Sistemas Operacionais

Aula 9 – Deadlocks (2)

Cleyton Slaviero

cslaviero@gmail.com

Deadlocks

- **Recuperação** de deadlock
  - Por meio de **preempção**
    - Possibilidade de retirar temporariamente um recurso de seu atual dono (processo) e entregá-lo a outro processo
    - Depende muito da natureza do recurso
    - Frequentemente difícil ou impossível
    - Exemplo: impressora?

Universidade Federal de Mato Grosso
Campus Rondonópolis

Deadlocks

- Recuperação de Deadlocks
  - Por meio de **rollback**:
    - O estado de cada processo (e recurso por ele usado) é periodicamente armazenado em um arquivo de verificação (checkpoint file);
    - Novas checagens são armazenadas em novos arquivos, à medida que o processo executa
    - Quando ocorre um deadlock, o processo que detém o recurso é voltado a um ponto antes de adquirir esse recurso (via o checkpoint file apropriado)
      - O trabalho feito após esse ponto é perdido
      - O processo é retornado a um momento em que não possuía o recurso, que será então dado a outro

Deadlocks

- Recuperação de Deadlocks:
  - Por meio de eliminação de processos (kill):
    - Um ou mais processos que estão no ciclo com deadlock são interrompidos.
      - Talvez isso evite o deadlock. Senão, continue eliminando até quebrar o ciclo
    - Melhor solução para processos que não causam algum efeito negativo ao sistema;
      - Ex1.: compilação – sem problemas;
        Ex2.: atualização de um base de dados – problemas;

Evitando Deadlocks

- Até então…
  - Quando um processo quer recursos, pede todos de uma vez
    - Nem sempre é assim!
  - Ha um algoritmo que pode evitar o deadlock fazendo sempre a escolha certa?
    - SIM! Mas
    - Precisamos saber informações de antemão

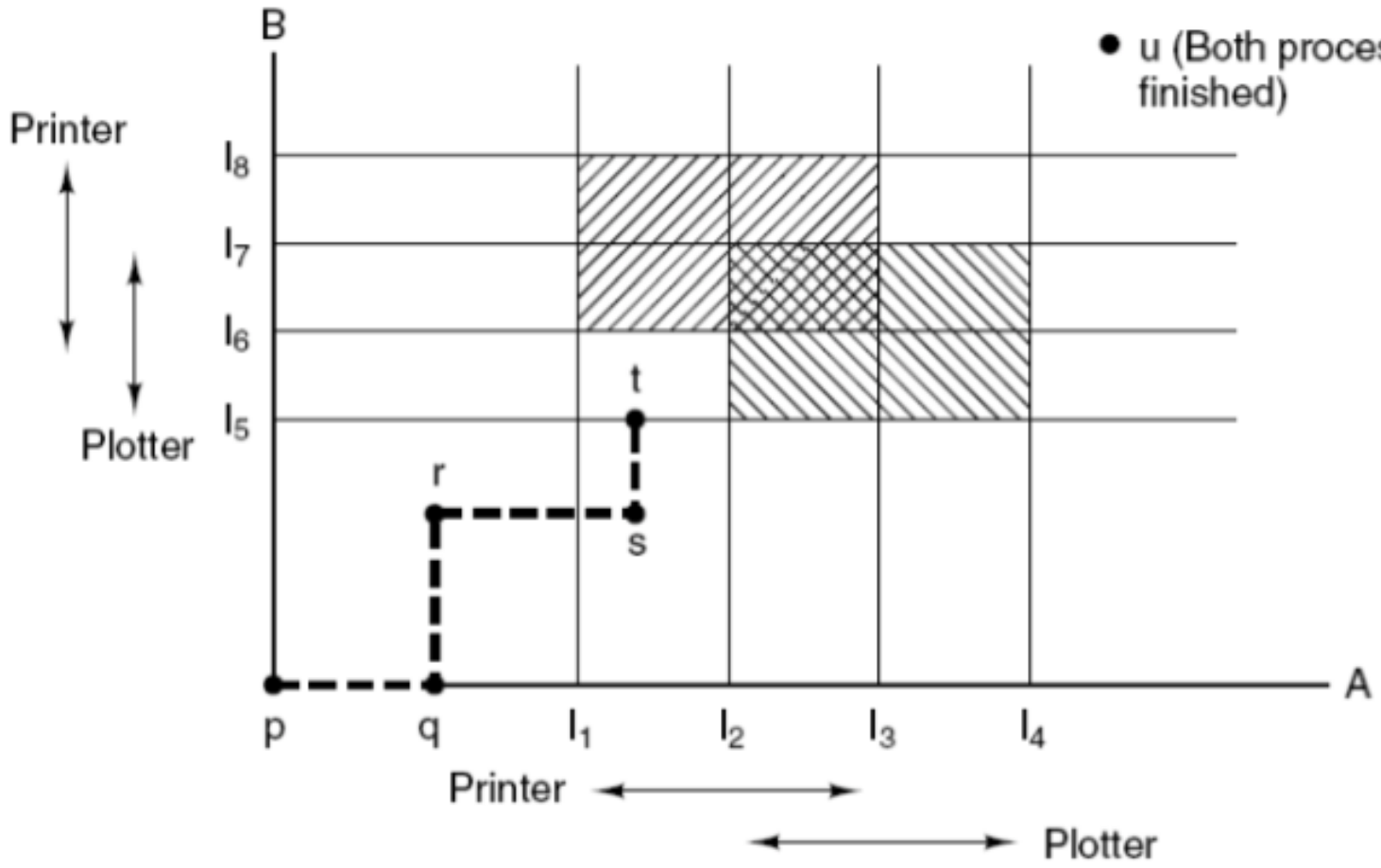
Evitando Deadlocks

- Algoritmos
  - Banqueiro para um único tipo de recurso;
  - Banqueiro para vários tipos de recursos;
- Usam a noção de Estados Seguros e Inseguros;

Universidade Federal
de Mato Grosso
Campus Rondonópolis

Evitando deadlocks

- Trajetória de recursos
  - Eixo horizontal: instruções de A
  - Eixo vertical: instruções de B
  - Cada um necessita de uma impressora e plotter em tempos distintos
  - Regiões cinzas são zonas não seguras
  - Em t, B quer um recurso **.O que fazer?**



Universidade Federal de Mato Grosso
Campus Rondonópolis

Evitando Deadlocks

- Estados seguros: não provocam deadlocks e há uma maneira de atender a todas as requisições pendentes finalizando normalmente todos os processos;
  - A partir de um estado seguro, existe a garantia de que os processos terminarão;
- Estados inseguros: podem provocar deadlocks, mas não necessariamente provocam;
  - A partir de um estado inseguro, não é possível garantir que os processos terminarão corretamente;

Evitando Deadlocks

- Usa as mesmas estruturas da detecção de impasses com vários recursos
  - Um estado atual consiste das estruturas E, A, C e R
    - Estado seguro
      - Há alguma ordem de alocação de recursos na qual todos os processos irão terminar, mesmo caso todos peçam seu número máximo de recursos imediatamente

Evitando Deadlocks

- Estado seguro
  - Ex: tabela para um único recurso

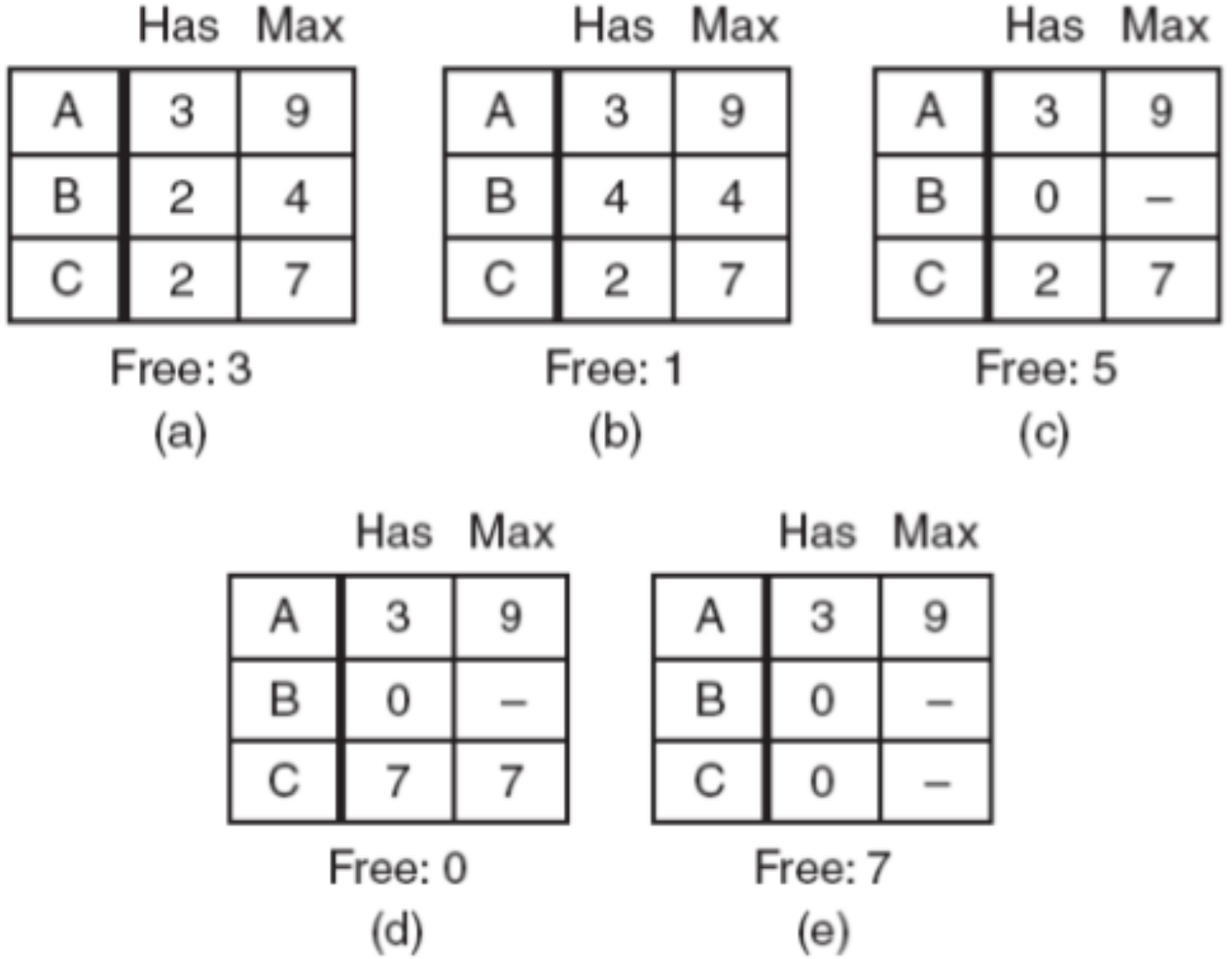
E = 10

	C	R
A	3	9
B	2	4
C	2	7

A 3

Evitando Deadlocks

- Estado seguro
  - Há uma sequência que permite que todos os processos terminem



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Evitando Deadlocks

- Evitar dinamicamente o problema
  - Estado seguro
    - Suponha agora a configuração abaixo
      - Essa sequência é segura?

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Evitando Deadlocks

- Evitar dinamicamente o problema
  - Estado seguro
    - Não há seqüência capaz de garantir que os programas terminem
    - A decisão que moveu de (a) para (b) levou de um estado seguro a um inseguro
      - Não deveríamos tê-la tomado

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Evitando Deadlocks

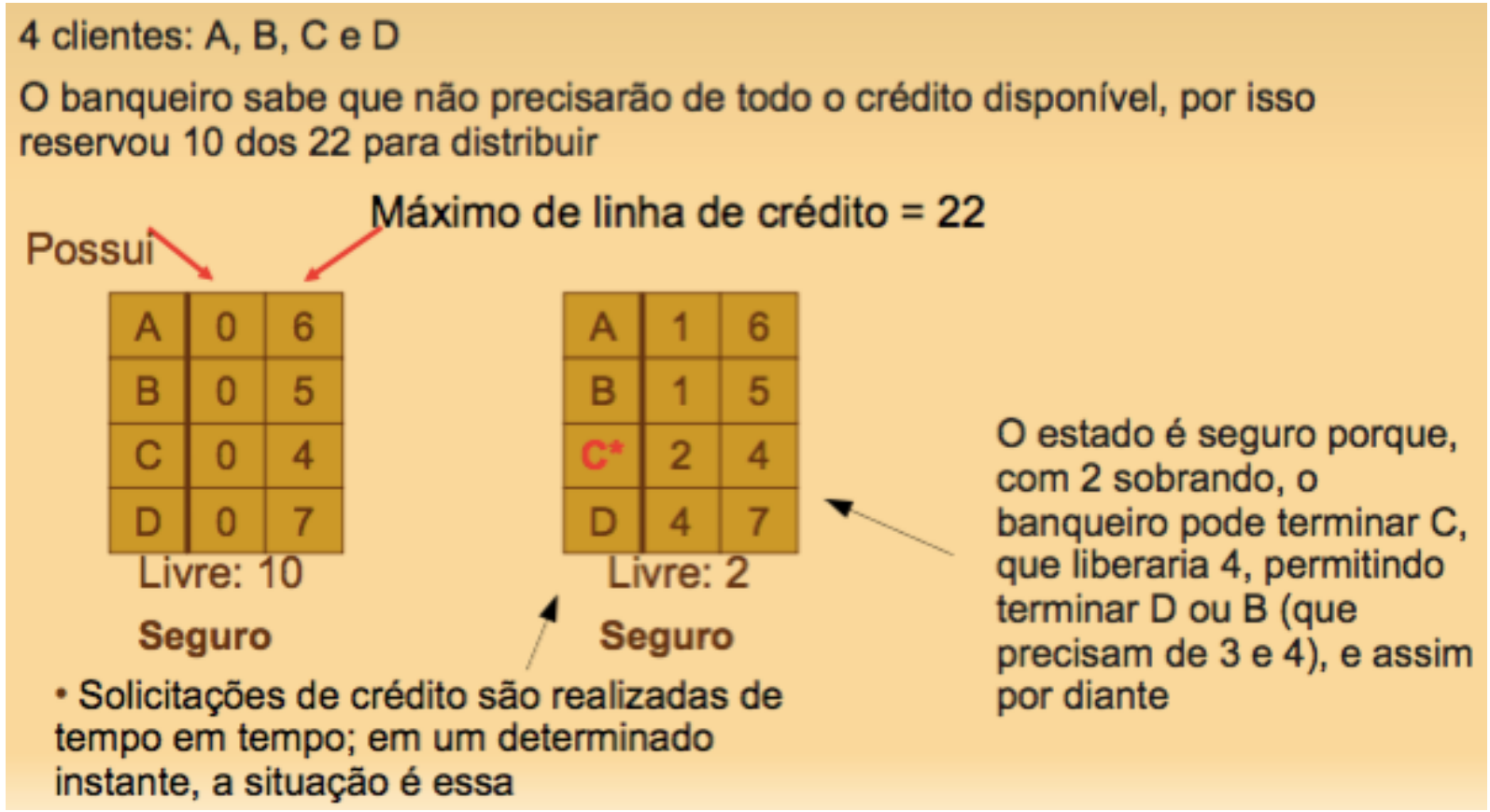
- Algoritmo do banqueiro
  - Idealizado por Dijkstra (1965);
  - Algoritmo de escalonamento capaz de evitar deadlock
  - Premissas adotadas por um ”banqueiro” (SO) para garantir ou não crédito (recursos) para seus clientes (processos);
    - Verifica se atender um pedido leva a um estado inseguro.
    - Se levar, o pedido é negado; senão, é executado
  - A intuição é garantir que sempre haverá possiblidade de alocar recursos solicitados para alguém

Evitando Deadlocks

- Algoritmo do Banqueiro para um único tipo de recurso:
  - Considera cada pedido à medida em que ocorre, e vê se atendê-lo leva a um estado seguro
  - Para ver se o estado é seguro, o banqueiro verifica se tem recursos suficientes para satisfazer algum cliente
  - Se tiver, assume que os ”empréstimos” serão pagos, e o cliente mais próximo do limite é checado
  - Sempre tenta emprestar o máximo a um único por vez

Evitando Deadlocks

- Algoritmo do banqueiro para um único tipo de recurso



Evitando Deadlocks

- Algoritmo do banqueiro para um único tipo de recurso

Evitando Deadlocks

- Algoritmo do Banqueiro para **vários** tipos de recursos:
  - Mesma idéia, mas duas matrizes são utilizadas;

Process	Tape drives	Plotters	Printers	Blu-rays
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Matriz **C**

Process	Tape drives	Plotters	Printers	Blu-rays
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still assigned

Matriz **R**

E = (6342)
P = (5322)
A = (1020)

Evitando Deadlocks

- Algoritmo do Banqueiro para vários tipos de recursos:
  - Busque em R uma linha que possa ser satisfeita pelos valores em A (deve possuir valores todos menores ou iguais aos de A)
    - Se não existir, pode haver deadlock
  - Assuma que o processo na linha escolhida pede todos os recursos de que precisa e termina.
    - Marque este processo como terminado e adicione todos seus recursos a A
  - Repita os passos acima até que todos os processos sejam marcados como terminados (caso em que o estado inicial era seguro) ou reste processo que não possa ser satisfeito (deadlock)

O que aconteceria se atendêssemos uma requisição de B?

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

C = Recursos Alocados

Alocados → P = (5 3 3 2);
Disponíveis → A = (1 0 1 0);

A	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

R = Recursos ainda necessários

- Podem ser atendidos: D, A ou E, C;

Ao atender E, teríamos um deadlock

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	1	0

C = Recursos Alocados

Alocados → P = (5 3 4 2);
Disponíveis → A = (1 0 0 0);

A	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	0	0

R = Recursos ainda necessários

- *Solução*: Adiar a requisição de E por alguns instantes;

Evitando Deadlocks

- Desvantagens
  - Pouco utilizado, pois é raramente se sabe quais recursos serão necessários;
  - Escalonamento cuidadoso é caro para o sistema;
  - O número de processos é dinâmico e pode variar constantemente, tornando o algoritmo custoso;
  - Recursos podem desaparecer/quebrar
- Vantagem
  - Na teoria o algoritmo é ótimo;
- Alguns sistemas usam heurísticas próximas as do banqueiro
  - Exemplo: redes

Prevenindo Deadlocks

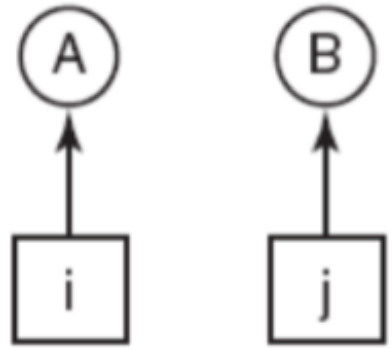
- Atacar uma das quatro condições:
- Exclusão mútua
  - Alocar todos os recursos usando um buffer/spool
    - Processos podem gerar output ao mesmo tempo
    - Deadlock pode ocorrer!!
    - Ainda assim é uma ideia interessante limitar ao máximo os processos que podem de fato solicitar o recurso
- Uso e espera
  - Processos requisitam todos os recursos de que precisam antes da execução
    - Difícil saber!
    - Recursos podem não ser usados de forma ótima
  - Podemos tentar também fazer com que o processo libere todos os recursos e pegue quando precisar, caso solicite um novo

Prevenindo deadlocks

- Não preempção
  - Retirar recursos dos processos
    - Pode ser ruim dependendo do tipo de recurso
    - Praticamente não implementável
- Espera circular
  - Ordenar numericamente os recursos, e realizar solicitações em ordem numérica
    - Não há ciclos
  - Permitir que o processo use um recurso por vez

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD-ROM drive

(a)



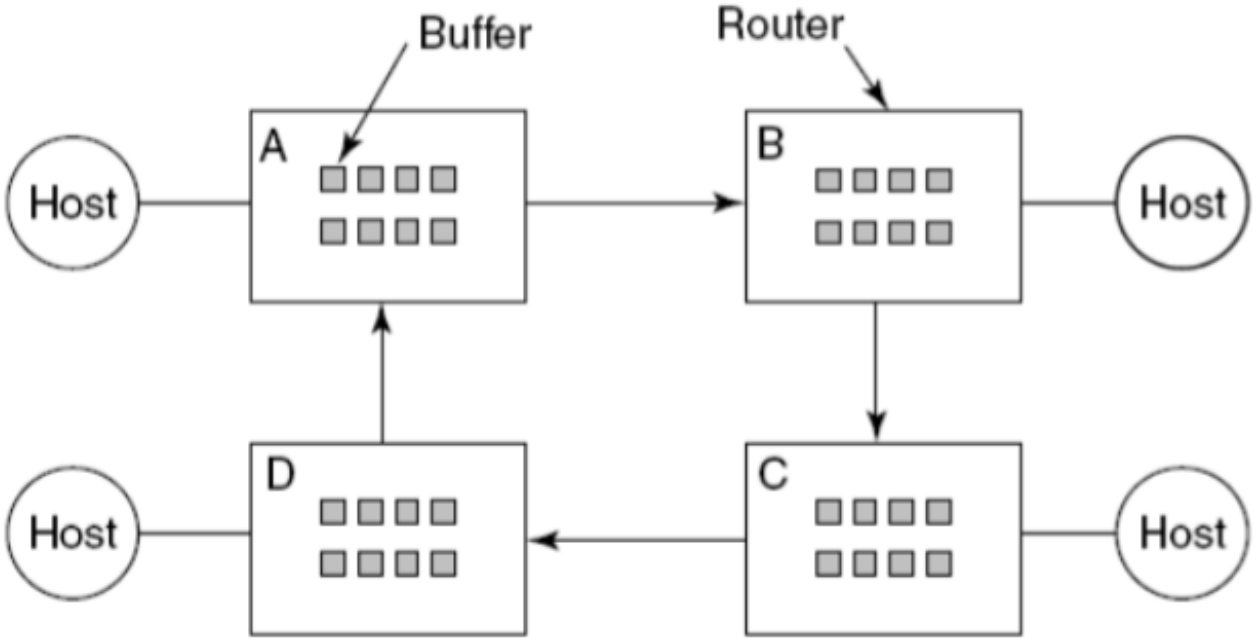
(b)

Outras questões

- Bloqueio em duas fases
  - Num banco de dados
    - Tenta-se travar todos os registros que precisa
    - Em seguida, faz os updates e libera as travas
    - Se na primeira fase alguém estiver bloqueado, tenta com todos de novo depois
  - Não é aplicável em geral
    - Só funciona caso a parada de processos seja possível e bem definida

Outras questões

- Deadlock de comunicação
  - Pode ocorrer quando dois processos esperam a resposta/comando um do outro
- Não dá pra evitar simplesmente numerando processos ou bloqueando
  - Solução: timeouts!
- Deadlocks de recursos também podem ocorrem em ambientes de rede!



Outras questões

- Livelock
  - Exemplo:
    - Duas pessoas andando uma de encontro para a outra na rua
    - Se as duas decidem ir para o mesmo lado...

- Um processo pode tentar desistir de locks que já adquiriu
- Se outro processo tem a mesma ideia…
  - Livelock!

Outras questões

- Inanição (Starvation)
  - Todos os processos devem conseguir utilizar os recursos que precisam, sem ter que esperar indefinidamente;
    - Solução? Colocar os processos em uma fila
- Alguns não fazem a distinção entre starvation e deadlock