Sistemas de Tempo Real:

Alocação de Recursos

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas - DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo

Introdução

- Processador é o recurso mais importante
  - Mais essencial
- Outros recursos também são usados
  - Canais de comunicação
  - Estruturas de dados
  - Arquivos
- Necessidade de alocar e liberar tais recursos
  - Exigem exclusão mútua no acesso

Introdução

- Existem várias questões a serem tratadas:
- Quando e sob quais condições os pedidos de recursos são atendidos ?
- Como são escalonados os jobs que estão esperando por um dado recurso ?
- Quanto tempo uma dada tarefa precisa esperar até conseguir acessar um dado recurso ?
- Como a existência de recursos outros além do processador afeta a escalonabilidade do sistema ?

Modelo de Recursos

- Sistema monoprocessador
- Podem existir diferentes recursos compartilhados
  - Estruturas de dados independentes
  - Periféricos
  - Etc
- Recursos não podem ser preemptados e requerem exclusão mútua
- Existe apenas uma unidade de cada recurso
  - Mais adiante esta suposição será revista
- Não estamos preocupados com recursos que
  - Podem ser acessados simultaneamente (exemplo: tabela read-only)
  - Existe grande número de unidades (exemplo: bytes na memória)

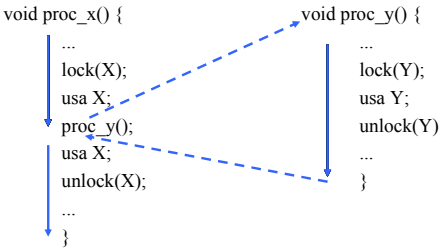
Modelo de Recursos

- Mecanismo de sincronização é usado pelas tarefas para controlar o acesso aos recursos compartilhados
- Um simples MUTEX será usado para controlar o acesso
- LOCK(X) é usado pela tarefa para indicar que ela requer o uso do recurso X
  - Tarefa fica bloqueada até o acesso ao recurso ser autorizado
  - Enquanto bloqueada, ela sai da fila de aptos e não disputa o processador
- UNLOCK(X) é usado pela tarefa para indicar que ela não mais requer acesso ao recurso X
  - Retorno imediato
  - Poderá liberar outra tarefa que estava esperando para usar o recurso X

Modelo de Recursos

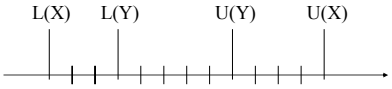
- Seção Crítica
  - Trecho de código onde a tarefa acessa algum recurso compartilhado X
  - Limitado pelas operações LOCK(X) e UNLOCK(X)
- Seções críticas podem ser aninhadas
- É suposto que apenas aninhamentos perfeitos são usados
- Situação mais comum:
  - Chama rotina proc_x que tem LOCK(X) no início e UNLOCK(X) no fim
  - Enquanto da rotina proc_x tem uma chamada para rotina proc_y
  - Rotina proc_y tem LOCK(Y) no início e UNLOCK(Y) no fim

Modelo de Recursos



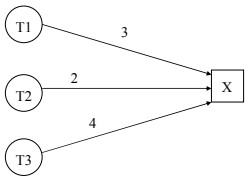
Modelo de Recursos

- Será usada uma notação baseada no livro da Jane Liu (não igual)
- [X,3]
  - Tarefa precisa do recurso X por 3 unidades de tempo
- Recursos aninhados são representados por colchetes aninhados
- [X,3[Y,5]4]
  - Aloca X, usa por 3 unidades de tempo, aloca Y, usa ambos por 5 unidades de tempo, libera Y, usa X por mais 4 unidades de tempo, libera X



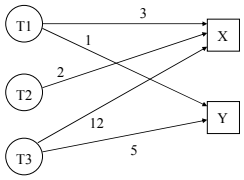
Especificação das Demandas de Recursos

- Grafo dirigido acíclico
- Arco da tarefa para o recurso indica que a tarefa requer o recurso
- Anotação no arco indica a duração da seção crítica
- Exemplo
  - T1:[X,3]
  - T2:[X,2]
  - T3:[X,4]



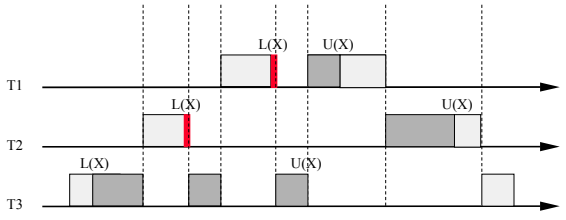
Especificação das Demandas de Recursos

- Exemplo
  - T1:[X,3][Y,1]
  - T2:[X,2]
  - T3:[X,3[Y,5]4]



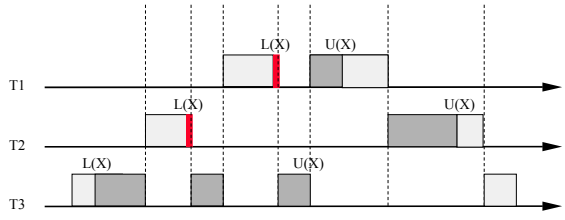
Modelo de Recursos

- Duas tarefas entram em conflito quando precisam usar o mesmo recurso ao mesmo tempo
- LOCK pode fazer a tarefa ficar bloqueada por algum tempo



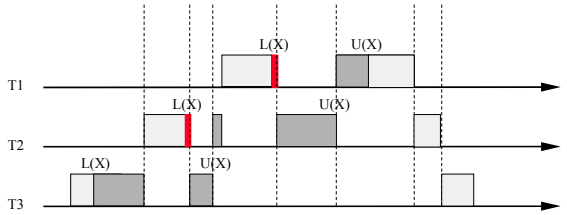
Inversão de Prioridade

- Temos em um dado momento a tarefa T1 esperando pela Tarefa T3
- As prioridades estão invertidas
- O tempo de resposta da tarefa T1 será afetado



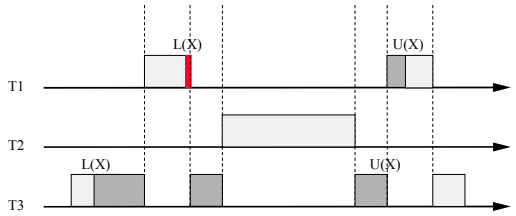
Anomalia da Inversão de Prioridades

- Se T3 reduzir o tamanho da sua seção crítica
  - O tempo de resposta da tarefa T1 AUMENTA



Inversão de Prioridade Descontrolada

- Pode ser ainda pior
- Agora Tarefa T1 espera por T3 e também por T2
- Tarefa T1 nem sequer compartilha qualquer recurso com T2



Possibilidade de Deadlock

- Deadlock:
  - Um conjunto de tarefas estão bloqueadas a espera de evento que somente pode ser causado por tarefas deste conjunto
- Exemplo:

T1	**T2**
Lock(X)	Lock(Y)
Lock(Y)	Lock(X)
...	...
Unlock(Y)	Unlock(X)
Unlock(X)	Unlock(Y)
...	...

- Se deadlock é possível, o tempo de resposta no pior caso é infinito

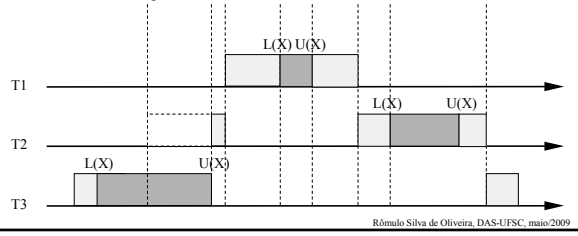
Protocolos de Acesso a Recursos

- Usar protocolos de acesso a recursos específicos
- Podem minimizar a inversão de prioridade
- Podem impedir deadlocks

- Exemplo clássico para impedir deadlocks:
  - Todos os recursos compartilhados são numerados
  - Recursos precisam ser alocados em ordem crescente
  - Tanto T1 como T2
    - Aloca X antes de Y
    - Ou aloca Y antes de X
    - Seja como for, agora é impossível ocorrer um deadlock

(1) Desliga Preempção

- Desliga a preempção antes de acessar qualquer recurso
- Todas as seções críticas executam de forma não preemptiva
- Tempo máximo de bloqueio Bi da tarefa Ti
  - Dado pela duração da maior seção crítica de qualquer tarefa com prioridade mais baixa do que Ti



(1) Desliga Preempção

- Forma mais simples de resolver o problema
- Inversão de prioridade descontrolada não acontece
- Deadlock não acontece
- Qualquer tarefa pode ser bloqueada no máximo uma vez

- Obviamente só funciona em monoprocessador
- Qualquer tarefa pode ser bloqueada por qualquer tarefa de prioridade mais baixa
  - Mesmo que elas não compartilhem recursos entre si

- Solução razoável quando todas as seções críticas forem pequenas
- Corresponde ao “Desabilita Interrupções” de sistemas pequenos

(2) Herança de Prioridade

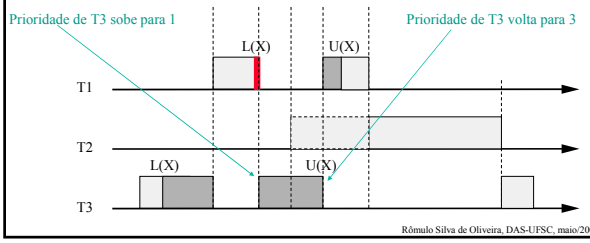
- Cada tarefa possui uma prioridade nominal fixa
  - Dada por RM, DM, etc
- Cada job possui uma prioridade efetiva
  - A prioridade efetiva varia ao longo do tempo
  - A prioridade efetiva é usada para decidir quem executa a seguir
- Inicialmente,
  a prioridade efetiva do job é igual à prioridade nominal da sua tarefa
- A prioridade efetiva pode mudar em decorrência da alocação de recursos pelo job em questão

(2) Herança de Prioridade: Regras

- Regra de alocação: Quando um Job J requer (LOCK) um recurso R no instante t:
  - Se R está livre, R é alocado para J até que J libere o recurso (UNLOCK)
  - Se R está ocupado, J bloqueia
- Regra da Herança de Prioridade:
  - Seja Jx o job que detém o recurso R solicitado por J
  - Job Jx herda a prioridade efetiva de J
  - Job Jx executa com a prioridade efetiva de J até liberar R
  - Quando liberar R, Jx retorna para a prioridade efetiva que tinha em t
  - Herança de prioridade é transitiva

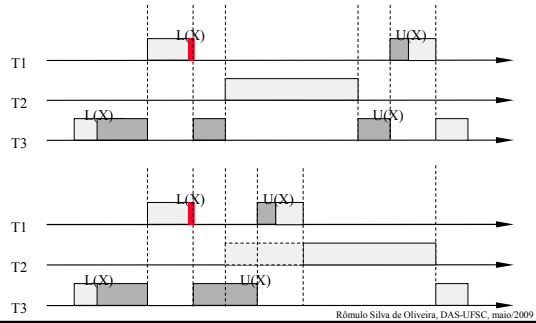
(2) Herança de Prioridade: Exemplo

- Inversão de prioridades descontrolada é evitada
- T3 termina a seção crítica mais cedo
- T1 sofre bloqueio direto de T3
- T2 sofre bloqueio por herança de T3



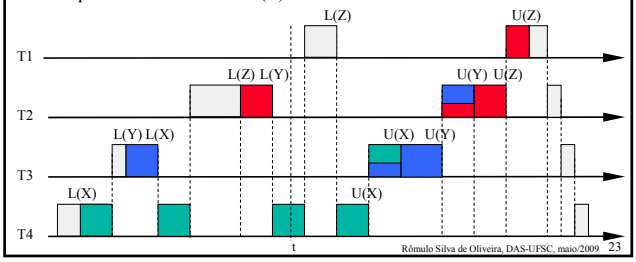
(2) Herança de Prioridade: Exemplo

- Sem herança e com herança de prioridades



(2) Herança de Prioridade: Exemplo

- Bloqueio direto de T2 sobre T1 via Z
- Bloqueio transitivo de T3 sobre T1 via T2(Y)
- Bloqueio transitivo de T4 sobre T1 via T3(X) e T2(Y)
- Suponha T4 executa Lock(Z) em t



(2) Herança de Prioridade

- Impede a inversão de prioridade descontrolada
- Uma tarefa pode ser bloqueada diretamente por outra tarefa de mais baixa prioridade somente uma vez
  - Isto pode acontecer para cada seção crítica usada
- Cada uma das tarefas de mais baixa prioridade é capaz de bloquear a tarefa de mais alta prioridade
  - Por no máximo uma vez
- Tipos de bloqueio
  - Direto
  - Transitivo
  - Por herança de prioridade
- Não impede o deadlock por si só
  - Mecanismo adicional é necessário

(2) Herança de Prioridade

- Não minimiza tempo de bloqueio no pior caso
- Job
  - que requer v recursos
  - e conflita com k jobs de mais baixa prioridade
- Pode ser bloqueado por min(v,k) vezes
  - A duração do bloqueio será aquela do pior caso
  - Precisa considerar o bloqueio transitivo
- Determinação do B para o pior caso pode ficar complexo
  - se os padrões de uso dos recursos forem complexos

(3) Priority Ceiling

- Estende a herança de prioridade para
  - Eliminar a possibilidade de deadlock
  - Reduzir o tempo de bloqueio no pior caso
- Assume que é sabido antes da execução quais recursos cada tarefa usa
- Priority Ceiling = Teto de prioridade
- O teto de prioridade de um recurso Ri corresponde a mais alta prioridade entre todas as tarefas que usam Ri
  - Denotado por Πi
- Uma prioridade imaginária mais baixa que todas as prioridades do sistema será denotada por Ω
- O teto de prioridade do sistema Π corresponde ao maior teto de prioridade entre todos os recursos alocados

(3) Priority Ceiling

- Cada tarefa possui uma prioridade nominal fixa
  - Dada por RM, DM, etc
- Cada job possui uma prioridade efetiva
  - A prioridade efetiva varia ao longo do tempo
  - A prioridade efetiva é usada para decidir quem executa a seguir
- Inicialmente,
  a prioridade efetiva do job é igual à prioridade nominal da sua tarefa
- A prioridade efetiva pode mudar em decorrência da alocação de recursos pelo job em questão
- As prioridades efetivas dos jobs são usadas para fins de escalonamento
  - Executa o job não bloqueado com a prioridade efetiva mais alta

(3) Priority Ceiling: Regra de Alocação

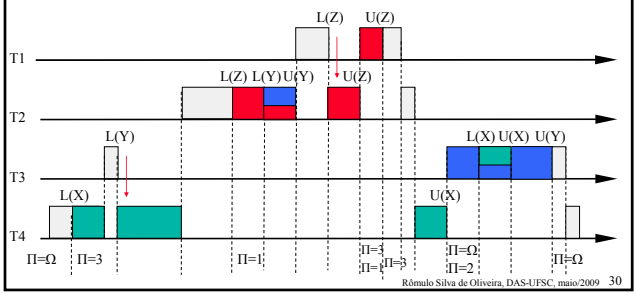
- Regra de Alocação
- Um job J requer um recurso R no instante t
- Se R está alocado para outro job
  - O pedido é negado e J fica bloqueado
- Se R está livre
  - O pedido será aceito se a prioridade efetiva de J for maior que o teto de prioridade do sistema Π
  - O pedido será aceito se a prioridade efetiva de J for igual ao teto de prioridade do sistema Π, mas for J o job que alocou o recurso Ri cujo teto de prioridade Πi define Π
  - Caso contrário, o pedido é negado e J fica bloqueado

(3) Priority Ceiling: Regra de Herança

- Regra de herança de prioridade
- Quando o job J tenta alocar o recurso R e fica bloqueado
  - o job Jx que detém o recurso R herda a prioridade efetiva de J
- Jx mantém esta prioridade até o momento que libera o recurso R
  - Neste instante ele retorna para sua prioridade efetiva anterior
- A herança de prioridade é transitiva

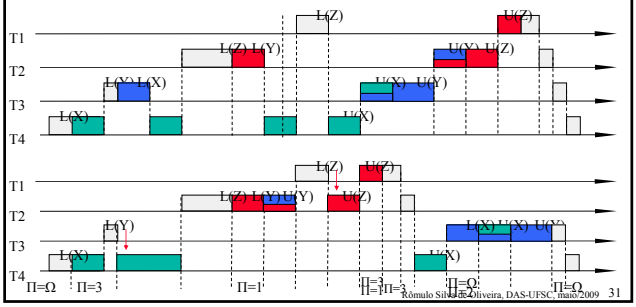
(3) Priority Ceiling: Exemplo

- Tetos de prioridade:
  - $\Pi(X)=3$, $\Pi(Y)=2$, $\Pi(Z)=1$



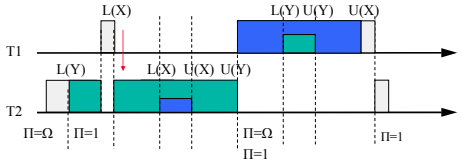
(3) Priority Ceiling: Exemplo

- Herança de prioridade versus Teto de prioridade



(3) Priority Ceiling

- Impossível ocorrer deadlock
  - T1:[X,1[Y,1]]
  - T2:[Y,1[X,1]]
  - Π(X)=1, Π(Y)=1



(3) Priority Ceiling

- Herança de Prioridade
  - Guloso
  - Se recurso estiver livre, o mesmo é alocado
- Teto de Prioridade
  - Conservador
  - Mesmo um recurso livre pode não ser alocado
  - Isto é feito para prevenir um comportamento pior mais adiante
- Semelhante aos protocolos para evitar deadlock
  - Algoritmo do banqueiro

(3) Priority Ceiling: Tipos de Bloqueios

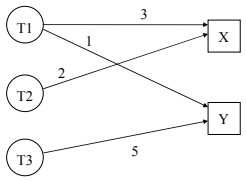
- Bloqueio pode ocorrer de três formas
- Bloqueio direto
  - Recurso está ocupado
- Bloqueio por herança de prioridade
  - Bloqueado por tarefa que herdou prioridade mais alta
- Bloqueio por teto
  - Recurso está livre, mas teto do sistema é mais alto
- Determinação do B é simplificada pois:
  - Um job pode ser bloqueado por no máximo a duração de uma seção crítica

(3) Priority Ceiling

- Uma tarefa Ti pode ser bloqueada apenas uma vez por uma mesma tarefa de prioridade mais baixa Tx
  - Ao liberar o recurso a primeira vez, Tx não executa mais e não aloca novamente
- Não é possível uma tarefa Ti ser bloqueada por Tx e Ty, se Tx e Ty tiverem prioridade mais baixa do que Ti
  - Após o primeiro bloqueio, a regra do teto impede um segundo bloqueio
- Uma tarefa Ti pode ser bloqueada no máximo pela duração de uma única seção crítica
  - Não importa quantas tarefas compartilham recursos com a tarefa Ti
  - Vale a seção crítica mais externa, quando aninhadas

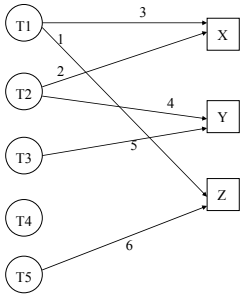
(3) Priority Ceiling

- T1
  - Bloqueio direto de T2 por 2 ut
  - Bloqueio direto de T3 por 5 ut
- T2
  - Bloqueio por teto de T3 por 5 ut
  - Bloqueio por herança de T3 por 5 ut
  - Somente um dos dois é possível
- T3
  - Não sofre bloqueio por definição



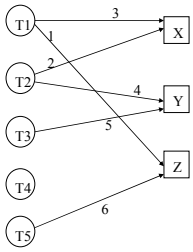
(3) Priority Ceiling

- Exemplo mais complexo



(3) Priority Ceiling: Bloqueio Direto

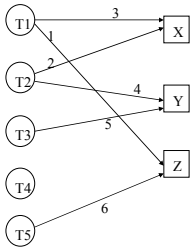
- Para cada tarefa
  - Determina as possibilidades de bloqueio direto
  - Lido direto do grafo de recursos



	por T1	por T2	por T3	por T4	por T5
T1	---	2 (X)	-	-	6 (Z)
T2	---	---	5 (Y)	-	-
T3	---	---	---	-	-
T4	---	---	---	---	-
T5	---	---	---	---	---

(3) Priority Ceiling: Bloqueio por Herança

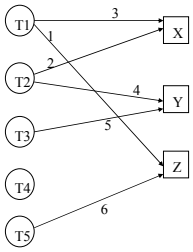
- Para cada tarefa
  - Determina as possibilidades de bloqueio por herança
  - Máximo da mesma coluna na tabela anterior, em linhas acima



	por T1	por T2	por T3	por T4	por T5
T1	---	-	-	-	-
T2	---	---	-	-	6 (Z)
T3	---	---	---	-	6 (Z)
T4	---	---	---	---	6 (Z)
T5	---	---	---	---	---

(3) Priority Ceiling: Bloqueio por Teto

- Para cada tarefa
  - Determina as possibilidades de bloqueio por teto
  - Mesma tabela anterior, exceto para tarefas que não usam recursos



	por T1	por T2	por T3	por T4	por T5
T1	---	-	-	-	-
T2	---	---	-	-	6 (Z)
T3	---	---	---	-	6 (Z)
T4	---	---	---	---	-
T5	---	---	---	---	---

(3) Priority Ceiling: Cálculo do B

- Tempo de bloqueio máximo de cada tarefa é o máximo da linha
- B1=6, B2=6, B3=6, B4=6, B5=0

	por T1	por T2	por T3	por T4	por T5
T1	---	2	-	-	6
T2	---	---	5	-	-
T3	---	---	---	-	-
T4	---	---	---	---	-
T5	---	---	---	---	---

	por T1	por T2	por T3	por T4	por T5
T1	---	-	-	-	-
T2	---	---	-	-	6
T3	---	---	---	-	6
T4	---	---	---	---	6
T5	---	---	---	---	---

	por T1	por T2	por T3	por T4	por T5
T1	---	-	-	-	-
T2	---	---	-	-	6
T3	---	---	---	-	6
T4	---	---	---	---	---
T5	---	---	---	---	---

(3) Priority Ceiling

- No pior caso,
  - Priority Ceiling não bloqueia mais do que Herança de Prioridade
- Bloqueio gera mais chaveamento de contexto
  - Precisa incluir no tempo de execução das tarefas
  - Tarefa que não usa recurso acrescenta 2 chaveamentos de contexto
    - Quando inicia e quando termina
  - Tarefa que usa recurso acrescenta 4 chaveamentos de contexto
    - Quando inicia, quando fica bloqueada, quando é liberada, quando termina
  - Cada tarefa tem o seu WCET acrescido de seus chaveamentos de contexto
- Regras são complexas
- É necessário saber quem bloqueia quem, de forma transitiva

(4) Immediate Priority Ceiling

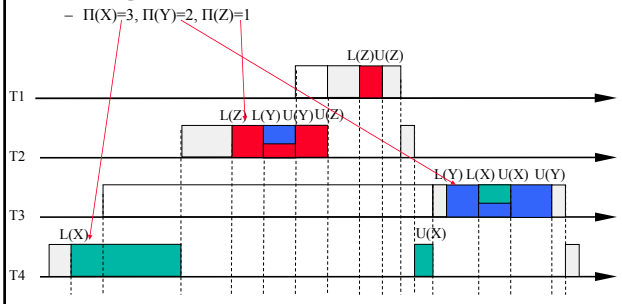
- Cada tarefa possui uma prioridade nominal
  - Atribuída por RM, DM, etc
- Cada recurso tem uma prioridade teto
  - Prioridade mais alta entre as tarefas que usam o recurso
- Cada tarefa tem uma prioridade efetiva
  - Mais alta entre
  - sua própria prioridade nominal
  - e as prioridades teto de quaisquer recursos que a tarefa tenha alocado
- Executa a tarefa apta com a mais alta prioridade efetiva
  - Para a mesma prioridade é usado FIFO

(4) Immediate Priority Ceiling

- Ao alocar um recurso,
  - Tarefa recebe a prioridade teto daquele recurso
  - Se for mais alta do que sua prioridade efetiva corrente
- Como conseqüência, uma tarefa irá sofrer apenas um bloqueio
  - No início de sua execução
- Uma vez que a tarefa realmente inicia sua execução
  - todos os recursos que ela necessita estarão livres
- Se eles não estivessem,
  - então alguma outra tarefa teria uma prioridade igual ou mais alta
  - e a execução da tarefa em questão seria postergada
- O bloqueio máximo sofrido por qualquer job é o mesmo de quando Priority Ceiling é usado

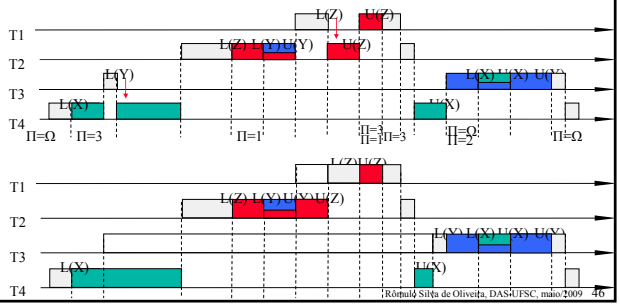
(4) Immediate Priority Ceiling: Exemplo

- Tetos de prioridade:
  - Π(X)=3, Π(Y)=2, Π(Z)=1



(4) Immediate Priority Ceiling: Exemplo

- Teto de Prioridade versos Teto de Prioridade Imediado



(4) Immediate Priority Ceiling

- Comportamento de pior caso é o mesmo
  - Priority Ceiling
  - Immediate Priority Ceiling
- Immediate Priority Ceiling
  - É mais fácil de implementar
  - Quem bloqueia quem não precisa ser monitorado
  - Gera menos chaveamentos de contexto pois não ocorre bloqueio após o início da execução
  - Requer mais movimentos de prioridade pois isto acontece a cada uso de recurso
- Immediate Priority Ceiling é chamado de
  - Priority Protect Protocol em POSIX
  - Priority Ceiling Emulation em Real-Time Java

Resumo

- Acesso a recursos compartilhados
  - Gera bloqueio (inversão de prioridades)
  - Pode gerar deadlock
- Mutex simples é ineficaz
- Necessário usar protocolos específicos para tempo real
- Desliga Preempção
- Herança de Prioridade
- Priority Ceiling
- Immediate Priority Ceiling