waiting process run
- **Control must transfer directly to waiting process**
- Signalling process resumes when waiting process done
- Requires forced context switch, unlocks/locks
- Condition checking decentralized

## Consequences of Hoare semantics

- Signalling process must be in monitor
- Signalling process releases lock, lets waiting process run
- Control must transfer directly to waiting process
- Signalling process resumes when waiting process done
- Requires forced context switch, unlocks/locks
- Condition checking decentralized

## Consequences of Hoare semantics

- Signalling process must be in monitor
- Signalling process releases lock, lets waiting process run
- Control must transfer directly to waiting process
- Signalling process resumes when waiting process done
- Requires forced context switch, unlocks/locks
- Condition checking decentralized

## Consequences of Hoare semantics

- Signalling process must be in monitor
- Signalling process releases lock, lets waiting process run
- Control must transfer directly to waiting process
- Signalling process resumes when waiting process done
- Requires forced context switch, unlocks/locks
- Condition checking decentralized

## Mesa monitor semantics

- **Notifying process makes no guarantee of condition**
- Waiting process must check condition after resume
- Notifying process retains lock, completes monitor call
- Waiting process can resume after notifying process returns
- Notifying process need not be in monitor

## Mesa monitor semantics

- Notifying process makes no guarantee of condition
- **Waiting process must check condition after resume**
- Notifying process retains lock, completes monitor call
- Waiting process can resume after notifying process returns
- Notifying process need not be in monitor

## Mesa monitor semantics

- Notifying process makes no guarantee of condition
- Waiting process must check condition after resume
- **Notifying process retains lock, completes monitor call**
- Waiting process can resume after notifying process returns
- Notifying process need not be in monitor

## Mesa monitor semantics

- Notifying process makes no guarantee of condition
- Waiting process must check condition after resume
- Notifying process retains lock, completes monitor call
- **Waiting process can resume after notifying process returns**
- Notifying process need not be in monitor

## Mesa monitor semantics

- Notifying process makes no guarantee of condition
- Waiting process must check condition after resume
- Notifying process retains lock, completes monitor call
- Waiting process can resume after notifying process returns
- **Notifying process need not be in monitor**

# Code changes

- Hoare monitors: if not condition, wait for condition variable
- Mesa monitors: *while* not condition, wait on condition variable
- Nothing else required
- Many practical improvements for implementation

## Code changes

- Hoare monitors: if not condition, wait for condition variable
- Mesa monitors: *while* not condition, wait on condition variable
- Nothing else required
- Many practical improvements for implementation

## Code changes

- Hoare monitors: if not condition, wait for condition variable
- Mesa monitors: *while* not condition, wait on condition variable
- **Nothing else required**
- Many practical improvements for implementation

## Code changes

- Hoare monitors: if not condition, wait for condition variable
- Mesa monitors: *while* not condition, wait on condition variable
- Nothing else required
- Many practical improvements for implementation

## Simpler notification

- **Notify never incorrect, just a performance issue**
- Broadcast notify
- Can periodically wake up waiters with no notify
- Condition checking centralized

## Simpler notification

- Notify never incorrect, just a performance issue
- **Broadcast notify**
- Can periodically wake up waiters with no notify
- Condition checking centralized

## Simpler notification

- Notify never incorrect, just a performance issue
- Broadcast notify
- Can periodically wake up waiters with no notify
- Condition checking centralized

## Simpler notification

- Notify never incorrect, just a performance issue
- Broadcast notify
- Can periodically wake up waiters with no notify
- **Condition checking centralized**

# Dynamic monitors and modularity

- Hoare monitors static, associated with specific variables
- Needed to create monitors dynamically
- Mesa: “monitored record”
- Examples: row-level locking, per-file locking
- Precursor to modern object-oriented monitors

## Dynamic monitors and modularity

- Hoare monitors static, associated with specific variables
- **Needed to create monitors dynamically**
- Mesa: “monitored record”
- Examples: row-level locking, per-file locking
- Precursor to modern object-oriented monitors

## Dynamic monitors and modularity

- Hoare monitors static, associated with specific variables
- Needed to create monitors dynamically
- Mesa: “monitored record”
- Examples: row-level locking, per-file locking
- Precursor to modern object-oriented monitors

## Dynamic monitors and modularity

- Hoare monitors static, associated with specific variables
- Needed to create monitors dynamically
- Mesa: “monitored record”
- Examples: row-level locking, per-file locking
- Precursor to modern object-oriented monitors

## Dynamic monitors and modularity

- Hoare monitors static, associated with specific variables
- Needed to create monitors dynamically
- Mesa: “monitored record”
- Examples: row-level locking, per-file locking
- Precursor to modern object-oriented monitors

# Monitors and exceptions

- **Mesa has exception handling**
  - Exceptions can unwind the stack
  - Monitors need to clean up after themselves
  - Mesa manages monitor lock

# Monitors and exceptions

- Mesa has exception handling
- Exceptions can unwind the stack
- Monitors need to clean up after themselves
- Mesa manages monitor lock

# Monitors and exceptions

- Mesa has exception handling
- Exceptions can unwind the stack
- **Monitors need to clean up after themselves**
- Mesa manages monitor lock

# Monitors and exceptions

- Mesa has exception handling
- Exceptions can unwind the stack
- Monitors need to clean up after themselves
- **Mesa manages monitor lock**

# Monitors and priorities

- Monitors can cause one process to wait on another
- Can lead to priority inversion
- Solved by priority boosting

# Monitors and priorities

- Monitors can cause one process to wait on another
- Can lead to priority inversion
- Solved by priority boosting

# Monitors and priorities

- Monitors can cause one process to wait on another
- Can lead to priority inversion
- Solved by priority boosting

## Multiple monitor subtlety

- **Monitor A calls monitor B**
- Monitor B waits
- Monitor B releases lock
- Monitor A holds lock
- Reasoning: A doesn't know B's internals, so A doesn't establish invariant

## Multiple monitor subtlety

- Monitor A calls monitor B
- **Monitor B waits**
- Monitor B releases lock
- Monitor A holds lock
- Reasoning: A doesn't know B's internals, so A doesn't establish invariant

## Multiple monitor subtlety

- Monitor A calls monitor B
- Monitor B waits
- **Monitor B releases lock**
- Monitor A holds lock
- Reasoning: A doesn't know B's internals, so A doesn't establish invariant

## Multiple monitor subtlety

- Monitor A calls monitor B
- Monitor B waits
- Monitor B releases lock
- **Monitor A holds lock**
- Reasoning: A doesn't know B's internals, so A doesn't establish invariant

## Multiple monitor subtlety

- Monitor A calls monitor B
- Monitor B waits
- Monitor B releases lock
- Monitor A holds lock
- Reasoning: A doesn't know B's internals, so A doesn't establish invariant

# Efficiency

- Other processes may run first
- No forced context switch
- No lock handoffs
- Notifying process finishes
- Can notify in time-critical routines

# Efficiency

- Other processes may run first
- **No forced context switch**
- No lock handoffs
- Notifying process finishes
- Can notify in time-critical routines

# Efficiency

- Other processes may run first
- No forced context switch
- **No lock handoffs**
- Notifying process finishes
- Can notify in time-critical routines

# Efficiency

- Other processes may run first
- No forced context switch
- No lock handoffs
- **Notifying process finishes**
- Can notify in time-critical routines

# Efficiency

- Other processes may run first
- No forced context switch
- No lock handoffs
- Notifying process finishes
- Can notify in time-critical routines

# Handling hardware devices

- **Interrupts and interrupt handlers**
- Different concurrency model
- Can bridge to native concurrency model (Linux: bottom half)
- In Pilot/Mesa: use monitors
- Bare notify for hardware events

# Handling hardware devices

- Interrupts and interrupt handlers
- Different concurrency model
- Can bridge to native concurrency model (Linux: bottom half)
- In Pilot/Mesa: use monitors
- Bare notify for hardware events

# Handling hardware devices

- Interrupts and interrupt handlers
- Different concurrency model
- Can bridge to native concurrency model (Linux: bottom half)
- In Pilot/Mesa: use monitors
- Bare notify for hardware events

## Handling hardware devices

- Interrupts and interrupt handlers
- Different concurrency model
- Can bridge to native concurrency model (Linux: bottom half)
- **In Pilot/Mesa: use monitors**
- Bare notify for hardware events

## Handling hardware devices

- Interrupts and interrupt handlers
- Different concurrency model
- Can bridge to native concurrency model (Linux: bottom half)
- In Pilot/Mesa: use monitors
- Bare notify for hardware events

# Hoare monitors versus Mesa monitors

## Hoare monitors

- Signal indicates condition is true
- Signalling process must guarantee condition
- Signalling process must hold lock
- Waiting process can assume condition
- if not condition, wait
- Sound theoretical foundation

## Mesa monitors

- Notify hints that condition may be true
- Notifying process makes no guarantee
- Notifying process need not hold lock
- Waiting process must check condition
- while not condition, wait
- Refinements for practical use

# Hoare monitors versus Mesa monitors

## Hoare monitors

- Signal indicates condition is true
- Signalling process must guarantee condition
- Signalling process must hold lock
- Waiting process can assume condition
- if not condition, wait
- Sound theoretical foundation

## Mesa monitors

- Notify hints that condition may be true
- Notifying process makes no guarantee
- Notifying process need not hold lock
- Waiting process must check condition
- while not condition, wait
- Refinements for practical use

# Hoare monitors versus Mesa monitors

## Hoare monitors

- Signal indicates condition is true
- Signalling process must guarantee condition
- Signalling process must hold lock
- Waiting process can assume condition
- if not condition, wait
- Sound theoretical foundation

## Mesa monitors

- Notify hints that condition may be true
- Notifying process makes no guarantee
- Notifying process need not hold lock
- Waiting process must check condition
- while not condition, wait
- Refinements for practical use

# Hoare monitors versus Mesa monitors

## Hoare monitors

- Signal indicates condition is true
- Signalling process must guarantee condition
- Signalling process must hold lock
- **Waiting process can assume condition**
- if not condition, wait
- Sound theoretical foundation

## Mesa monitors

- Notify hints that condition may be true
- Notifying process makes no guarantee
- Notifying process need not hold lock
- **Waiting process must check condition**
- while not condition, wait
- Refinements for practical use

# Hoare monitors versus Mesa monitors

## Hoare monitors

- Signal indicates condition is true
- Signalling process must guarantee condition
- Signalling process must hold lock
- Waiting process can assume condition
- if not condition, wait
- Sound theoretical foundation

## Mesa monitors

- Notify hints that condition may be true
- Notifying process makes no guarantee
- Notifying process need not hold lock
- Waiting process must check condition
- while not condition, wait
- Refinements for practical use

# Hoare monitors versus Mesa monitors

## Hoare monitors

- Signal indicates condition is true
- Signalling process must guarantee condition
- Signalling process must hold lock
- Waiting process can assume condition
- if not condition, wait
- **Sound theoretical foundation**

## Mesa monitors

- Notify hints that condition may be true
- Notifying process makes no guarantee
- Notifying process need not hold lock
- Waiting process must check condition
- while not condition, wait
- **Refinements for practical use**