Real-Time Programming

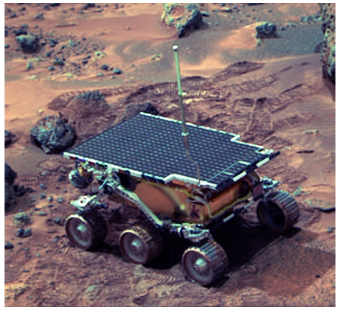
Farhang Nemati
Spring 2016

Repetition

- Scheduling with Precedence Constraints
  - LDF
  - EDF*
- Periodic Task Scheduling
  - Time Line Scheduling
  - Rate Monotonic Scheduling
  - EDF
- Jitter

Resource Access Protocols

- Mars Pathfinder



Mars Pathfinder

- An American spacecraft landed on Mars 1997. Its mission was to send meteorological data from Mars
- Its computing system was running on VxWorks
- A few days after it started its mission the system was experiencing resets causing the meteorological data being lost
- What was the problem?

Mars Pathfinder

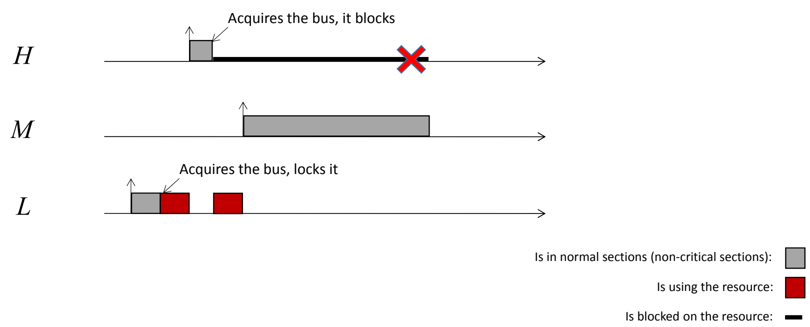
- The problem of Mars Pathfinder
  - There was a communication bus on the system used to transfer data among different components on Mars Pathfinder which could be used by one task at a time (mutual exclusive resource)
  - 3 tasks with different priorities were using the bus which was protected by a Mutex:
    - A bus management task with high priority
    - A not frequent but long running communication task with medium priority
    - A not frequent task for sending meteorological data with low priority which would acquire the mutex when using the bus

Mars Pathfinder

- The problem of Mars Pathfinder (Cont'd)
  - Most of the time the combination of the 3 tasks was working fine
  - However, in very infrequent times while the low priority task (meteorological data task) was using the bus (locking the mutex), the higher priority task (bus management task) would arrive and ask for the mutex and thus would block and wait. In this situation the medium priority task (communication task) would arrive and preempt lower priority task. The medium priority task would run for a long time and keeping lower priority task from releasing the bus. Thus the higher priority task would be blocked too long.
  - After sometime a watchdog timer would notice that the higher priority task has been running for too long time and would conclude that some thing had gone drastically wrong and would reset the system.
  - This problem is known as Priority Inversion

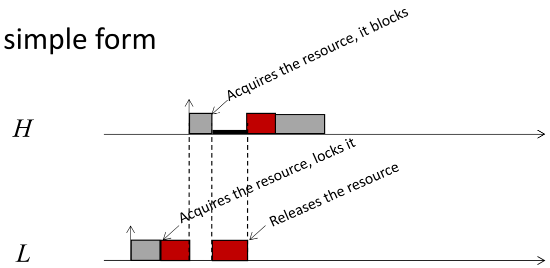
Mars Pathfinder

- The problem of Mars Pathfinder (Cont'd)



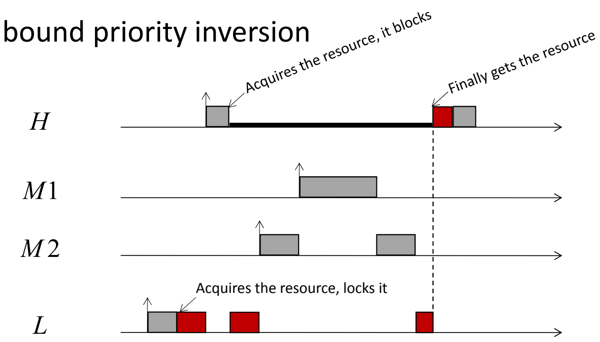
Priority Inversion

- The simple form



Priority Inversion

- Unbound priority inversion



Resource Access Protocols

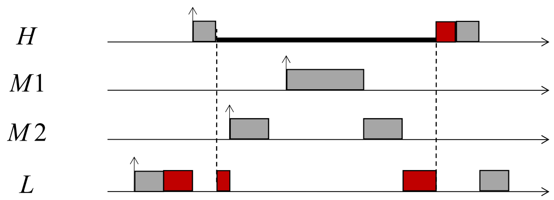
- How to overcome unbound priority inversion?
- Solution: Put some rules when tasks lock/unlock semaphores or mutexes
- Resource Sharing Protocols are used to apply the rules

Non-Preemptive Protocol (NPP)

- Protocol Rules
  1. Whenever a task locks a semaphore/mutex it runs non-preemptive, i.e., by boosting its priority to the highest priority
  2. Whenever a task releases the semaphore/mutex it becomes preemptive again, i.e., by returning to its original priority

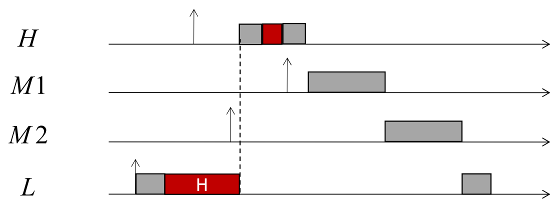
Non-Preemptive Protocol (NPP)

- Example
- Without the protocol:



Non-Preemptive Protocol (NPP)

- Example
- With the protocol:



Non-Preemptive Protocol (NPP)

- + Simple
- + Deadlock free
- + A task can be blocked by at most one lower priority task
- - Non preemption blocks all tasks including those that do not use any resource

Priority Inheritance Protocol (PIP)

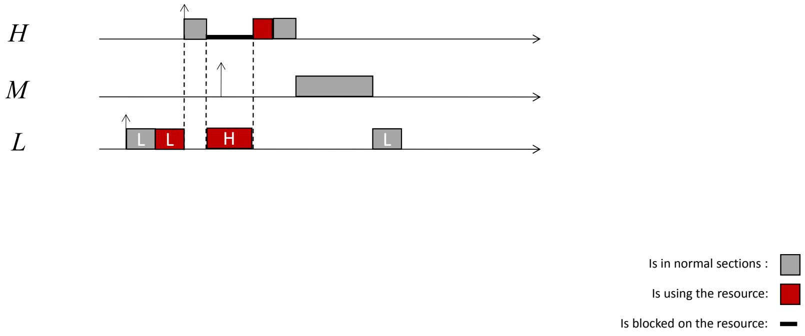
- Protocol Rules
  1. A task, τ_l, locks a semaphore S
  2. While τ_l is using the resource another task, τ_h ,with higher priority arrives preempts τ_l , and at some time instant acquires semaphore S too. It will block since S is locked
  3. τ_l inherits the priority of τ_h , and runs with the priority of τ_h. I.e., the priority of lower priority task is boosted to the priority of the higher priority task
  4. When τ_l releases semaphore S, it will get back its original priority
  5. PIP is transitive meaning if τ_l blocks τ_m and τ_m blocks τ_h then τ_l inherits the priority of τ_h

Priority Inheritance Protocol (PIP)

- To sum up: Any time a lower priority task blocks a higher priority task it inherits the priority of the higher priority task

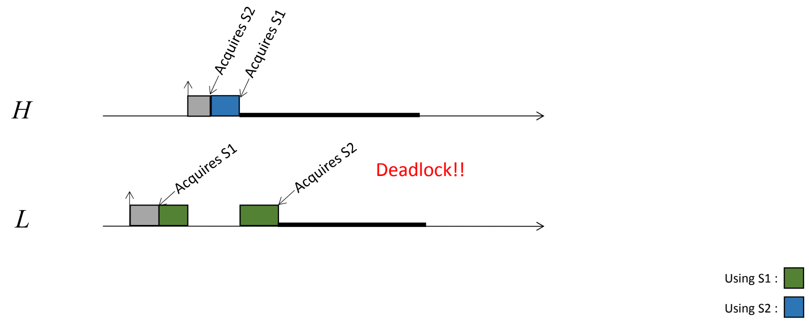
Priority Inheritance Protocol (PIP)

- Example



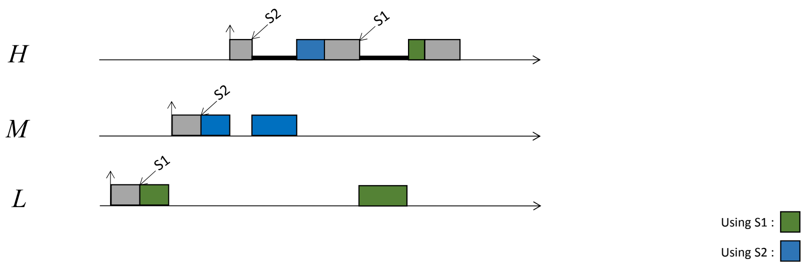
Priority Inheritance Protocol (PIP); Problems

- Deadlock



Priority Inheritance Protocol (PIP); Problems

- Multiple blockings



Priority Inheritance Protocol (PIP)

- + Has bounded priority inversion
- + No need to know resource usage of tasks in advance
- + Relatively good performance
- - Deadlock possible
- - Multiple blockings (Chained blocking)

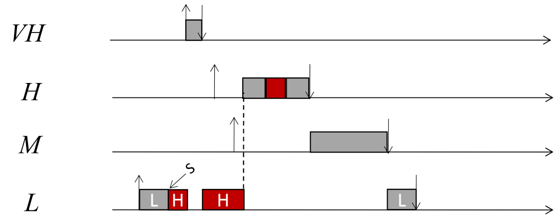
Highest Locker Priority Protocol (HLP)

- Also known as Immediate Priority Ceiling Protocol (IPC)
- Protocol Rules
  1. A ceiling is assigned to each resource (semaphore) which is the highest priority of all tasks that may use it

     $$ceil(R) = \max\{\rho_i \mid \tau_i \ uses \ R\}$$
  2. Whenever a task starts using the resource (locks the semaphore) its priority is immediately boosted to the ceiling of the resource

Highest Locker Priority Protocol (HLP)

- Example: Tasks with priority H and L use S: $ceil(S) = H$



Highest Locker Priority Protocol (HLP)

- Maximum Blocking time of a task τ_i denoted by B_i
- τ_i can be blocked by at most one lower priority task that has priority ceiling higher or equal to τ_i
  - B_i equals to the duration of largest critical section belonging to any lower priority task that has priority ceiling higher than or equal to τ_i
  - Let $CS_j(R)$ denote the duration of longest critical section of task τ_j in which it uses resource R. Let ρ_i denote the priority of task τ_i

$$B_i = \max\{CS_j(R) \mid \rho_j < \rho_i \ \ and \ \ \rho_i \leq ceil(R)\}$$

Highest Locker Priority Protocol (HLP)

- Maximum Response Time

$$R_i = B_i + e_i + \sum_{\tau_j \in H_i} \left\lceil \frac{R_i}{p_j} \right\rceil e_j$$

Highest Locker Priority Protocol (HLP)

- + Has bounded priority inversion
- + No Deadlock
- + No Multiple blockings; a task is blocked at most once
- + Relatively good performance

PIP and HLP in POSIX

```
int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
       *restrict attr, int *restrict protocol);

int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
       int protocol);
```

- protocol:
  - PTHREAD_PRIO_NONE
  - PTHREAD_PRIO_INHERIT: PIP
  - PTHREAD_PRIO_PROTECT: HLP

Priority Ceiling Protocol (PCP)

- Protocol Rules
  1. A ceiling is assigned to each resource (semaphore) which is the highest priority of all tasks that may use it
     $$ceil(R) = \max\{\rho_i \mid \tau_i \ uses \ R\}$$
  2. During run-time the System Ceiling is the highest ceiling among all resources that are currently locked
     $$sysceil = \max\{ceil(R) \mid R \text{ is locked } \}$$
  3. Whenever a task τ_i acquires a resource it can lock the resource only if its priority is strictly higher than the system ceiling; $\rho_i > sysceil$
  4. If $\rho_i \leq sysceil$ then τ_i is said to be blocked by task τ_j that has locked the resource with ceiling equal to $sysceil$. In this case τ_j inherits the priority of τ_i

Priority Ceiling Protocol (PCP)

- + Has bounded priority inversion
- + No Deadlock
- + No Multiple blockings; a task is blocked at most once
- + Better response times for higher priority tasks
- - Complex implementation