# SOP - TADS
## Impasses - Deadlock

Prof. Ricardo José Pfitscher
dcc2rjp@joinville.udesc.br
Material cedido por:
Prof. Rafael Rodrigues Obelheiro
Prof. Maurício Aronne Pillon

# Cronograma

- Noções de deadlock
- Modelagem de deadlock
- Tratamento de deadlock

# Noções de Deadlock

- Conceito informal de deadlock
  - Em sistemas multiprogramados os processos competem por recursos do sistema
    - Memória, CPU, dispositivos de E/S, tabelas do SO...
  - Em determinadas situações os recursos que são alocados a um processo não podem ser retirados a força do mesmo
    - Gravador de CD, Impressora, etc...
  - Se P1 detém o recurso X e quer Y e P2 detém Y quer X, temos um impasse
    - P1 e P2 bloqueiam e nenhum dos dois pode prosseguir
  - Podemos fazer uma analogia com o dia a dia

# Noções de Deadlock

- Recursos
  - Deadlocks ocorrem quando se garante acesso exclusivo a recursos
    - Podemos caracterizar dois tipos de recursos:
    - Preemptíveis: Podem ser retirados de um processo sem problemas
      - CPU, Memória
    - Não-Preemptíveis: A retirada deste recurso pode gerar falha ao processo
      - Impressora, gravador de CD, Scanner...
  - Quais destes recursos podem ocasionar deadlocks??
    - Somente recursos não preemptíveis

# Noções de Deadlock

- Utilização de recursos
  - Para utilizar um recurso, o processo tipicamente:
    1. Solicita o recurso
    2. Usa o recurso
    3. Libera o recurso
  - Quando uma solicitação falha, o processo espera até que o recurso esteja disponível
    - Solicitação bloqueia
    - Solicitação retorna erro, o processo fica em loop
  - Se um processo não libera um recurso após usa-lo a probabilidade de ocorrer deadlock aumenta

# Noções de Deadlock

- Definição formal
  - *"Um processo está em situação de deadlock se todo processo pertencente ao conjunto estiver esperando por um evento que somente um processo desse mesmo conjunto poderá provocar"*
- Normalmente um evento é a liberação de um recurso atualmente retido
- Nenhum dos processos pode
  - Executar
  - Liberar recursos
  - Ser acordado

# Noções de Deadlock

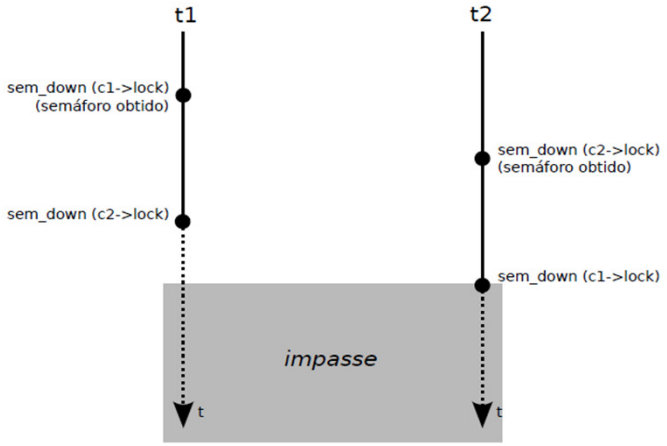
- Exemplo

```
typedef struct conta_t
{
   int saldo ;                    // saldo atual da conta
   sem_t lock ;                   // semáforo associado à conta
   ...                            // outras informações da conta
} conta_t ;

void transferir (conta_t* contaDeb, conta_t* contaCred, int valor)
{
   sem_down (contaDeb->lock) ;    // obtém acesso a contaDeb
   sem_down (contaCred->lock) ;   // obtém acesso a contCred

   if (contaDeb->saldo >= valor)
   {
      contaDeb->saldo -= valor ;  // debita valor de contaDeb
      contaCred->saldo += valor ; // credita valor em contaCred
   }
   sem_up (contaDeb->lock) ;      // libera acesso a contaDeb
   sem_up (contaCred->lock) ;     // libera acesso a contaCred
}
```

- Caso dois clientes (t1 e t2) queiram fazer transferências entre suas contas (c1 e c2),(t1: c1→ c2,t2:c2→c1) o que acontece?

# Noções de Deadlock

- Exemplo

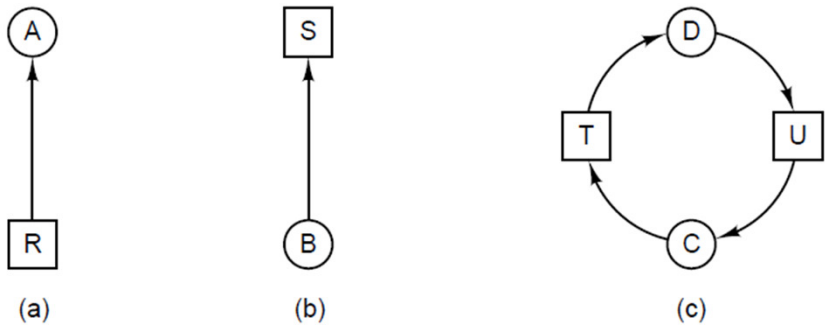


# Modelagem de Deadlock

- Condições para ocorrência de deadlock
  1. Exclusão Mútua
     - Todo recurso está ou associado a um processo ou disponível
  2. Posse e espera
     - Processos que retêm recursos podem solicitar novos recursos
  3. Não preempção
     - Recursos concedidos previamente não podem ser tomados à força
  4. Espera circular
     - Deve haver uma cadeia circular de dois ou mais processos
     - Cada um está à espera de recurso cedido pelo membro seguinte dessa cadeia

# Modelagem de Deadlock

- Grafo dirigido de alocação de recursos




a) A alocou R

b) B solicitou S (está bloqueado, esperando a alocação)

c) C e D em deadlock sobre T e U

# Tratamento de Deadlocks

- Estratégias para tratar deadlocks
  1. Ignorar completamente o problema
  2. Detecção e recuperação
  3. Evitar dinamicamente a ocorrência
     - Alocar os recursos com cuidado
  4. Prevenção
     - Negar uma das quatro condições necessárias

# Tratamento de Deadlocks

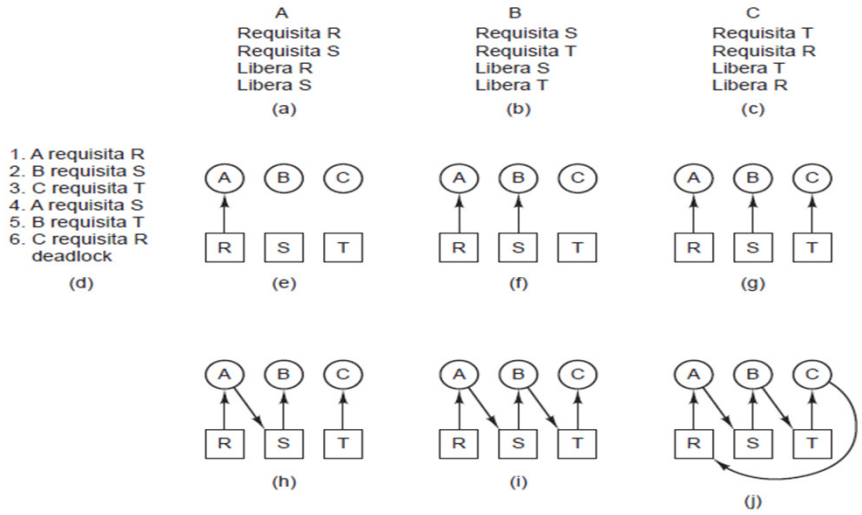
- Como ocorre um deadlock
- Considere a tabela:

| | **Processos** | | |
|---|---|---|---|
| | A | B | C |
| Alocação de Recursos | Requisita R | Requisita S | Requisita T |
| | Requisita S | Requisita T | Requisita R |
| | Libera R | Libera S | Libera T |
| | Libera S | Libera T | Libera R |

- Escalonamento: 1 procedimento por processo
  - Ordem de execução A-B-C
- Como fica o grafo de alocação de recursos?

# Tratamento de Deadlocks

- Como ocorre um deadlock




# Tratamento de Deadlocks

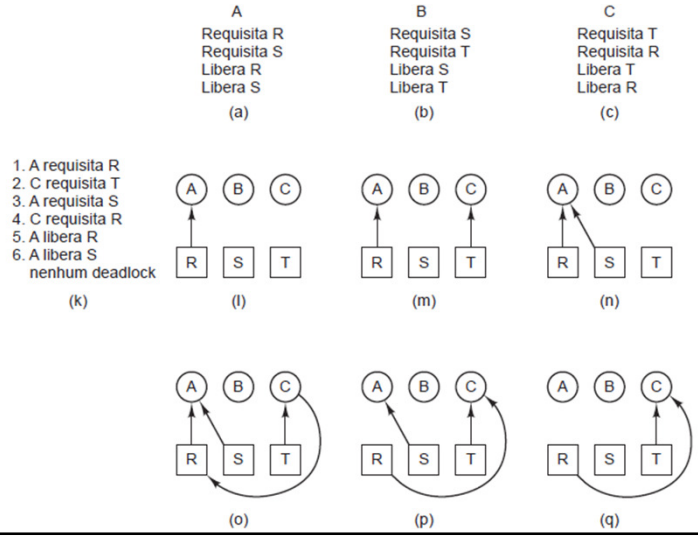
- Como evitar o deadlock?

| | Processos | | |
|---|---|---|---|
| | A | B | C |
| Alocação de Recursos | Requisita R | Requisita S | Requisita T |
| | Requisita S | Requisita T | Requisita R |
| | Libera R | Libera S | Libera T |
| | Libera S | Libera T | Libera R |

- Ordem de execução?

# Tratamento de Deadlocks

- Evitando um deadlock




# Tratamento de Deadlocks

- Algoritmo do Avestruz
  - "Enterre a cabeça na areia e finja que o problema não existe"

  - Ignora a existência de deadlocks, se algum ocorrer, o usuário que resolva
  - Baseia-se no princípios que os deadlocks são infrequentes na prática
    - É mais provável que o sistema trave antes por outro motivo
    - Evita o custo associado aos mecanismos de tratamento de deadlocks
      - Desempenho e conveniência
  - Estratégia usada no UNIX e no Windows

# Tratamento de Deadlocks

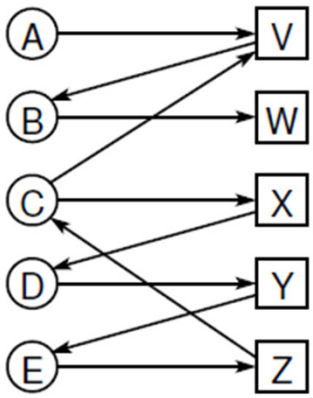
- Detecção de deadlocks
  - Um algoritmo simples para detectar a ocorrência de deadlock baseia-se no grafo de alocação de recursos
    - Monitorar a alocação de recursos e disparar um procedimento de recuperação caso **um ciclo** seja encontrado no grafo

# Tratamento de Deadlocks

- Algoritmo para detecção de ciclos
  - Para todos os nós do grafo executar *(L* é uma lista de nós)*:*
    1. *L=[ ] ,* todos os nós do grafo são desmarcados
    2. Insira o nó atual em *L* e verifique se aparece duas vezes, se sim, o grafo tem um ciclo, e o algoritmo termina
    3. Ache um arco desmarcado **saindo** do nó corrente
       i. Se houver, marque o arco e visite o nó, voltando ao passo 2
       ii. Se não houver, retire o nó corrente de *L* e retorne ao nó anterior, voltando ao passo 3
           i. Se for o primeiro, o algoritmo termina, e não há ciclos.

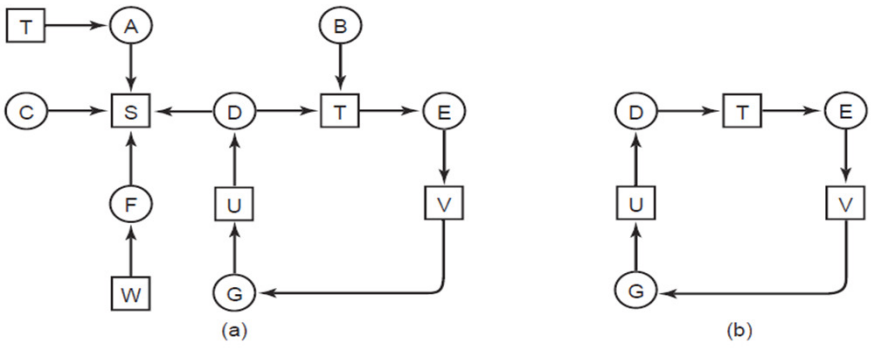
# Tratamento de Deadlocks

- Executando o algoritmo (1/2)
  - Execute para o grafo abaixo




# Tratamento de Deadlocks

- Executando o algoritmo (2/2)




iniciando em B: L = [B *T E V G U D T*]

# Tratamento de Deadlocks

- Métodos de Recuperação
  - Preempção
    - Retira o recurso de algum outro processo
      - Depende da natureza do recurso
  - Reversão de Estado
    - Armazena periodicamente o estado do processo (checkpointing)
    - Reinicia um processo do estado salvo (chekpoint) em caso de deadlock
      - Tudo que foi depois do chekpoint é perdido e precisa ser refeito
  - Eliminação de processos
    - Escolhe um processo para ser eliminado, quebrando o ciclo
      - O processo escolhido deve deter recursos que estão causando o deadlock
    - Preferencialmente se escolhe um processo que possa ser reiniciado sem grandes consequências

# Exercícios

- Sugerem-se os seguintes exercícios do Cap 6 de Tanenbaum (3ª edição):
  - 2,3,5,7,17
  - Podem fazer todos ser preferirem
- Cap 3 de Tanembaum (2ª edição):
  - 2,3,4,5,6,7,15

## Aviso

- O horário da monitoria mudou para melhor atendê-los
- Ficou assim:
  - Terça-feira: 17:00-18:40
  - **Quarta-feira**: 18:00-19:40
  - Sexta-feira: 13:30-15:10

## Bibliografia


- Andrew S. Tanenbaum. Sistemas Operacionais Modernos, 3a Edição. Capítulo 6.Pearson Prentice-Hall, 2009.
- Carlos A. Maziero Cap 4:
- http://dainf.ct.utfpr.edu.br/~maziero/doku.php/so:livro_de_sistemas_operacionais