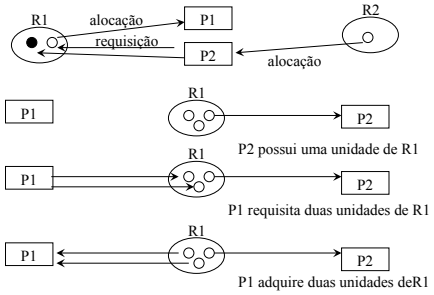
ALGORITMOS DISTRIBUÍDOS
Deadlock

Algoritmos Distribuídos *(deadlock)*

- Um *deadlock* é causado pela situação onde um conjunto de processos está bloqueado permanentemente, i.e., não conseguem prosseguir a execução, esperando um evento que somente outro processo do conjunto pode causar.
- Situação de *deadlock*:
  alocação de recursos formando ciclo
- Outra situação de *deadlock:*
  - vários processos tentam alocar recursos para realizar suas tarefas, alocando o total de recursos de uma máquina.
  - não podem acabar as tarefas por não terem recursos suficientes, não liberam recursos por não terem acabado as tarefas.
  - serializar a execução dos processos, não permitindo concorrência de processos que utilizem na soma mais que a quantidade de recursos disponíveis

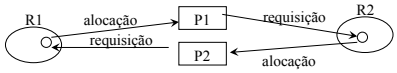
Algoritmos Distribuídos *(deadlock)*

- **Representação da relação entre processos e recursos**



Algoritmos Distribuídos *(deadlock)*

- **Deadlock**



Algoritmos Distribuídos *(deadlock)*

- **Condições necessárias para *deadlock* [Coffman, et.al., 1971]**
  - *exclusão mútua*: se recurso bloqueado por processo P, então outros processos tem que esperar processo P para usar o recurso;
  - *segura e espera*: processos podem requerer uso de novos recursos sem liberar recursos em uso;
  - *não preempção*: recurso se torna disponível somente pela liberação do recurso pelo processo;
  - *espera circular*: 2 ou mais processos formam uma cadeia circular na qual cada processo está a espera de um recurso bloqueado pelo próximo membro da cadeia.
- As quatro condições devem ser válidas ao mesmo tempo
- Se uma delas for quebrada, então *deadlock* não acontece no sistema

Algoritmos Distribuídos *(deadlock)*

- **Estratégias de tratamento de deadlock**
  - evitar
    - evita *deadlock* alocando recursos cuidadosamente
  - prevenir
    - estaticamente faz com que *deadlocks* não ocorram
  - detectar
    - permite que o *deadlock* ocorra, detecta e tenta recuperar

Algoritmos Distribuídos *(deadlock)*

- **Métodos para evitar deadlocks**
  - usam conhecimento sobre quantidades de recursos que processos irão usar, prevendo estado futuro do sistema que possa ocasionar *deadlock*
  - **noção de *safe-state:*** sistema está em um *safe-state* se não está em *deadlock* e existe uma ordem de execução dos processos do sistema que garanta que eles possam alocar os recursos necessários para prosseguir e terminar
  - **formação de uma *safe-state sequence***:
    - para qualquer processo Pi em de uma *safe sequence*, os recursos que Pi pode ainda requisitar podem ser satisfeitos pelos recursos disponíveis no momento mais os recursos bloqueados por todos processos antes de Pi na *safe sequence*
    - se Pi não pode executar imediatamente, isto significa que Pi pode esperar que os processos antes dele na *safe-sequence* acabem, para então prosseguir

Algoritmos Distribuídos *(deadlock)*

- **Métodos para evitar *deadlocks***
  - usados raramente
  - assume-se que existe conhecimento prévio sobre o comportamento dos processos - difícil na prática
  - assume-se que número de unidades de um recurso é fixa e conhecida *a priori*
  - restringe muitos pedidos de alocação que não necessariamente levariam a *deadlock*
  - em sistemas distribuídos, dificuldade para:
    - colecionar informação sobre necessidade/oferta de recursos é mais difícil

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir *deadlocks***
  - projetar o sistema de tal maneira que *deadlocks* sejam impossíveis
  - não necessita teste durante *run-time*
  - condições necessárias e suficientes:
    - exclusão mútua
    - segura e espera
    - não preempção
    - espera circular
  - técnicas: garantir que ao menos uma das condições não é nunca satisfeita
  - métodos: *collective requestes*, *ordered requests*, preempção

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *collective requests*
    - evita que condição segura e espera possa ser satisfeita
    - se um processo tem um recurso ele não pode ter outros
    - políticas
      - a) fazer *request* de todos recursos antes de sua execução: se todos recursos estão disponíveis, o processo pode executar senão, nenhum recurso é alocado e o processo espera
      - b) processo pode requerer recursos durante execução se ele liberar outros - tem que liberar todos e alocar recursos necessários até poder novamente liberar para depois alocar
    - *b* melhor que *a* pois processo pode não saber quantos recursos são necessários antes de começar a execução;

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *collective requests*
    - má utilização de recursos: processo pode ter vários recursos alocados e não usar alguns por longos períodos
    - *starvation*: se processo precisa de muitos recursos, cada vez que faz pedido pode encontrar um ou mais já alocados, tem que esperar e voltar a pedir .

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *ordered requests*
    - evita que espera circular aconteça
    - associa número a recurso
    - processo só pode alocar recursos em uma determinada ordem

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *ordered requests*

r = { r1, r2, r3, ... ,rn} // recursos
função f = r → N onde N é o conjunto dos números naturais
f (disco) = 1
f (fita) = 2
f (impressora) = 3

Processo que requisita Ri somente pode requisitar Rj se e somente se f (Rj) > f (Ri), caso contrário somente requisita Rj após liberar Ri

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - preempção
    - possibilita preempção de recurso
    - recurso preemptável: estado pode ser facilmente salvo para ser restaurado depois (ex.: CPU e memória)
    - se processo precisa de recurso, faz *request*, se outro processo que detém recurso está bloqueado a espera de outro recurso, então recurso é preemptado e passado ao primeiro processo, caso contrário primeiro processo espera
    - durante espera recursos podem ser preemptados deste processo para que outros progridam
    - processo é desbloqueado quando recurso requerido, e qualquer outro recurso preemptado dele, possam ser bloqueados para o processo

Algoritmos Distribuídos *(deadlock)*

- **Métodos para detecção de deadlocks**
  - não evita nem previne, deixa que aconteçam e depois detecta para corrigir
  - algoritmo examina estado do sistema para determinar se existe *deadlock*
  - se existe - toma ação corretiva
  - técnica equivalente a sistema centralizado
  - mantém informação sobre alocação de recursos, formando grafo de alocação de recursos, e procurando ciclos neste grafo (grafo WFG - wait for graph)

Algoritmos Distribuídos *(deadlock)*

- **Algoritmo centralizado de detecção**
  - Em cada máquina se mantém um grafo de alocação de recursos pelos processos
  - Um coordenador centralizado mantém um grafo completo do sistema (a união dos grafos locais)
  - Quando o coordenador detecta um ciclo, ele mata um dos processos e acaba com o *deadlock*

Algoritmos Distribuídos *(deadlock)*

- Diferente de sistemas centralizados, onde as máquinas estão disponíveis automaticamente, em um sistema distribuído estas informações devem ser enviadas ao coordenador explicitamente
- Como cada máquina possui um grafo local, quando um arco é incluído ou excluído do grafo local, uma mensagem deve ser enviada para o coordenador
- Problema: quando uma mensagem demora para chegar, pode causar um **falso deadlock**

Algoritmos Distribuídos *(deadlock)*

- **Algoritmo distribuído de detecção**
  - Proposto por Chandy-Misra-Haas
  - Funciona da seguinte forma:
    - iniciado quando um processo tem que esperar um recurso alocado por outro processo
    - processo envia msg (*probe message*) para processo (ou processos) que está(ao) utilizando recurso
    - msg contém 3 informações:
      - número do processo que está bloqueado
      - número do processo que enviou a msg
      - número do processo que está recebendo a msg
    - quando a msg chega a um processo, ele verifica se está esperando por recurso
      - se sim a msg é atualizada e enviada para o processo que está usando o recurso
    - se a msg dá toda a volta e chega ao processo que iniciou a msg, um ciclo existe e o sistema está em *deadlock*