Universidade Federal do Ceará

Campus Quixadá

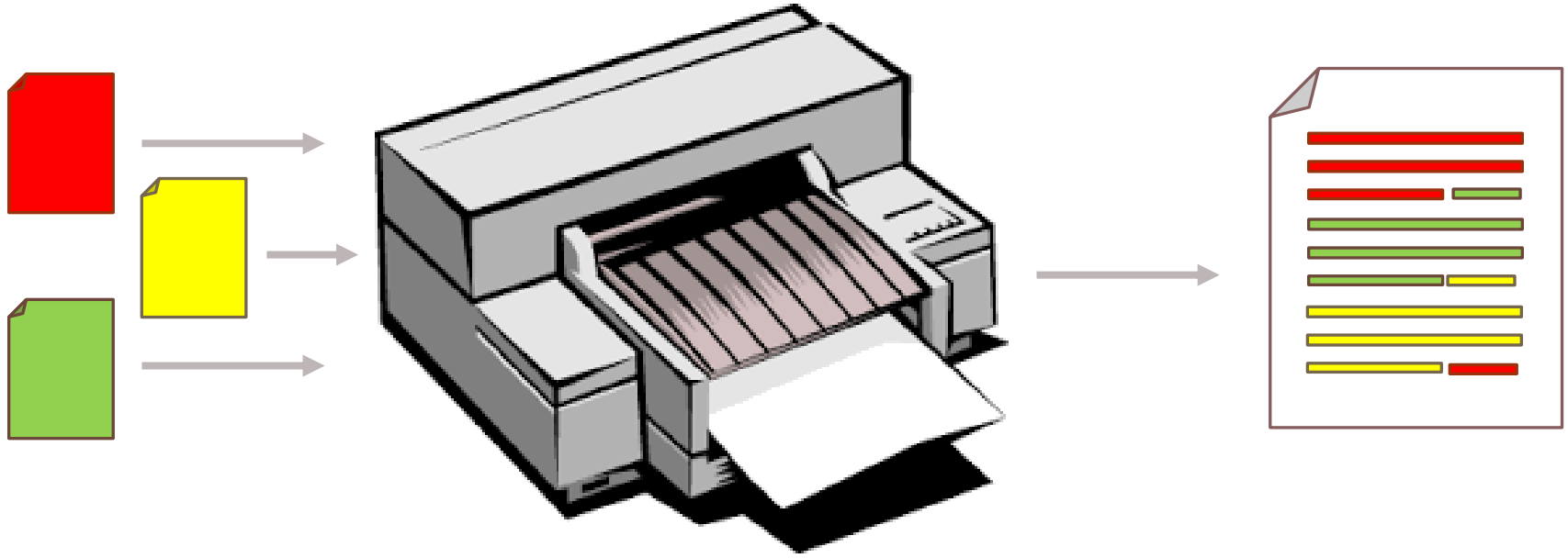
Sistemas Operacionais

IMPASSES / DEADLOCK

Introdução

Os sistemas de computadores possuem inúmeros recursos adequados ao uso, mas somente um processo pode usar por vez.

Se dois ou mais processos quiserem escrever simultaneamente na mesma impressora, isso gerará uma bagunça – também chamado de impasse ou deadlock.



Introdução

Todos os sistemas operacionais devem ser capazes de garantir (de forma temporária) o acesso exclusivo de um processo a certos recursos

Adicionalmente, para muitas aplicações, um processo necessita de acesso exclusivo não só para um recurso, mas para vários.

Qual é o problema que isso pode causar?

Introdução

Dois processos querem, cada um, gravar em CD um documento escaneado.

1. O processo P solicita permissão para usar o scanner e é autorizado
2. Já o processo Q, solicita primeiramente permissão para usar o gravador de CD, e é autorizado
3. Então, o processo P pede para usar o gravador de CD, mas a solicitação é negada até que o processo B o libere
4. Em vez de liberar o gravador de CD, o processo Q pede para usar o scanner
5. Nesse instante, ambos os processos ficam bloqueados esperando que o outro libere o recurso

Introdução

"Em alguns casos, um processo em espera não consegue modificar novamente seu estado (ou seja, estar autorizado a manipular dados da região compartilhada), porque os recursos que ele solicitou estão reservados para outros processos em espera. Essa situação é chamada de deadlock."

SILBERSCHARTZ, P. B. GALVIN, G. GAGNE (2009)

Recursos

Há dois tipos de recursos:

- **Recursos preemptíveis** – aqueles que podem ser retirados do processo proprietário sem nenhum prejuízo
  - Exemplo: memória (pode ser realocada)

- **Recursos não preemptíveis** – aqueles que não podem ser retirados do processo proprietário sem que a computação apresente falhas.
  - Exemplo: controle do gravador de CD – se for retirado, vai cancelar o processo de gravação corrente.

Deadlock

"Um conjunto de processos estará em situações de impasse se todo processo pertencente ao conjunto estiver esperando por um evento que somente outro processo desse mesmo conjunto poderá fazer acontecer."

TANENBAUM (2010)

Condições de um possível deadlock

Exclusão mútua
- Apenas um processo pode usar o recurso por vez

Segurar e esperar
- Um processo pode conter recursos alocados enquanto aguarda a atribuição de outros

Sem preempção
- Nenhum recurso pode ser forçosamente removido de um processo que o detém

Espera circular
- Encadeamento circular de dois ou mais processos; cada um à espera de um recurso que está sendo usado pelo membro seguinte da cadeia

Condições de um possível deadlock

Todas essas quatro condições devem estar presentes para que um impasse ocorra.

- Se faltar uma delas, o impasse do recurso não ocorrerá.

Importante notar que cada condição está relacionada a uma política que o sistema pode ou não adotar

- Algum recurso pode ser associado a mais de um processo por vez?
- Um processo pode ter um recurso e requisitar outro?
- Os recursos podem sofrer preempção?

Lidando com Impasses

Em geral, quatro estratégias são usadas para lidar com impasses:

Desconsiderar a ocorrência de deadlock

- Fingir que deadlocks nunca ocorrerão no sistema

Prevenir a possibilidade de ocorrência de deadlock

- Negando estruturalmente uma das quatro condições

Evitar a ocorrência de deadlocks

- Por meio de uma alocação cuidadosa de recursos

Detectar a ocorrência de deadlock e recuperar

- Deixar que os impasses ocorram

Seção 1

IGNORANDO O PROBLEMA

Algoritmo do Avestruz

- **"enfie a cabeça na areia e finja que não há nenhum problema".**
- Pressupõe é mais rentável permitir que o problema ocorra do que tentar a sua prevenção.

Algoritmo do Avestruz

Estratégia mais econômica

- Projetista tem conhecimento de que o problema pode ocorrer
  - Mas finge que não há problemas
  - Dado que a ocorrência de deadlocks é baixa e o impacto é suave, pode ser uma boa alternativa.

Decisão é tomada levando em consideração

- Frequência com que isto pode ocorrer
- O problema que isto pode causar

Algoritmo do Avestruz

E quando acontece o deadlock ?

- Trata-se os sintomas do problema!

PERGUNTA!

Qual a relação do algoritmo do avestruz com a solução de não considerar os deadlocks? Na sua opinião, porque os sistemas operacionais optam por essa solução?

Seção 2

DETECÇÃO E RECUPERAÇÃO DE IMPASSES

Detecção e Recuperação de Impasses

Uma segunda técnica é a detecção e recuperação.

O sistema não tenta prevenir a ocorrência do deadlock

- Ele deixa que ocorra e tenta detectá-lo a medida que ocorre
- Busca se recuperar após a ocorrência

Existem algumas maneiras de detectar e recuperar os impasses

Detecção de Impasses com um Recurso de cada Tipo

Caso mais básico:

- Só tem um recurso de cada tipo

Tomemos como exemplo um sistema com sete processos (de A a G) e seis recursos (de R a W).

- Elementos
  - Processos de A a G
  - Recursos de S a W

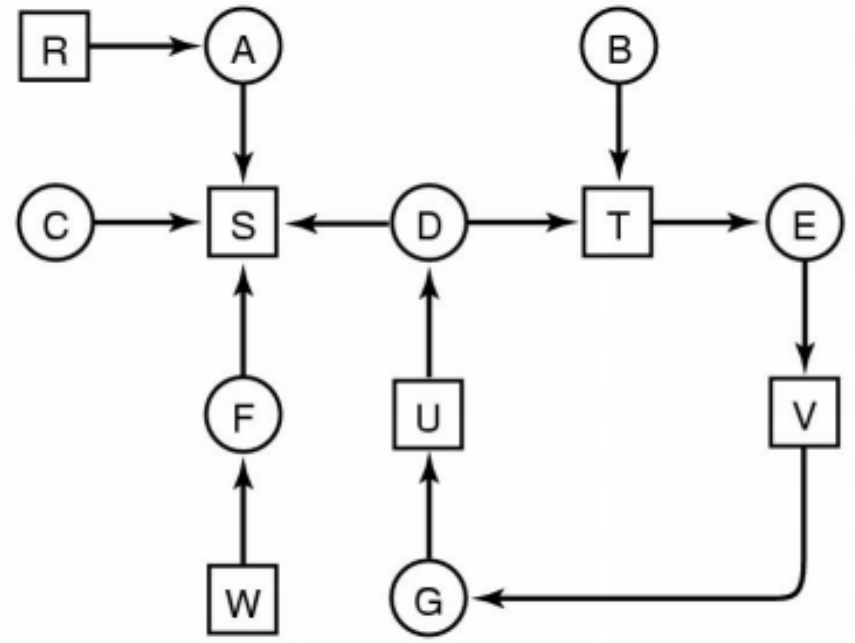
- Uso e requisições de recursos
  - Processo A - Possui o recurso R, Requisita S
  - Processo B - Requisita T
  - Processo C - Requisita S
  - Processo D - Possui o recurso U, Requisita S e T
  - Processo E - Possui o recurso T, Requisita V
  - Processo F - Possui o recurso W, Requisita S
  - Processo G - Possui o recurso V, Requisita U

PERGUNTA!

Esse sistema está em impasse? Se tiver, quais os processos envolvidos?

- Elementos
  - Processos de A a G
  - Recursos de S a W

Os processos são representados por círculos e os recursos por quadrados!



- Uso e requisições de recursos
  - Processo A - Possui o recurso R, Requisita S
  - Processo B - Requisita T
  - Processo C - Requisita S
  - Processo D - Possui o recurso U, Requisita S e T
  - Processo E - Possui o recurso T, Requisita V
  - Processo F - Possui o recurso W, Requisita S
  - Processo G - Possui o recurso V, Requisita U

Detecção de Impasses com um Recurso de cada Tipo

Exclusão mútua
- Apenas um processo pode usar o recurso por vez

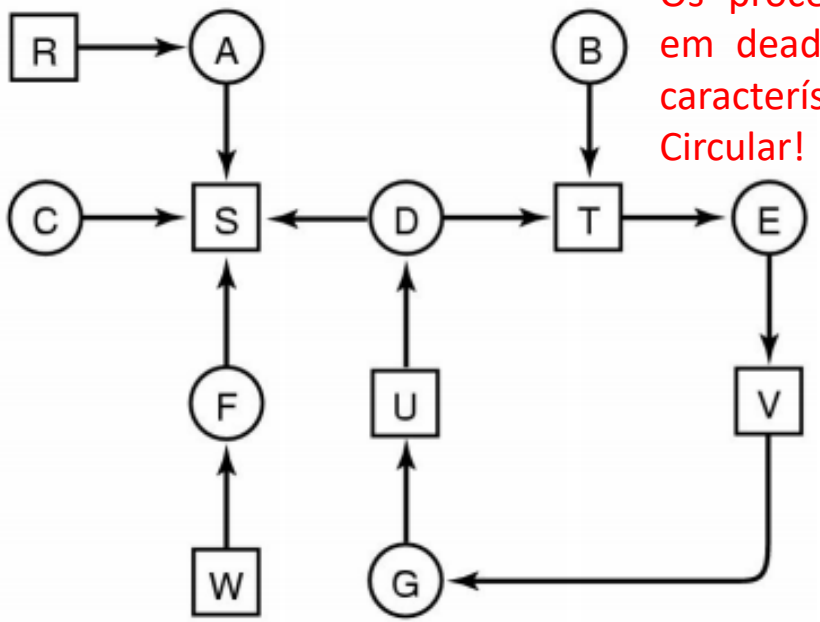
Segurar e esperar
- Um processo pode conter recursos alocados enquanto aguarda a atribuição de outros

Sem preempção
- Nenhum recurso pode ser forçosamente removido de um processo que o detém

Espera circular
- Encadeamento circular de dois ou mais processos; cada um à espera de um recurso que está sendo usado pelo membro seguinte da cadeia

Podemos observar que os processos D, E e G estão em situação de impasse

Os processos A, C e F não estão sofrendo impasse
- Pois o recurso S é passível de ser alocado a qualquer um deles



Os processos que não estão em deadlock não possuem a característica da Espera Circular!

Detecção de Impasses com um Recurso de cada Tipo

Embora seja relativamente simples detectar visualmente os processos em situação de impasse, para uso em sistema real, é necessário um algoritmo formal de detecção

Exercício:

- Em casa, pesquise sobre os algoritmos para detecção de ciclos em grafos dirigidos.

Para cada nó, execute os passos que segue

Passo 1:

- Inicializar uma lista vazia (L) e assinale os arcos como desmarcados

Passo 2:

- Insira o nó atual no final da lista (L) e analise se o nó aparece duas vezes
  - Se sim: há um ciclo (finaliza-se a execução do algoritmo)
  - Se não: ainda não foi detectado ciclo (siga para o passo 3)

Passo 3:

- No nó corrente, verificar se há um arco de saída desmarcado
  - Se sim, siga para o passo 4, senão siga para o passo 5

Passo 4

- Escolha uma das saídas desmarcadas
  - Marque-a, siga-o para obter o novo nó, e vá para o passo 2

Passo 5

- Se o nó é o inicial, então não há ciclo e o algoritmo pode ser encerrado
- Se o nó não é o inicial, então o final foi alcançado
  - Remova-o da lista; volte ao nó anterior; marque-o como o nó atual e volta ao passo 2

Detecção de Impasses com um Recurso de cada Tipo

O que esse algoritmo faz é tomar cada nó, um após o outro, como nó "de uma árvore" e fazer uma busca em profundidade

- Se tornar a passar por um nó já percorrido, isso significa que encontrou um ciclo

PERGUNTA!

E quando existe várias cópias do mesmo recurso?

- Neste cenário, é necessário a utilização de métodos diferentes para a detecção de impasses

EXERCÍCIO:

- Em casa leia a seção 6.4 do livro TANEMBAUM (2010) sobre os algoritmos de detecção de impasses com um ou múltiplos recursos

Recuperação de Situações de Impasse

Após a detecção do impasse, o segundo passo é recuperar, de algum modo, o sistema

- O colocando em condições normais de execução

Existem algumas soluções para esta situação, dentre elas:

- Recuperação por meio de preempção
- Recuperação por meio de retrocesso
- Recuperação por meio de eliminação de processo

Recuperação por Preempção

Em alguns casos pode ser possível retomar provisoriamente um recurso de seu proprietário atual para dá-lo a outro

A habilidade de retirar um recurso de um processo, entrega-lo a outro e depois devolvê-lo, sem que perceba, é *altamente dependente da natureza do recurso*

- A escolha do processo que será suspenso depende amplamente de quais processos têm recursos passíveis de serem facilmente retomados

Recuperação por Retrocesso

Projetistas preocupados com impasses poderão fazer com que os processos salvem periodicamente seus estados

- Guardando seu estado em um arquivo

Para fazer a recuperação, um processo que tem um dos recursos necessários é revertido para um estado em um instante anterior

- Isso é feito reinicializando o processamento a partir de um de seus pontos de salvamento
- Chamado também de rollback

Recuperação por Eliminação

A maneira mais grosseira de eliminar um impasse – mas também a mais simples – é “matar” um ou mais processos.

Busca-se matar um processo presente no ciclo

- Com um pouco de sorte, os demais processos serão capazes de prosseguir
- Se não for suficiente, essa ação poderá ser repetida até o ciclo ser quebrado

Recuperação por Eliminação

Abortar um processo em deadlock de cada vez:

- Solução amena
- A cada processo abortado, o algoritmo de detecção deve ser invocado para determinar se ainda existe deadlock

Seção 3

EVITANDO IMPASSES

Evitando Impasses

Presumimos que, quando um processo requisita recursos, ele requisita todos de uma só vez

- No entanto, esses recursos são requisitados um de cada vez

Assim, o sistema deve ser capaz de decidir se liberar um recurso é seguro ou não

Evitando Impasses

"Um método alternativo para evitar deadlocks *é demandar dados adicionais* sobre como os recursos devem ser solicitados. Com o conhecimento da sequência completa de solicitações e liberações de cada processo, o sistema pode decidir se o processo deve ou não esperar de modo a impedir um possível deadlock futuro."

SILBERSCHARTZ, P. B. GALVIN, G. GAGNE (2009)

Evitando Impasses

Considere o cenário com dois recursos – impressora e drive de disco – e dois processos P e Q

- O processo P aloca os recursos impressora e drive, nessa ordem.
- O processo Q aloca os recursos drive e impressora, também nessa ordem

De posse dessa informação, o sistema operacional poderia reconhecer uma possibilidade de deadlock

PERGUNTA!

Como o sistema operacional pode reconhecer esses dados e chegar a essa conclusão?

Evitando Impasses

Primeiro passo:

- Exigir que os processos declarem o máximo de recurso que eles potencialmente vão precisar

Segundo passo:

- Sistema operacional aplica um *algoritmo de impedimento*
  - Este algoritmo verifica (dinamicamente) o estado da alocação de recursos para garantir que nunca aconteça uma espera circular

Evitando Impasses

Existem dois principais algoritmos para análise de impedimentos:

- Algoritmo do grafo de alocação do recurso
- Algoritmo do banqueiro

Algoritmo do grafo de alocação do recurso

Grafo composto por um conjunto de vértices V e um grupo de arestas A, onde o conjunto de V é dividido em duas partes:

- O conjunto P, composto pelos processos ativos
- O conjunto R, composto pelos recursos do sistema

Uma aresta orientada entre um processo e um recurso indica que o processo P_i **solicitou** o recurso R_j ($P_i \rightarrow R_j$)

Uma aresta orientada entre um recurso e um processo indica que o o recurso R_j **foi dado** ao processo P_i ($P_i \leftarrow R_j$)

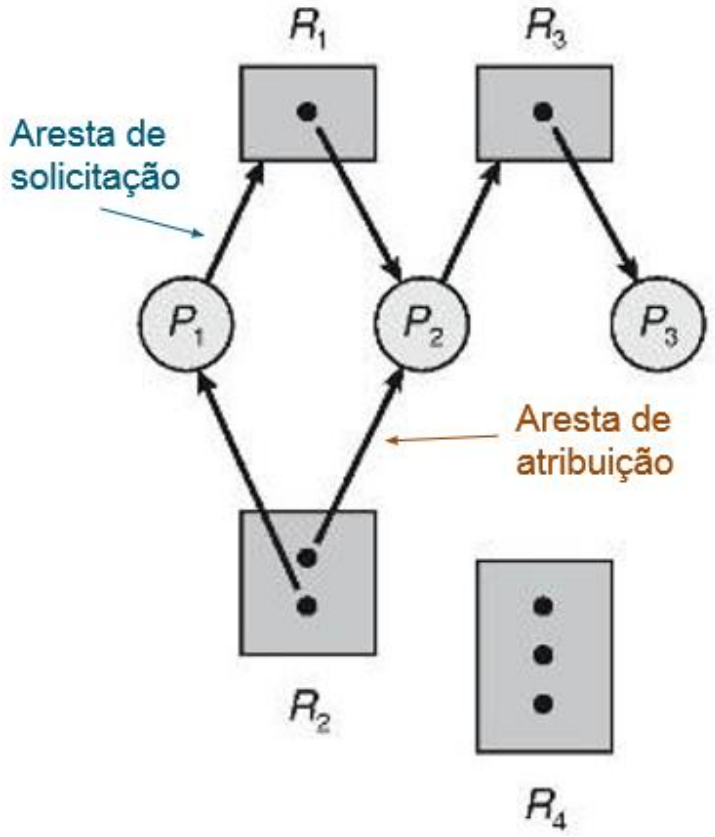
Algoritmo do grafo de alocação do recurso

Os círculos do grafo representam os processos ativos no sistema

Os retângulos representam os recursos

As setas de P para R indicam sua solicitação

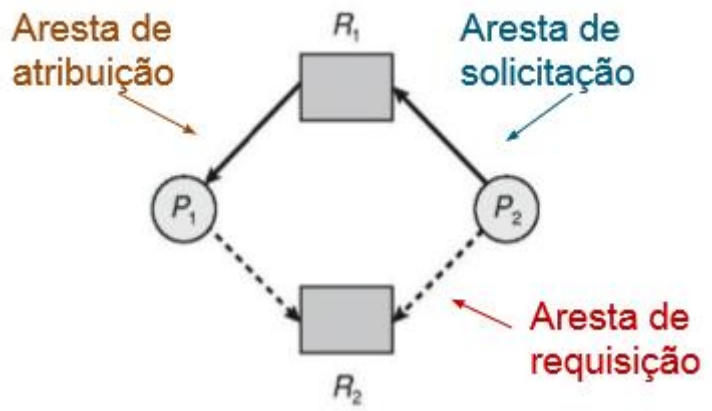
As setas de R para P indicam sua atribuição



Algoritmo do grafo de alocação do recurso

Uma terceira aresta é utilizada na representação do grafo

- A **aresta de requisição**
- É representada por uma seta orientada tracejada



Algoritmo do grafo de alocação do recurso

Se o grafo não possui ciclos

- Não existe deadlock

Se o grafo possui ciclos

- Há apenas uma instância por tipo de recurso?
  - Sim! Tem-se deadlock
- Há mais de uma instância por tipo de recurso?
  - Sim! Há a possibilidade de um deadlock

Em alguns casos, as solicitações podem levar a um estado inseguro

Algoritmo do banqueiro

Algoritmo de escalonamento desenvolvido por Dijkstra em 1965

- Faz uma analogia ao empréstimo de crédito de um banqueiro aos seus clientes

Este algoritmo considera cada solicitação de empréstimo quando ela ocorre, analisando se sua concessão levará a um estado seguro

- Se levar, a requisição será atendida

Algoritmo do banqueiro

Para saber se um estado é seguro, o banqueiro verifica se ele dispõe de recursos suficientes para satisfazer algum dos clientes

- Se sim
  - Presume-se que os empréstimos a esse cliente serão devolvidos
  - Em seguida, o cliente mais próximo ao limite é considerado, e assim por diante
  - Se todos os empréstimos puderem ser devolvidos, o estado será seguro e a requisição inicial poderá ser atendida

Seção 4

PREVENÇÃO DE IMPASSES

Prevenção de Impasses

O objetivo é impedir que ao menos uma das quatro condições estabelecidas por Coffman et. al. (1971) não sejam satisfeitas

- Exclusão mútua, posse e espera, espera circular e não preempção

Como impedir a **Exclusão Mútua**?

- Utilizar, sempre que possível, recursos compartilháveis, pois não requerem acesso mutualmente exclusivos

Prevenção de Impasses

Como evitar **Posse e Espera**?

- Impedir que processos que já mantêm a posse de recursos esperem por mais recursos

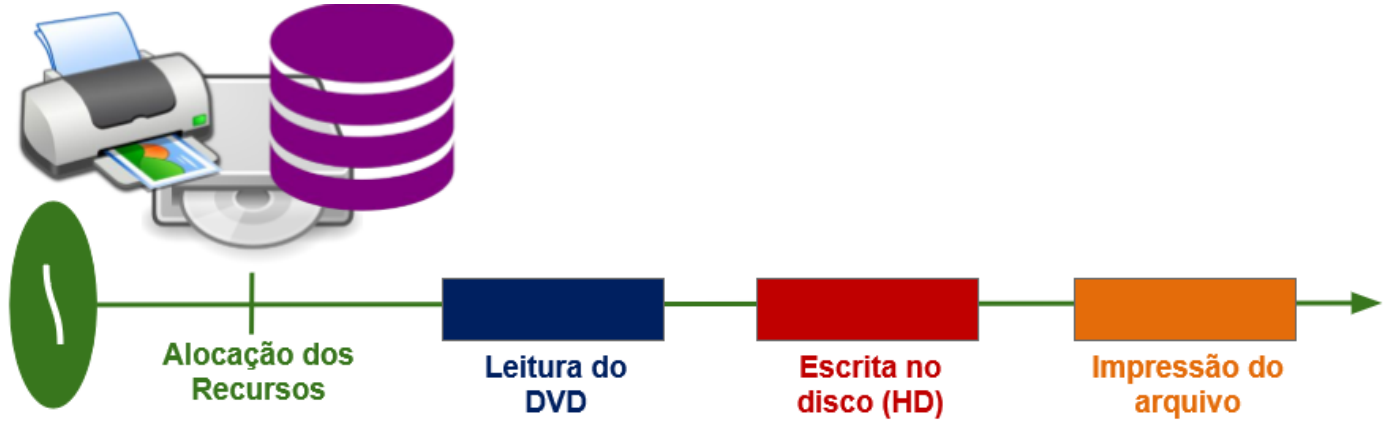
Duas soluções possíveis:

1. Exigir que todos os processos requisitem todos os seus recursos antes de inicializarem suas execuções
2. Todos os processos solicitam recursos, contudo só os recebem quando liberarem seus recursos atuais

Prevenção de Impasses

Exemplo da Solução 1:

- Considere um processo que copie dados de um drive de DVD para um arquivo em disco, persista o arquivo e, então, imprima os resultados em uma impressora
- TODOS OS RECURSOS SÃO ALOCADOS ANTES DO PROCESSO INICIAR



Prevenção de Impasses

Exemplo da Solução 2:

- Considere um processo que copie dados de um drive de DVD para um arquivo em disco, persista o arquivo e, então, imprima os resultados em uma impressora
- CADA RECURSO SÓ É ALOCADO QUANDO OS RECURSOS ANTERIORES SÃO LIBERADOS

Prevenção de Impasses

Como evitar a **Não Preempção**?

- Revogar (forçado) os recursos de um ou mais processos

Exemplo: considere um cenário com dois processos: um com os recursos A e B e outro com o recurso C

- O primeiro processo deseja alocar o recurso C para si
- Deve-se fazer a análise:
  - O segundo processo também está esperando por algum outro recurso?
    - Se não, o primeiro processo aguarda a liberação do recurso C
    - Se sim, o primeiro processo tem todos os seus recursos revogados e liberados a outros processos

Prevenção de Impasses

Como evitar a **Espera Circular**?

- Impor uma ordenação absoluta aos recursos e exigir que os processos os solicitem em ordem crescente
- Cada recurso recebe um número **inteiro e exclusivo**
- Cada processo só pode solicitar recursos em uma ordem de numeração **crescente**
  - Deste modo, caso deseje dois recursos, o processo deve primeiro alocar aquele de menor identificador

Prevenção de Impasses

Só pode haver um impasse se A solicitar o recurso j, e B solicitar o recurso i.

Presumindo que i e j são recursos distintos, eles terão números diferentes.

Se i > j, então A não tem permissão para solicitar j porque este tem ordem menor do que a do recurso já obtido por A. Se i < j, então B não tem permissão para solicitar i porque este tem ordem menor do que a do recurso já obtido por B.

De qualquer maneira, o impasse é impossível.

Prevenção de Impasses

"O desenvolvimento de uma ordenação não impede, por si só, o deadlock. É responsabilidade dos desenvolvedores de aplicações escrever programas que sigam a ordenação. Observe também que a distribuição dos identificadores deve seguir a ordem natural de uso dos recursos do sistema."

SILBERSCHARTZ, P. B. GALVIN, G. GAGNE (2009)

Prevenção de Impasses

FIGURA 6.14 Resumo das abordagens para prevenir impasses.

Condição	Abordagem contra impasses
Exclusão mútua	Usar spool em tudo
Posse e espera	Requisitar todos os recursos necessários no início
Não preempção	Retomar os recursos alocados
Espera circular	Ordenar numericamente os recursos