PCS 3446 - Sistemas Operacionais

Prof. João José Neto

AULA 10

Administração de Processos (2)

Conceitos Gerais

Process Scheduling (Dispatching, Low-Level Scheduling)

- Nos exemplos vistos, o tempo do processador era igualmente dividido entre os jobs multiprogramados.
- Todavia, o *job scheduler* e o *process scheduler* podem interagir: por exemplo, o segundo pode decidir remover, ou pospor, a execução de um processo.
- Neste caso, há necessidade de colocar o job afetado novamente no nível hierárquico mais externo de *scheduling* para que o sistema possa completá-lo.
- Isso acontece em alguns sistemas de *time-sharing.*

- Se, por exemplo, houver vários processos competindo pela posse de recursos (por exemplo, fitas magnéticas ou impressoras físicas), há a necessidade de bloquear algum ou alguns dos processos até que aqueles que estejam na posse dos recursos solicitados possam liberá-los.
- Enquanto isso, o tempo ocioso do processador fica liberado, para uso por outros processos.
- Observe-se que às vezes o bloqueio de processos pode permitir um remanejamento de recursos livres entre processos não bloqueados.

Funções do *Process Scheduling*

- Acompanhar os processos no sistema, bem como os respectivos estados.
- Decidir qual desses processos irá receber o processador, quando, e por quanto tempo.
- Alocar e recuperar o processador aos processos prontos, nas ocasiões oportunas.
- O módulo do *process scheduler*, que controla e acompanha o estado dos processos, é também chamado *traffic controller*.

Process Control Block

- Para isto, o sistema utiliza para cada processo um PCB (*process control block*), estrutura de dados que abriga, entre outras, as seguintes informações:
  - identificação do processo
  - estado corrente do processo
  - prioridade do processo
  - cópia do conteúdo dos registradores de trabalho do processo
  - apontador para outros processos que estão no mesmo estado (*link* de fila).

Ready-list

- Normalmente formam-se listas ligadas de PCB's relativos a processos no mesmo estado.
- Tais listas às vezes recebem nomes especiais. Por exemplo, as listas de processos no estado "pronto" são chamadas *ready-lists*.
- O *traffic controller* é ativado sempre que o sistema decide executar alguma operação de mudança de estado dos processos.

- As diversas listas podem estar, por sua vez, subdivididas em classes menores (por exemplo, dedicando-se uma lista a cada recurso).
- Quando o recurso é liberado, o *traffic controller* verifica se há algum processo na fila a ele associada, e, se isto for verdade, esse processo é colocado no estado "pronto".

Políticas de Alocação do Processador

- Os *dispatchers* devem:
  - decidir **qual processo deve receber o processador** em cada instante.
  - decidir sobre o **tempo de utilização do processador**. Este depende de ocorrer uma das interrupções seguintes:
    - o processo **terminar**
    - o processo **ser bloqueado** (p/exemplo: pedido de Entrada/Saída)
    - o processo **ser retirado do processador** por algum processo mais prioritário
    - o processo **ser interrompido** por fim de ***time-slice***
    - o processo **ser interrompido** por ter **excedido o tempo máximo** de UCP a que tem direito
    - o processo **ser descontinuado por** ocorrência de algum **erro** (por exemplo *Overflow*, instrução ilegal, violação de memória, etc).

Dispatchers

- Os *dispatchers* decidem qual processo será executado a seguir, examinando os componentes da *ready-list* e aplicando sobre eles alguma política de alocação.
- Descrevem-se a seguir as políticas mais utilizadas para a alocação do processador:
  - *round-robin*,
  - inverso do resto do *quantum*,
  - prioridade,
  - balanceamento do sistema,
  - mínimo tempo de resposta,
  - automática.

Round-robin

- Nesta política, muito simples, os processos são todos tratados igualmente.
- Cada processo ganha algum ***quantum* fixo** (tipicamente da ordem de 100ms), e a ***ready-list* se reduz a uma fila circular**.
- Os processos prontos para a execução são sempre inseridos, em ordem de chegada, no final da *ready-list*.

Inverso do Resto do Quantum

- Este método é bastante justo, e funciona bem para **jobs com entradas/saídas frequentes**, pois dá **preferência** de alocação do processador a jobs que efetuam muito processamento (processos ***CPU-bound***):
  - Se o processo usou todo o *quantum* na última vez que recebeu o processador, vai para o final da *ready-list*.
  - Se usou a metade do *quantum*, vai para o centro da lista, e assim por diante.

Prioridade

- A política de inserção de um processo na ***ready-list*** resulta, neste caso, da **ordenação** da mesma **de acordo com a prioridade** atribuída ao processo ou ao job a que pertence tal processo.
- O resultado da aplicação desta política depende, portanto, do critério (muitas vezes, arbitrário) adotado para a atribuição de prioridades aos processos.

Balanceamento do Sistema

- Esta política dá preferência aos **processos *I/O- bound*** (processos que efetuam muitas operações de Entrada/Saída, como é o caso de softwares comerciais de processamento de folhas de pagamento e similares), com a finalidade de procurar, na medida do possível, **evitar a ociosidade dos periféricos**.

Mínimo Tempo de Resposta

- Devem ter preferência neste caso os **processos interativos**, para que seja obtida uma maior **rapidez no atendimento a terminais *on line***.
- Isto se justifica pelo efeito psicológico a que está sujeito um operador que utiliza o sistema através de um terminal: esta classe de usuários sente-se desconfortável quando tem a sensação, mesmo que falsa, de que o sistema operacional não esteja dando permanente atenção a seu terminal.

Automática

- Às vezes é conveniente que o próprio sistema estabeleça (**automaticamente**) **prioridades** para os processos, segundo os **méritos e deméritos dinâmicos** dos mesmos, ou então para **balancear a carga** do computador, ou para atingir algum **índice de mérito** estabelecido qualquer.

- Às vezes o sistema **aumenta a prioridade** de um processo que **utiliza muitos recursos**, pois dessa forma ele pode ser favorecido para terminar mais depressa, e assim liberar os muitos recursos a ele alocados, o que não seria possível comodamente se a sua prioridade não pudesse ser alterada dinamicamente.

Sincronização de Processos

- Surge da necessidade de compartilhar, entre os processos, os recursos do sistema.
- Devido ao compartilhamento de recursos, surgem "condições de competição" e os "deadlocks".
- Ocorrem condições de competição sempre que vários processos requisitam simultaneamente os mesmos recursos.
- Nestas condições, é possível o aparecimento de "corridas" pelo uso de algum recurso compartilhado (por exemplo, uma impressora).
- Nestes casos, dependendo da ordem de alocação, os resultados globais do processamento podem ser diferentes.

- Por exemplo, as saídas geradas por uma impressora podem ser totalmente diferentes, dependendo da ordem em que ela é escalada para atender os processos requisitantes.
- Se não se tomarem os devidos cuidados na alocação da impressora aos diversos processos, as listagens poderão, inclusive, sair totalmente embaralhadas.
- Para este último problema, uma possível solução é a de exigir que o processo sempre requeira o uso de um recurso antes de poder efetivamente utilizá-lo.
- O recurso passa a estar bloqueado para outros processos até que seja liberado.

Sincronização de Processos

- Além dos dispositivos, também as tabelas e arquivos compartilhados necessitam de sincronismo para seu uso.
- Por exemplo: O recurso é a "ready-list", e o processo é o "scheduler", que é um programa que determina qual deve ser o próximo processo a receber o processador.
- Se houver mais de um processador no sistema, poderá haver problema de conflito: possivelmente em um dado instante dois processadores estarão disponíveis para executar o mesmo processo, e possivelmente algum processo poderá deixar de ser executado devido à aplicação de um critério inconveniente de alocação de processador.

- Uma solução para este problema é a utilização de um "lock byte", variável que indique se os dados estão ou não em uso no instante da requisição.
- Observe-se que é fundamental que haja cooperação entre os processos, sem o que o sincronismo não funcionará.
- Se um processo, proposital ou acidentalmente, deixar de liberar seus recursos, tais recursos permanecerão inacessíveis aos demais processos do sistema.
- Por isso, por precaução, os sistemas operacionais costumam liberar todos os recursos do processo ao ocorrer o seu final de execução, para garantir a devolução dos recursos a ele alocados, mas não espontaneamente liberados.

Operações Primitivas

- Note-se que, uma vez iniciadas, todos esses procedimentos devem ser executados até o final sem interrupções, ou seja, tais operações devem ser monolíticas, implementadas como regiões críticas, caso contrário, haverá o risco concreto de inconsistências na sincronização.
- A exemplo de outras situações envolvendo a atualização de informação do sistema, essas operações devem ser sempre implementadas com o sistema de interrupções da máquina desativado.
- Rotinas do sistema com essas características são chamadas operações primitivas, ou operações atômicas, pois se comportam monoliticamente, como se fossem instruções de máquina: uma vez iniciadas, só poderá haver o atendimento de pedidos de interrupção após o término de sua execução.

Primitivas para sincronização

Todas as primitivas são executadas com a interrupção inibida.

- **LOCK/UNLOCK** – Mecanismo primitivo de sincronização que fecha/abre o acesso a uma região crítica do programa
- **WAIT/SIGNAL** – Mecanismo primitivo para efetuar o bloqueio/desbloqueio de processos
- **P/V** – Mecanismo de utilização de semáforos contadores para sincronizar processos solicitando/liberando recursos compartilhados
- **SEND/RECEIVE** – Mecanismo de sincronização usando o sistema de comunicação para enviar/aguardar o recebimento de mensagens que tenham a finalidade de comunicar e/ou sincronizar os processos mutuamente

LOCK/UNLOCK

LOCK (X)

Inibir o tratamento de interrupções

Examina se o semáforo X é = 0 ou ≠ 0

Faz semáforo ≠ 0

Se o valor original já era ≠ 0, chama WAIT (X)

Liberar o tratamento de interrupções

UNLOCK (X)

Inibir o tratamento de interrupções

Faz semáforo = 0

Chama SIGNAL (X)

Liberar o tratamento de interrupções

WAIT/SIGNAL

WAIT (X)

Inibir o tratamento de interrupções

Coloca como bloqueado o processo requisitante, inserindo-o no final da fila dos processos no estado de "aguardando o recurso X".

Liberar o tratamento de interrupções

SIGNAL (X)

Inibir o tratamento de interrupções

Verifica a fila dos processos "aguardando o recurso X".
Se houver algum, o primeiro da fila é selecionado e marcado como "Pronto" para ser executado.

Liberar o tratamento de interrupções

Operações P e V em Semáforos Contadores

- Uma forma mais geral para as operações LOCK/UNLOCK é dada pelas operações P e V, definidas por Dijkstra, e que operam sobre semáforos, baseados em variáveis contadoras que podem assumir valores inteiros:

P(X):

Inibir o tratamento de interrupções

Faz X ← X – 1

Se X<0 então faz uma chamada para a rotina WAIT(X)

Liberar o tratamento de interrupções

V(X):

Inibir o tratamento de interrupções

Faz X ← X + 1

Se X ≤ 0 então chama a rotina SIGNAL(X)

Liberar o tratamento de interrupções

- Usando valores iniciais adequados, as operações P e V podem servir para várias finalidades na sincronização de processos.

Exemplo: Produtor/Consumidor

- Segue uma implementação do clássico problema do produtor e do consumidor de recursos.
- Produtor é um processo que disponibiliza recursos.
- Consumidor é um processo que utiliza tais recursos.
- N representa o número total de células existentes no sistema
- X1 semáforo que controla o uso de células vazias, nas quais um recurso pode ser disponibilizado
- X2 semáforo que controla o uso de células cheias, cujo conteúdo é um recurso disponível, mas ainda não consumido
- P/V funcionam como requisição/disponibilização de célula vazia (se referenciar X1) ou de célula cheia (se referenciar X2)
- Valores iniciais: X1=N (o número inicial de células vazias é N) e X2=0 (o número inicial de células cheias é 0)

Lógica básica do par Produtor-Consumidor

Produtor:	Consumidor: P (X2)
.....	.....
Produz um elemento	Retira elemento do "buffer"
P (X1)	.....
.....	V (X1)
Coloca elemento no "buffer"	.....
.....	Consome o elemento
V (X2)	.....
Go to Produtor	Go to Consumidor

Dinâmica de um par de processos produtor-consumidor

Estouro de fila bloqueia o produtor

P	C	Fila (MÁX.=3)
		>>
1		>1>
2		>2 1>
3		>3 2 1>
	1	>3 2>
4		>4 3 2>
	2	>4 3>
5		>5 4 3>
	3	>5 4>
	4	>5>
	5	>>
	BL	>>
6	~~BL~~	>6>
	6	>>
7		>7>

P	C	Fila (MÁX.=3)
	7	>>
8		>8>
9		>9 8>
10		>10 9 8>
11 BL		>10 9 8>
~~BL~~	8	>11 10 9>
	9	>11 10>
12		>12 11 10>
	10	>12 11>
	11	>12>
13		>13 12>
	12	>13>
14		>14 13>
	13	>14>
	14	>>
	BL	

Desbloqueio

Desbloqueio

Fila vazia bloqueia o consumidor

Fila vazia bloqueia o consumidor

Comunicação por mensagens: SEND/RECEIVE

SEND (P_r, M)

O processo correntemente em execução envia para o processo P_r a mensagem M.

RECEIVE (P_s, M)

O processo corrente obtém a mensagem M e a identificação do processo P_s que a enviou.
Se não havia nenhuma mensagem para o processo que executou a rotina RECEIVE, este processo ficará bloqueado até que para ele seja enviada uma mensagem.

"DEADLOCKS"

São situações em que há uma cadeia de processos, em que um processo aguarda que outro seja completado, este a um terceiro, etc., e o último, ao primeiro.

O sistema entra em colapso, pois os processos não se completam devido à espera cíclica.

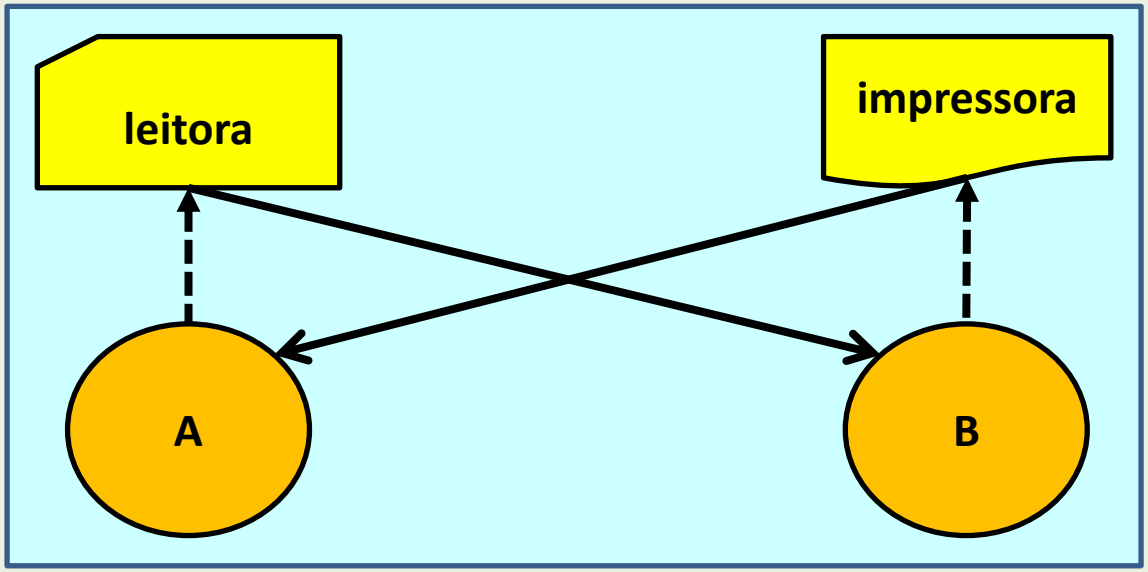
Exemplo

Sejam os dois processos (concorrentes) seguintes:

Processo A:	**Processo B:**
1. Requisita Impressora	1. Requisita Leitora
2. Requisita Leitora	2. Requisita Impressora
3. Libera Impressora	3. Libera Impressora
4. Libera Leitora	4. Libera Leitora

Instala-se um “deadlock” quando por exemplo ocorre a seguinte sequência:

A1, B1, A2, B2:



Métodos de prevenção de “deadlock”

- Há diversos métodos usados para evitar os efeitos da instalação de um “deadlock” em um sistema operacional.
- Os mais usuais, a serem detalhados a seguir, são:
  - Pré-alocação dos recursos
  - Alocação Restringida
    - Alocação Controlada
    - Alocação Padronizada
  - Detecção e Recuperação

Pré-alocação dos recursos

- Se cada processo, antes de ser executado, obrigatoriamente solicitar **todos** os recursos que lhe serão necessários, e se os processos ficarem bloqueados enquanto todos os recursos solicitados não lhe forem alocados, o aparecimento dos "deadlocks" é evitado de forma simples e segura.
- **Desvantagens:**
  - É necessário esperar que **todos** os recursos a serem utilizados pelo processo estejam **disponíveis**.
  - Durante a não utilização dos recursos, estes ficam mesmo assim bloqueados pelo processo, o que causa **desperdício**.

Alocação Restringida

- Impondo-se restrições ao tipo de alocação permitida, é possível impedir que venham a acontecer determinadas situações nas quais possa instalar-se um "***deadlock***".
- Dois tipos de restrições, descritos adiante, são usuais para isso:
  - Alocação controlada
  - Alocação padronizada

Alocação Controlada

- O supervisor deve detectar a possibilidade do aparecimento de "*deadlock*", **evitando-se a alocação** se essa possibilidade existir.
  - Detectada a eventual possibilidade de "*deadlock*", o job terá sua execução bloqueada até que essa possibilidade tenha sido eliminada.
  - Essa técnica exige que o número máximo de recursos a ser alocado seja declarado "a priori" (desvantagem).
- Desvantagens:
  - precisa-se conhecer previamente as necessidades do job
  - o job pode precisar esperar desnecessariamente, uma vez que na prática um possível "*deadlock*" teórico pode não se concretizar.

Algoritmo de alocação controlada de recursos

O seguinte algoritmo pode ser utilizado na **alocação controlada** (cada dispositivo e cada processo possuem um marcador associado)

1. Limpar todos os marcadores
2. Supor que um recurso seja alocado
3. Há algum processo que possa ser completado, nessa situação?
4. **Sim**. Marcar o processo e simular a recuperação dos recursos a ele alocados. Voltar ao passo 3 (há mais recursos agora).
5. **Não**. Há processos não marcados?
6. **Sim**: A proposta não é segura. A suposta alocação não deve ser efetuada.
7. **Não**: A proposta é segura. A suposta alocação pode ser concretizada.

Alocação Padronizada

- Numeram-se arbitrariamente os dispositivos
- As requisições devem ser feitas obrigatoriamente em uma ordem arbitrariamente escolhida (por exemplo, crescente) da numeração dos dispositivos
- Se houver alocação possível, executá-la; se não, esperar
- Dispensa o conhecimento prévio das necessidades de uso dos recursos pelo programa
- Problema: a requisição deve ser sempre feita na ordem certa
- "Solução": em algumas situações, atribuir números convenientes aos recursos pode reduzir esse problema.
- É usada em casos bem especiais
  (por ex. em tabelas internas do OS/MVT)

Detecção e Recuperação

- Com este método, permite-se a ocorrência de "deadlock" desde que seja possível detectá-lo.
- Mantêm-se, neste caso, duas tabelas:
- **recursos** × **processos** ao qual o recurso está alocado. (**REC**)
- **processo** × **recursos** que este processo está aguardando (**PRO**).
- Sempre que os recursos são alocados/liberados, deve-se verificar se há possibilidade de se ter instalado um "deadlock“ como efeito colateral.

- Instalado um "deadlock", deve-se removê-lo: jobs cujos recursos estão provocando o "deadlock" devem liberá-los.
- Observe-se que a recuperação de um "deadlock" nem sempre é possível ou economicamente viável.
- É o que se chama de "backtracking": voltar atrás algum ou alguns processos "descomputando" algum processamento já realizado.
- Para isto é necessário guardar "backups" periódicos dos estados dos diversos processos para que se possam recuperar tais estados em casos de necessidade.
- (Aparecem muitos problemas de viabilidade prática quando estiverem envolvidos arquivos do usuário).

Algoritmo de detecção de "deadlock"

1. Supor que o recurso I está sendo requisitado pelo Processo J.
2. Colocar I em PRO (tabela de processos × recursos aguardados).
3. Procurar I em REC, buscando o processo K que o bloqueia.
4. Indexar PROC com K.
5. K está aguardando algum recurso I'?
6. Não: então J pode esperar (não há "deadlock"): FIM.
7. Sim: procurar I' em REC, para saber qual processo (K') o bloqueia.
8. K'=J?
9. Sim: um "deadlock" foi detectado; deve-se recuperá-lo
10. Não: É o fim da tabela PRO?
11. Sim: J pode esperar (não há "deadlock"): FIM
12. Não: Fazer K←K' e voltar para o item 5.