Handling shared resources

Problems caused by mutual exclusion

Using semaphores

- Make critical sections as short as possible.

```
int    x, y;  // these are global shared variables
mutex  s;     // this is the semaphore to protect them

task   reader() {
int    i;              // these are local variables
float  d, v[DIM];
       ...
       wait(s);
       d = sqrt(x*x + y*y);
       for (i=0; i++; i<DIM) {
            v[i] = i*(x + y);
            if (v[i] < x*y) v[i] = x + y;
       }
       signal(s);
       ...
}
```

Can we shorten this critical section?

critical section length

Using semaphores

- Make critical sections as short as possible.

```
task   reader() {
int    i;              // these are local variables
float  d, v[DIM];
float  a, b;           // two new local variables
       ...
       wait(s);        // copy global vars
       a = x; b = y;   // to local vars
       signal(s);
       d = sqrt(a*a + b*b);     // make computation
       for (i=0; i++; i<DIM) {  // using local vars
            v[i] = i*(a + b);
            if (v[i] < a*b) v[i] = a + b;
       }
       ...
}
```

critical section length

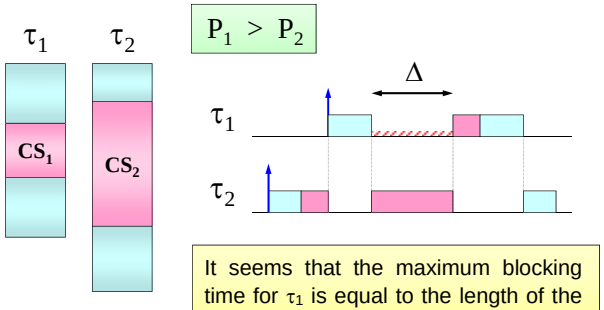
Using semaphores

- Make critical sections as short as possible.
- Try to avoid nested critical sections.
- Avoid making critical sections across loops or conditional statements.

This code is very UNSAFE, since the signal could never be executed, and τ_1 could be blocked forever!

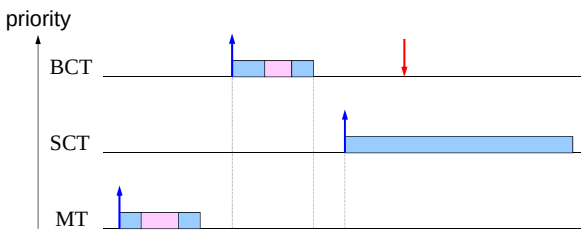
```
       ...
       wait(s);
       results = x + y;
       while (result > 0) {
              v[i] = i*(x + y);
              if (v[i] < x*y) results = results - y;
              else signal(s);
       }
}
```


How long is blocking time?

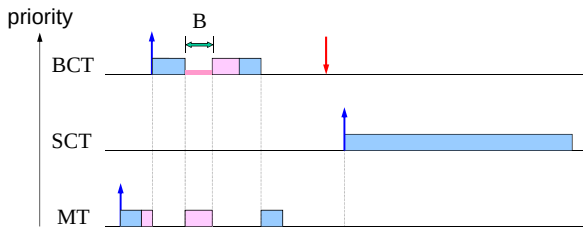


It seems that the maximum blocking time for τ_1 is equal to the length of the critical section (CS_2) of τ_2, but …

Schedule with no conflicts

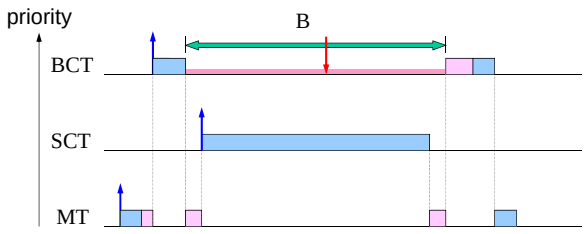


Conflict on a critical section



7

Conflict on a critical section



8

Priority Inversion

A high priority task is blocked by a lower priority task a for an unbounded interval of time

Solution

Introduce a concurrency control protocol for accessing critical sections.

9

Protocol key aspects

Access Rule:	Decides whether to block and when.
Progress Rule:	Decides how to execute inside a critical section.
Release Rule:	Decides how to order the pending requests of the blocked tasks.

Other aspects

Analysis:	estimates the worst-case blocking times.
Implementation:	finds the simplest way to encode the protocol rules.

10

Rules for classical semaphores

The following rules are normally used for classical semaphores:

Access Rule (Decides whether to block and when):

- Enter a critical section if the resource is free, block if the resource is locked.

Progress Rule (Decides how to execute in a critical section):

- Execute the critical section with the nominal priority.

Release Rule (Decides how to order pending requests):

- Wake up the blocked task in FIFO order.
- Wake up the blocked task with the highest priority.

11

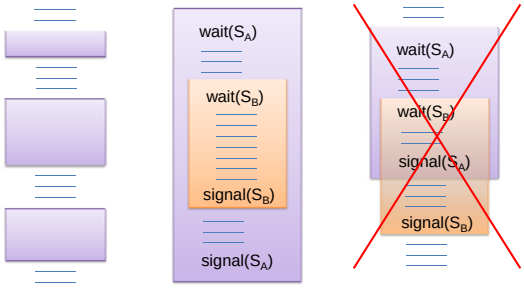
Resource Access Protocols

- Classical semaphores (No protocol)
- Non Preemptive Protocol (**NPP**)
- Highest Locker Priority (**HLP**)
- Priority Inheritance Protocol (**PIP**)
- Priority Ceiling Protocol (**PCP**)
- Stack Resource Policy (**SRP**)

12

Assumption

Critical sections are correctly accessed by tasks:



13

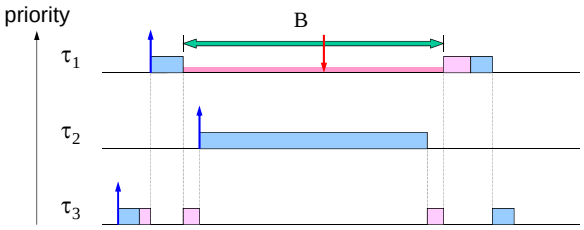
Non Preemptive Protocol

Access Rule:	A task never blocks at the entrance of a critical section, but at its activation time.
Progress Rule:	Disable preemption when executing inside a critical section.
Release Rule:	At exit, enable preemption so that the resource is assigned to the pending task with the highest priority.

14

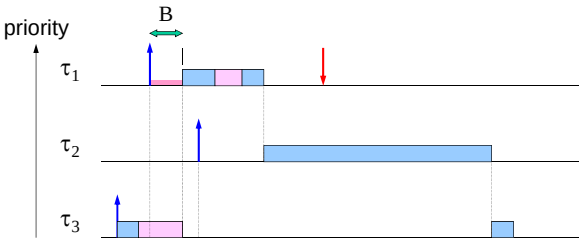
Conflict on a critical section

(using classical semaphores)



15

NPP: example



16

NPP: implementation notes

Each task τ_i must be assigned two priorities:

- a nominal priority P_i (fixed) assigned by the application developer;
- a dynamic priority p_i (initialized to P_i) used to schedule the task and affected by the protocol.

Then, the protocol can be implemented by changing the behavior of the wait and signal primitives:

wait(s): $p_i = \max(P_1, \ldots, P_n)$

signal(s): $p_i = P_i$

17

NPP: pro & cons

ADVANTAGES: simplicity and efficiency.

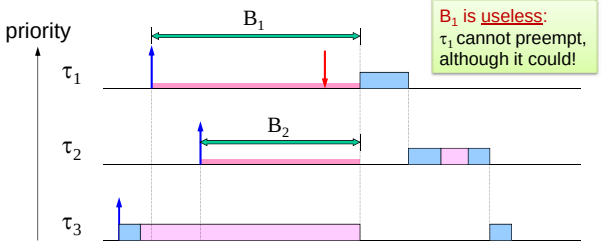
- Semaphores queues are not needed, because tasks never block on a wait(s).
- Each task can block at most on a single critical section.
- It prevents deadlocks and allows stack sharing.
- It is transparent to the programmer.

PROBLEMS:

1. Tasks may block even if they do not use resources.
2. Since tasks are blocked at activation, blocking could be unnecessary (pessimistic assumption).

NPP: problem 1

Long critical sections delay all high priority tasks:



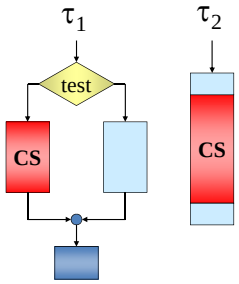
B_1 is useless: τ_1 cannot preempt, although it could!

Priority assigned to τ_i inside critical sections: $p_i = P_{max} = \max(P_1, \ldots, P_n)$

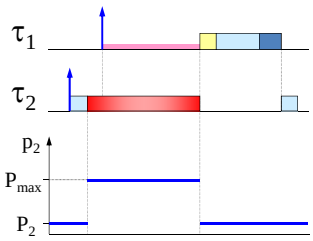
19

NPP: problem 2

A task could block even if not accessing a critical section:



τ_1 blocks just in case ...



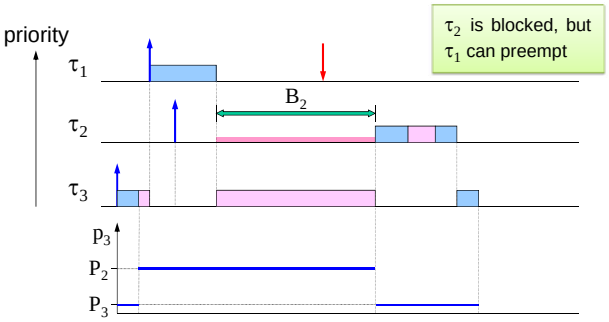
20

Highest Locker Priority

Access Rule:	A task never blocks at the entrance of a critical section, but at its activation time.
Progress Rule:	Inside resource R, a task executes at the highest priority of the tasks that use R.
Release Rule:	At exit, the dynamic priority of the task is reset to its nominal priority P_i.

21

HLP: example



τ_2 is blocked, but τ_1 can preempt

Priority assigned to τ_i inside a resource R: $p_i(R) = max\{P_j \mid \tau_j \text{ uses } R\}$

22

HLP: implementation notes

- Each task τ_i is assigned a nominal priority P_i and a dynamic priority p_i.
- Each semaphore S is assigned a resource ceiling $C(S)$:

$$C(S) = max\{P_i \mid \tau_i \text{ uses } S\}$$

Then, the protocol can be implemented by changing the behavior of the wait and signal primitives:

wait(S): $p_i = C(S)$

signal(S): $p_i = P_i$

Note: **HLP** is also known as Immediate Priority Ceiling (**IPC**).

23

HLP: pro & cons

ADVANTAGES: simplicity and efficiency.

- Semaphores queues are not needed, because tasks never block on a wait(s).
- Each task can block at most on a single critical section.
- It prevents deadlocks.
- It allows stack sharing.

PROBLEMS:

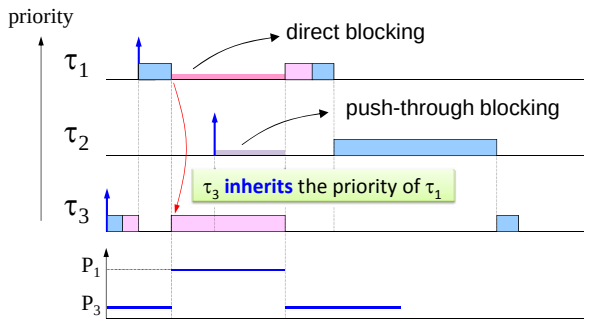
- Since tasks are blocked at activation, blocking could be unnecessary (same pessimism as for NPP).
- It is not transparent to programmers (due to ceilings).

Priority Inheritance Protocol

Access Rule:	A task blocks at the entrance of a critical section if the resource is locked.
Progress Rule:	Inside resource R, a task executes with the highest priority of the tasks blocked on R.
Release Rule:	At exit, the dynamic priority of the task is reset to its nominal priority P_i.

25

PIP: example



26

PIP: types of blocking

- **Direct blocking**

  A task blocks on a locked semaphore

- **Indirect blocking (Push-through blocking)**

  A task blocks because a lower priority task inherited a higher priority.

BLOCKING:
a delay caused by lower priority tasks

27

PIP: implementation notes

Inside a resource R the dynamic priority p_i is set as

$$p_i(R) = max \{ P_h \mid \tau_h \textit{ blocked on } R \}$$

wait(s):
```
if (s == 0) {
    <suspend the calling task τexe in the semaphore queue>
    <find the task τk that is locking the semaphore s>
    pk = Pexe          // priority inheritance
    <call the scheduler>
}
else s = 0;
```

signal(s):
```
if (there are blocked tasks) {
    <awake the highest priority task in the semaphore queue>
    pexe = Pexe
    <call the scheduler>
}
else s = 1;
```

Identifying blocking resources

Under PIP, a task τ_i can be blocked on a semaphore S_k only if:

1. S_k is directly shared between τ_i and lower priority tasks (direct blocking)

OR

2. S_k is shared between tasks with priority lower than τ_i and tasks having priority higher than τ_i (push-through blocking).

29

Identifying blocking resources

Lemma 1: A task τ_i can be blocked at most once by a lower priority task.

If there are n_i tasks with priority lower than τ_i, then τ_i can be blocked at most at most n_i times, independently of the number of critical sections that can block τ_i.

30

Identifying blocking resources

Lemma 2: A task τ_i can be blocked at most once on a semaphore S_k.

If there are m_i distinct semaphores that can block a task τ_i, then τ_i can be blocked at most m_i times, independently of the number of critical sections that can block τ_i.

Bounding blocking times

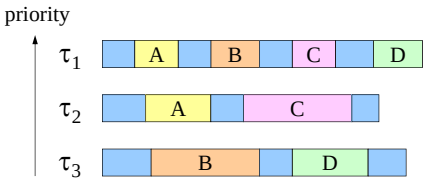
A theorem follows from the previous lemmas:

Theorem: τ_i can be blocked at most for the duration of $\alpha_i = \min(n_i, m_i)$ critical sections.

n_i = number of tasks with priority less than τ_i

m_i = number of semaphores that can block τ_i (either directly or indirectly).

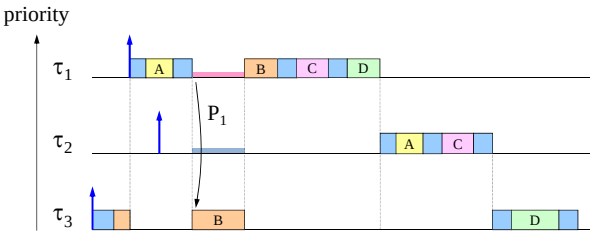
Example 1



- τ_1 can be blocked once by τ_2 (on A_2 or C_2) and once by τ_3 (on B_3 or D_3)
- τ_2 can be blocked once by τ_3 (on B_3 or D_3)
- τ_3 cannot be blocked

Example 1

Example in which τ_2 is blocked on B_3 by push-through



Identifying blocking times

To derive a general analysis, we define the following notation:

Z_{ik}	longest (external) critical section used by τ_i protected by semaphore S_k.
δ_{ik}	worst-case duration of Z_{ik}
σ_i	set of the longest critical sections used by τ_i for each semaphore S_k: $\sigma_i = \{ Z_{ik} \mid \forall S_k$ used by $\tau_i \}$
β_{ij}	set of critical sections used by τ_j that can block τ_i
β_i	set of critical sections that can block τ_i
α_i	maximum number of critical sections that can block τ_i
B_i	worst-case blocking time for τ_i

Identifying blocking times

C.S. of τ_j that can block τ_i for direct blocking:

$$\beta_{ij}^{dir} = \sigma_i \cap \sigma_j$$

C.S. of τ_j that can block τ_i for push-through blocking:

$$\beta_{ij}^{pt} = \bigcup_{h=1}^{i-1} \{\sigma_h \cap \sigma_j\}$$

C.S. of τ_j that can block τ_i

$$\beta_{ij} = \beta_{ij}^{dir} \cup \beta_{ij}^{pt} = \bigcup_{h=1}^{i} \{\sigma_h \cap \sigma_j\}$$

C.S. that can block τ_i

$$\beta_i = \bigcup_{j=i+1}^{n} \beta_{ij}$$

For the other protocols

$$\beta_{ij}^{NPP} = \{Z_{jk} \mid (Z_{jk} \in \sigma_j) \text{ AND } (P_j < P_i)\}$$

$$\beta_{ij}^{HLP} = \{Z_{jk} \mid (Z_{jk} \in \sigma_j) \text{ AND } (P_j < P_i) \text{ AND } (C(S_k) \geq P_i)\}$$

$$\beta_{ij}^{PIP} = \bigcup_{h=1}^{i} \{\sigma_h \cap \sigma_j\}$$

C.S. that can block τ_i $\quad \beta_i = \bigcup_{j=i+1}^{n} \beta_{ij}$

Identifying blocking time B_i

1. Identify the set β_{ij} for all lower priority tasks
2. Identify the set β_i
3. Compute α_i
4. Compute B_i as the highest sum of the α_i durations δ_{ik} of $Z_{ik} \in \beta_i$

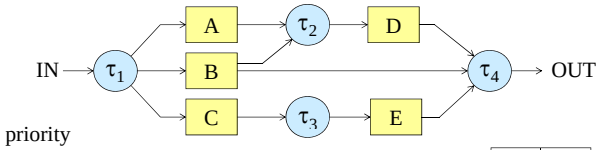
NOTE:

The α_i critical sections selected from β_i

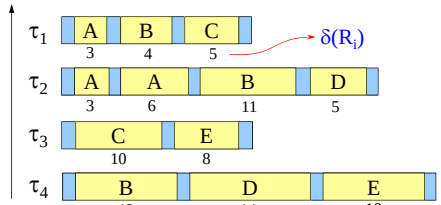
- must belong to different tasks (for Lemma 1);
- must refer to different semaphores (for Lemma 2);

Example 2

Consider the following application:

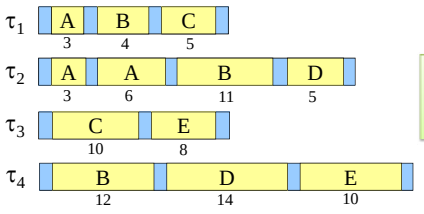


priority



	C_i	T_i
τ_1	15	60
τ_2	30	100
τ_3	20	150
τ_4	40	200

Example 2



From the task code we can derive the following table:

	C_i	T_i	A	B	C	D	E
τ_1	15	60	3	4	5	–	–
τ_2	30	100	6	11	–	5	–
τ_3	20	150	–	–	10	–	8
τ_4	40	200	–	12	–	14	10

Identification of β_1

B_1 →

	C_i	T_i	A	B	C	D	E
τ_1	15	60	3	4	5	–	–
τ_2	30	100	(6)	(11)	–	5	–
τ_3	20	150	–	–	(10)	–	8
τ_4	40	200	–	(12)	–	14	10

$$\beta_1 = \{A_2, B_2, C_3, B_4\}$$

- τ_1 can only experience direct blocking because it is the highest priority task.

Identification of β_2

B_2 →

	C_i	T_i	A	B	C	D	E
τ_1	15	60	3	4	5	–	–
τ_2	30	100	6	11	–	5	–
τ_3	20	150	–	–	(10)	–	8
τ_4	40	200	–	(12)	–	(14)	10

$$\beta_1 = \{A_2, B_2, C_3, B_4\}$$

$$\beta_2 = \{C_3, B_4, D_4\}$$

- τ_2 can be blocked directly by B_4 and D_4, and indirectly by C_3 and B_4.

Identification of β_3

	C_i	T_i	A	B	C	D	E
τ_1	15	60	3	4	5	–	–
τ_2	30	100	6	11	–	5	–
B_3 → τ_3	20	150	–	–	10	–	8
τ_4	40	200	–	(12)	–	(14)	(10)

$\beta_1 = \{A_2, B_2, C_3, B_4\}$

$\beta_2 = \{C_3, B_4, D_4\}$

$\beta_3 = \{B_4, D_4, E_4\}$

- τ_3 can be blocked directly by E_4 and indirectly by B_4 and D_4

43

Identification of β_4

	C_i	T_i	A	B	C	D	E
τ_1	15	60	3	4	5	–	–
τ_2	30	100	6	11	–	5	–
τ_3	20	150	–	–	10	–	8
B_4 → τ_4	40	200	–	12	–	14	10

$\beta_1 = \{A_2, B_2, C_3, B_4\}$

$\beta_2 = \{C_3, B_4, D_4\}$

$\beta_3 = \{B_4, D_4, E_4\}$

$\beta_4 = \{\}$

44

Identification of α_i

	C_i	T_i	A	B	C	D	E	β_i	n_i	m_i	α_i
τ_1	15	60	3	4	5	–	–	$\{A_2, B_2, C_3, B_4\}$	3	3	3
τ_2	30	100	6	11	–	5	–	$\{C_3, B_4, D_4\}$	2	3	2
τ_3	20	150	–	–	10	–	8	$\{B_4, D_4, E_4\}$	1	3	1
τ_4	40	200	–	12	–	14	10	$\{\}$	0	0	0

$\alpha_i = \min(n_i, m_i)$

number of tasks with priority less than τ_i

number of semaphores that can block τ_i (either directly or indirectly).

45

Identification of B_i

	C_i	T_i	A	B	C	D	E	β_i	α_i	B_i
τ_1	15	60	3	4	5	–	–	$\{A_2, B_2, C_3, B_4\}$	3	28
τ_2	30	100	6	11	–	5	–	$\{C_3, B_4, D_4\}$	2	24
τ_3	20	150	–	–	10	–	8	$\{B_4, D_4, E_4\}$	1	14
τ_4	40	200	–	12	–	14	10	$\{\}$	0	0

NOTES

- For τ_1, if we select B_2, we cannot select B_4, because each semaphore can block only once (Lemma 2).
- For τ_2, we cannot select B_4 and D_4, because each task can block only once (Lemma 1).

46

PIP: pro & cons

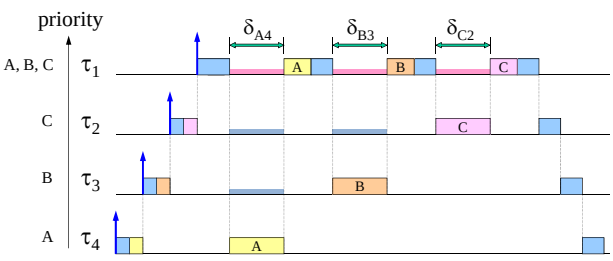
ADVANTAGES:

- It removes the pessimisms of NPP and HLP (a task is blocked only when really needed).
- It is transparent to the programmer.

PROBLEMS:

- More complex to implement (especially to support nested critical sections).
- It is prone to chained blocking.
- It does not avoid deadlocks.

PIP: Chained blocking



NOTE: τ_1 can be blocked at most once for each lower priority task.

48

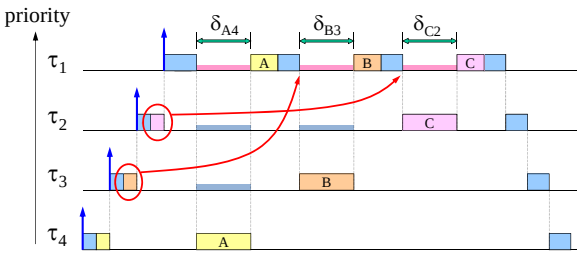
Priority Ceiling Protocol

Access Rule:	A task can access a resource only if it passes the PCP access test.
Progress Rule:	Inside resource R, a task executes with the highest priority of the tasks blocked on R.
Release Rule:	At exit, the dynamic priority of the task is reset to its nominal priority P_i.

NOTE: **PCP** can be viewed as **PIP** + access test

49

Avoiding chained blocking



To avoid multiple blocking of τ_1 we must prevent τ_3 and τ_2 to enter their critical sections (even if they are free), because a low priority task (τ_4) is holding a resource used by τ_1.

50

Resource Ceilings

To keep track of resource usage by high-priority tasks, each resource is assigned a resource ceiling:

$$C(s_k) = max \{ P_i \mid \tau_i \ uses \ s_k \}$$

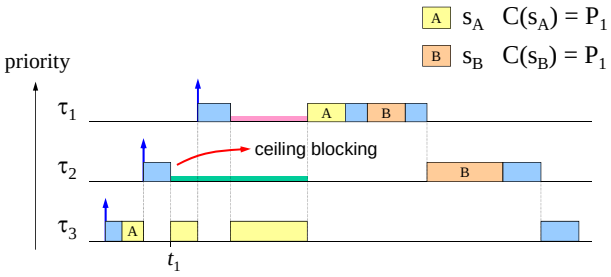
Then a task τ_i can enter a critical section only if its priority is higher than the maximum ceiling of the locked semaphores:

PCP access test

$$P_i > max \{ C(s_k) : s_k \text{ locked by tasks } \neq \tau_i \}$$

51

PCP: example



t_1: τ_2 is blocked by the PCP, since $P_2 < C(s_A)$

52

PCP: properties

Theorem 1

Under PCP, a task can block at most on a single critical section.

Theorem 2

PCP prevents chained blocking.

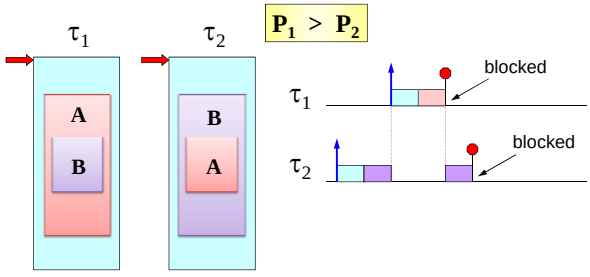
Theorem 3

PCP prevents deadlocks.

53

Typical deadlock

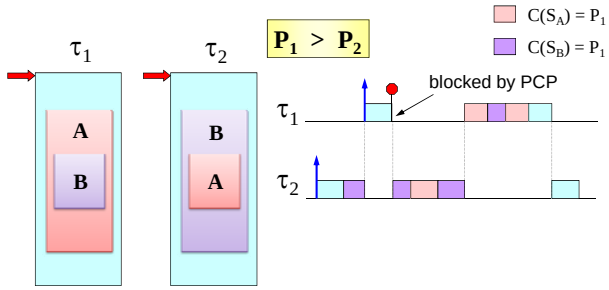
It can only occur with nested critical sections:



54

PCP: deadlock avoidance

It can only occur with nested critical sections:



55

PCP: pro & cons

ADVANTAGES:

- It limits blocking to the length of a single critical section.
- It avoids deadlocks when using nested critical sections.

PROBLEMS:

- It is complex to implement (like PIP).
- It can create unnecessary blocking (it is pessimistic like HLP).
- It is not transparent to the programmer: resource ceilings must be specified in the source code.

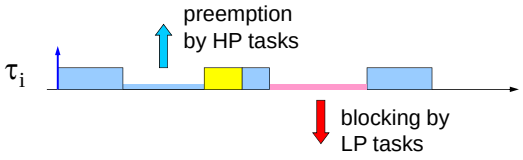
56

Analysis under shared resources

1. Select a scheduling algorithm to manage tasks and a protocol for accessing shared resources.
2. Compute the maximum blocking time B_i for each task.
3. Perform the guarantee test including the blocking terms.

57

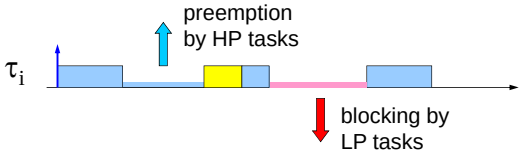
Analysis under RM



$$\forall i \quad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq i\left(2^{1/i} - 1\right)$$

58

Hyperbolic Bound



$$\forall i \quad \prod_{k=1}^{i-1}\left(\frac{C_k}{T_k} + 1\right)\left(\frac{C_i + B_i}{T_i} + 1\right) \leq 2$$

59

Response Time Analysis

$$R_i = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Iterative solution:

$$\begin{cases} R_i^0 = C_i + B_i \\ R_i^{(s)} = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(s-1)}}{T_k} \right\rceil C_k \end{cases}$$

iterate while $R_i^s > R_i^{(s-1)}$

60