
EECE 276

Embedded Systems

Scheduling anomalies:
Priority inversion

Mars Sojourner (1996)

- Worked well for a while
- Suspicious hang-ups and reboots
- Glitches were also observed during ground testing, but were attributed to hardware
- Analysis revealed task restarts via watchdog timers



Task interactions

- Tasks:

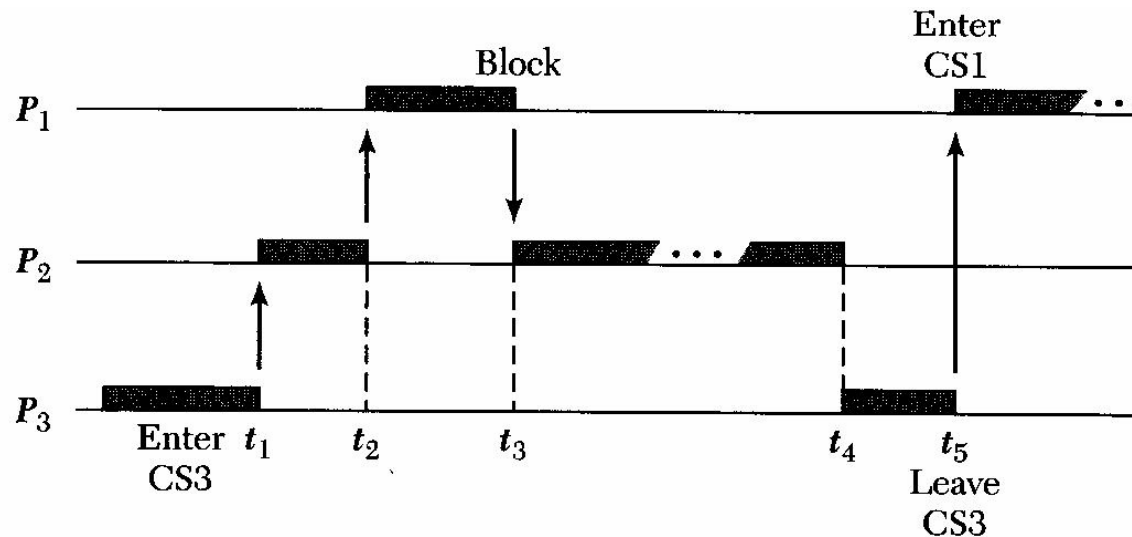
P1: { ... P(S1); CriticalSection1; V(S1); }

P2: { }

P3: { ... P(S1); CriticalSection3; V(S1); }

// S1 semaphore protects critical sections

Priority: $\text{pri}(P1) > \text{pri}(P2) > \text{pri}(P3)$



What's wrong?

Priority inversion

Low priority task blocks high priority one

Delay includes: execution time of P3 **and** P2

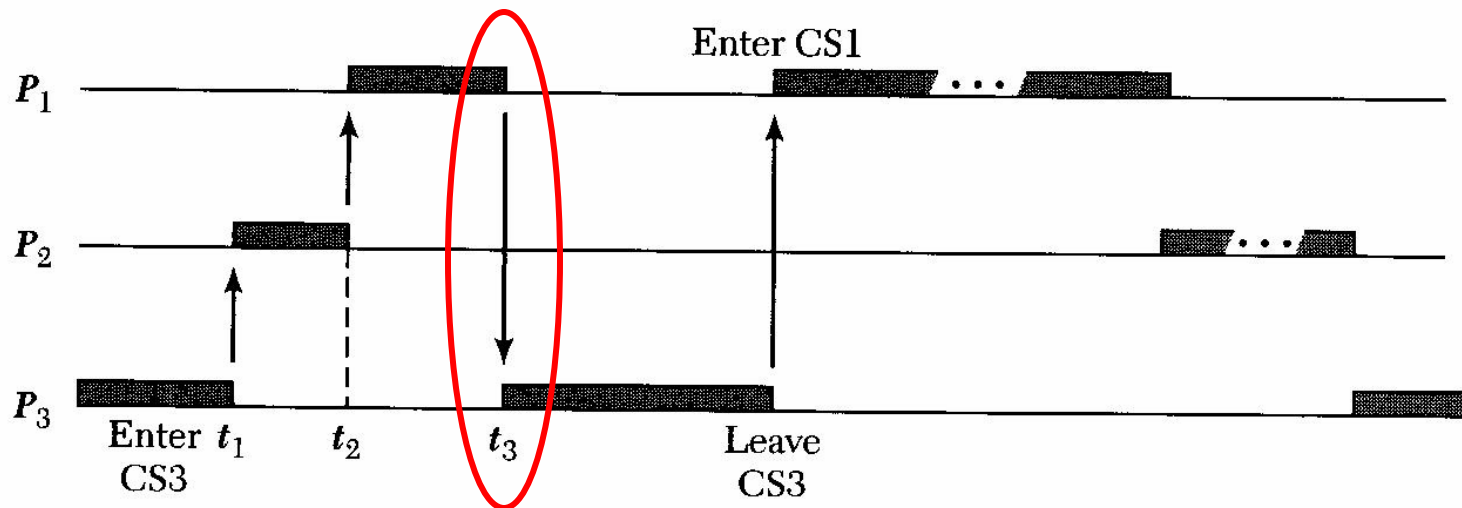
In general, P1 would have to wait for an unbounded period of time.

Simple solutions:

1. Make critical sections non-preemptable
 - Not practical as resources could be held for a long time.
2. Execute critical sections at the highest priority of the task that could use it.
 - Too strong (it always raises the priority of a task)

Priority inheritance protocol

Idea: temporarily raise the priority of a task only if and when it actually blocks a higher priority one. Upon leaving the C/S, priority reverts.



Worst-case blocking times: predictable but may be long.

Multi-level blocking

- Tasks:

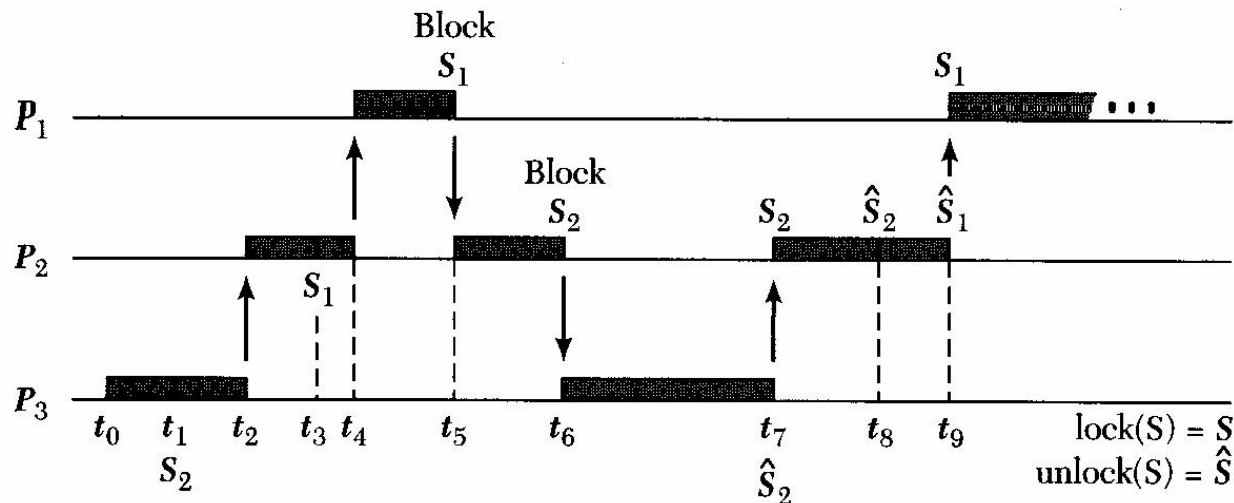
P1: { ... P(S1); CS1; V(S1); }

P2: { ... P(S1); CS21; P(S2); CS22; V(S2); CS23; V(S1); ... }

P3: { ... P(S2); CS3; V(S2); }

// S1,S2 semaphores protect critical sections

Priority: pri(P1) > pri(P2) > pri(P3)



Even with P/I,
P1 is blocked
for CS3,
CS21..CS23.

Priority inheritance and deadlock

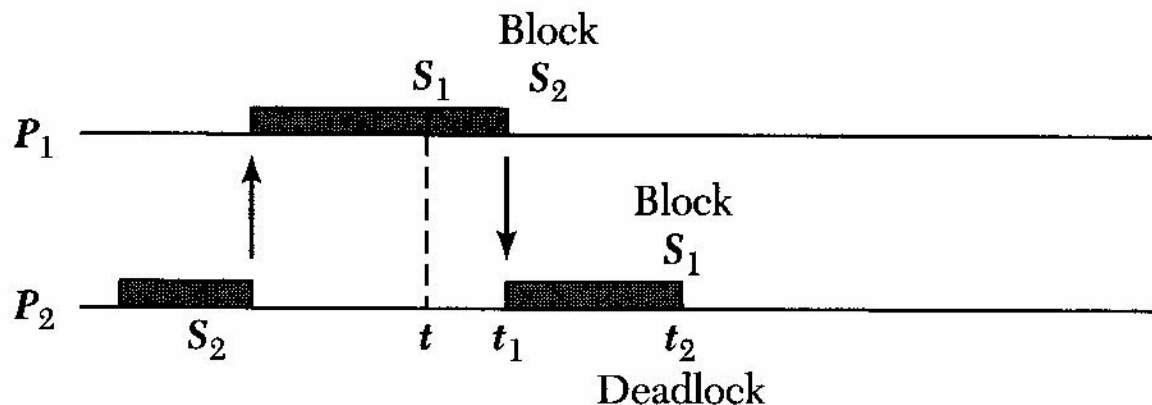
- Tasks:

$P_1: \{ \dots P(S_1); \dots; P(S_2); \dots V(S_2); \dots V(S_1); \dots \}$

$P_2: \{ \dots P(S_2); \dots; P(S_1); \dots V(S_1); \dots V(S_2); \dots \}$

// S1,S2 semaphores protect critical sections

Priority: $\text{pri}(P_1) > \text{pri}(P_2)$



Standard solution: all semaphores must be acquired in the same order – not practical

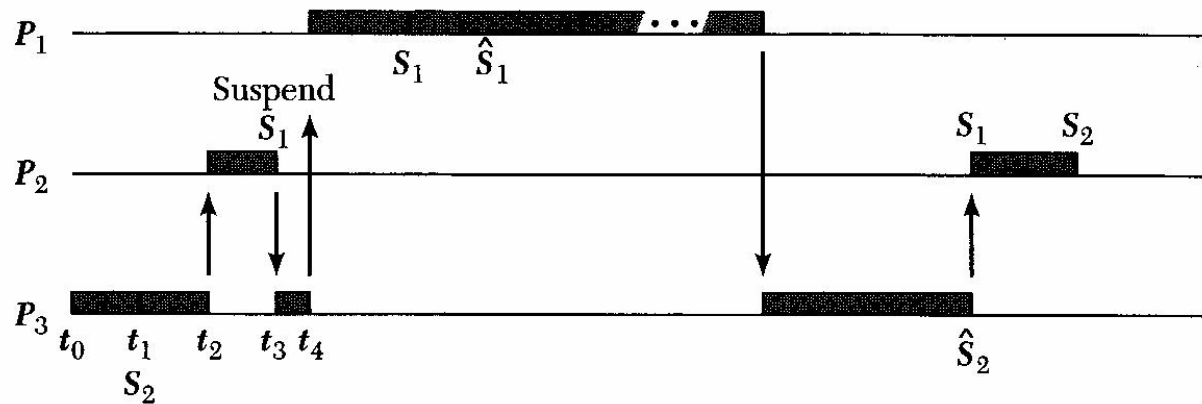
Priority ceiling protocol

- Priority Ceiling:
PC(S) = highest priority of all processes that may lock S.
- PC protocol:
A task P that attempts to lock a semaphore will be suspended unless its priority is *higher* than PC(S) **for all S currently locked** by all tasks Q != P.
- Example (3 tasks, 2 semaphores):
PC(S1) = max(pri(P1), pri(P2)) = pri(P1)
PC(S2) = max(pri(P2), pri(P3)) = pri(P2)

P/C Example

- 3 tasks, 2 semaphores

P2 is suspended at t_3 when it attempts to lock S_1 , because its priority is not higher than $PC(S_2)$. When P3 is resumed, it inherits $pri(P_2)$.



P/C: Deadlock example

- 2 tasks, overlapping semaphore locks
 $PC(S_1) = PC(S_2) = \max(\text{pri}(P_1), \text{pri}(P_2)) = \text{pri}(P_1)$
P1 gets suspended when it attempts to lock S1, P2 inherits $\text{pri}(P_1)$, etc.

