Deadlock Detection and Recovery

- Allow deadlocks to occur
- Detect them once they have happened
  - Preferably as soon as possible after they occur
- Do something to break the deadlock and allow someone to make progress
- Is this a good approach?
  - Either in general or when you don't want to avoid or prevent

Implementing Deadlock Detection

- Need to identify all resources that can be locked
- Need to maintain wait-for graph or equivalent structure
- When lock requested, structure is updated and checked for deadlock
  - In which case, might it not be better just to reject the lock request?
  - And not let the requester block?

Deadlock Detection and Health Monitoring

- Deadlock detection seldom makes sense
  - It is extremely complex to implement
  - Only detects "true deadlocks" for a known resources
  - Not always clear cut what you should do if you detect one
- Service/application "health monitoring" makes more sense
  - Monitor application progress/submit test transactions
  - If response takes too long, declare service "hung"
- Health monitoring is easy to implement
- It can detect a wide range of problems
  - Deadlocks, live-locks, infinite loops & waits, crashes

Related Problems Health Monitoring Can Handle

- Live-lock
  - Process is running, but won't free R1 until it gets message
  - Process that will send the message is blocked for R1
- Sleeping Beauty, waiting for "Prince Charming"
  - A process is blocked, awaiting some completion
  - But, for some reason, it will never happen
- Neither of these is a true deadlock
  - Wouldn't be found by deadlock detection algorithm
  - Both leave the system just as hung as a deadlock
- Health monitoring handles them

How To Monitor Process Health

- Look for obvious failures
  - Process exits or core dumps
- Passive observation to detect hangs
  - Is process consuming CPU time, or is it blocked?
  - Is process doing network and/or disk I/O?
- External health monitoring
  - "Pings", null requests, standard test requests
- Internal instrumentation
  - White box audits, exercisers, and monitoring

What To Do With "Unhealthy" Processes?

- Kill and restart "all of the affected software"
- How many and which processes to kill?
  - As many as necessary, but as few as possible
  - The hung processes may not be the ones that are broken
- How will kills and restarts affect current clients?
  - That depends on the service APIs and/or protocols
  - Apps must be designed for cold/warm/partial restarts
- Highly available systems define restart groups
  - Groups of processes to be started/killed as a group
  - Define inter-group dependencies (restart B after A)

Failure Recovery Methodology

- Retry if possible ... but not forever
  - Client should not be kept waiting indefinitely
  - Resources are being held while waiting to retry
- Roll-back failed operations and return an error
- Continue with reduced capacity or functionality
  - Accept requests you can handle, reject those you can't
- Automatic restarts (cold, warm, partial)
- Escalation mechanisms for failed recoveries
  - Restart more groups, reboot more machines

Priority Inversion and Deadlock

- Priority inversion isn't necessarily deadlock, but it's related
  - A low priority process P1 has mutex M1 and is preempted
  - A high priority process P2 blocks for mutex M1
  - Process P2 is effectively reduced to priority of P1
- Solution: mutex priority inheritance
  - Check for problem when blocking for mutex
  - Compare priority of current mutex owner with blocker
  - Temporarily promote holder to blocker's priority
  - Return to normal priority after mutex is released

Priority Inversion on Mars

- A real priority inversion problem occurred on the Mars Pathfinder rover
- Caused serious problems with system resets
- Difficult to find

The Pathfinder Priority Inversion

- Special purpose hardware running VxWorks real time OS
- Used preemptive priority scheduling
  - So a high priority task should get the processor
- Multiple components shared an "information bus"
  - Used to communicate between components
  - Essentially a shared memory region
  - Protected by a mutex

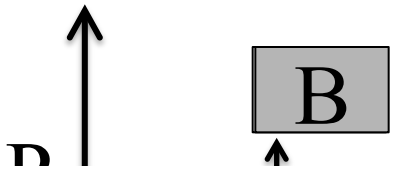
A Tale of Three Tasks

- A high priority bus management task (at P1) needed to run frequently
  - For brief periods, during which it locked the bus
- A low priority meteorological task (at P3) ran occasionally
  - Also for brief periods, during which it locked the bus
- A medium priority communications task (at P2) ran rarely
  - But for a long time when it ran
  - But it didn't use the bus, so it didn't need the lock
- P1>P2>P3

What Went Wrong?

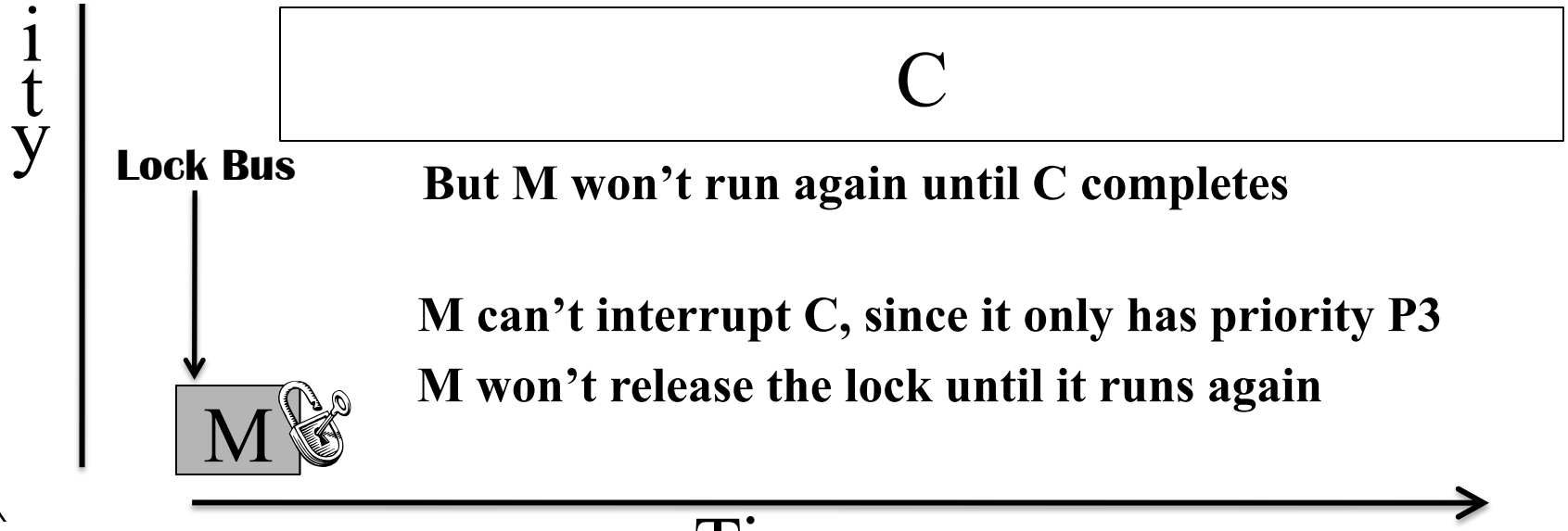
- Rarely, the following happened:
  - The meteorological task ran and acquired the lock
  - And then the bus management task would run
  - It would block waiting for the lock
    - Don't pre-empt low priority if you're blocked anyway
- Since meteorological task was short, usually not a problem
- But if the long communications task woke up in that short interval, what would happen?

The Priority Inversion at Work



B's priority of P1 is higher than C's, but B can't run because it's waiting on a lock held by M

A HIGH PRIORITY TASK DOESN'T RUN AND A LOW PRIORITY TASK DOES



But M won't run again until C completes

M can't interrupt C, since it only has priority P3

M won't release the lock until it runs again

The Ultimate Effect

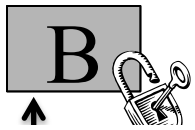
- A watchdog timer would go off every so often
  - At a high priority
  - It didn't need the bus
  - A health monitoring mechanism
- If the bus management task hadn't run for a long time, something was wrong
- So the watchdog code reset the system
- Every so often, the system would reboot

Solving the Problem

- This was a priority inversion
  - The lower priority communications task ran before the higher priority bus management task
- That needed to be changed
- How?
- Temporarily increase the priority of the meteorological task
  - While the high priority bus management task was block by it
  - So the communications task wouldn't preempt it
  - *Priority inheritance*: a general solution to this kind of problem

The Fix in Action

When M releases the lock it loses high



Tasks run in proper priority order and Pathfinder can keep exploring Mars!