Sistemas operacionais: Conceitos e Mecanismos

Capítulo 10: Coordenação entre tarefas

Integrantes do grupo: Jefferson Botitano e Leonardo Ludvig
Professor: Arliones Stevert Hoeller Junior
Disciplina: Sistemas operacionais

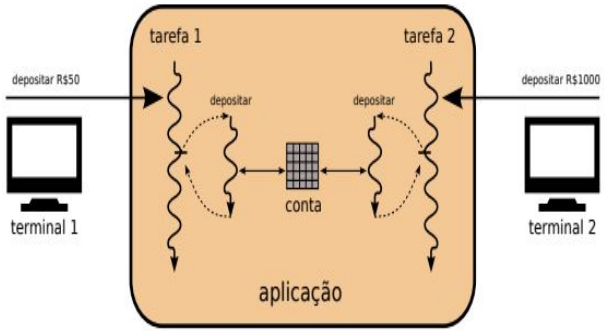
INSTITUTO FEDERAL
Santa Catarina

Campus São José

O problema da concorrência

O problema da concorrência ocorre duas ou mais tarefas acessam ao mesmo tempo um recurso compartilhado,por causa desse conflito ocorre problemas do tipo consistência dos dados ou do estado do recurso acessado.

- Uma aplicação concorrente: A figura abaixo representa um exemplo de acessos concorrentes a variáveis compartilhadas,onde dois terminais diferentes tentam realizar a operação de depósito em uma conta sendo que os dois terminais,sendo que as duas tarefas dos terminais tentam acessar um mesmo recurso que é a conta.



- Condições de disputa: Podem ocorrer em sistemas onde várias tarefas acessam de forma concorrente recursos compartilhados,sob certas condições. Estas disputas podem fazer com que erros no código fonte não apareçam e sim apenas durante a execução do código,no entanto até mesmo durante a execução não podem ser identificados caso ocorra erros mas apenas quando ocorrem entrelaçamentos de processos ocorrem.Os problemas de condições de disputa só se manifestam com acessos simultâneos aos dados e difícil detecção na depuração do código.

- Condições de Bernstein:As condições de disputa entre tarefas paralelas é formalizada pelas condições de bernstein.Por exemplo: temos duas tarefas t1 e t2, R(ti) sendo o conjunto de variáveis lidas por ti e W(ti) o conjunto de variáveis escritas por ti,então pelas condições de bernstein elas podem executar de acordo com as seguintes regras sem risco de condição de disputa:

$$t_1 \parallel t_2 \iff \begin{cases} \mathcal{R}(t_1) \cap \mathcal{W}(t_2) = \varnothing & (t_1 \text{ não lê as variáveis escritas por } t_2) \\ \mathcal{R}(t_2) \cap \mathcal{W}(t_1) = \varnothing & (t_2 \text{ não lê as variáveis escritas por } t_1) \\ \mathcal{W}(t_1) \cap \mathcal{W}(t_2) = \varnothing & (t_1 \text{ e } t_2 \text{ não escrevem nas mesmas variáveis}) \end{cases}$$

- Seções críticas:As seções críticas delimita os trechos de códigos que podem ocorrer acesso compartilhado entre tarefas e que podem cair em condições de disputa.

Exclusão mútua

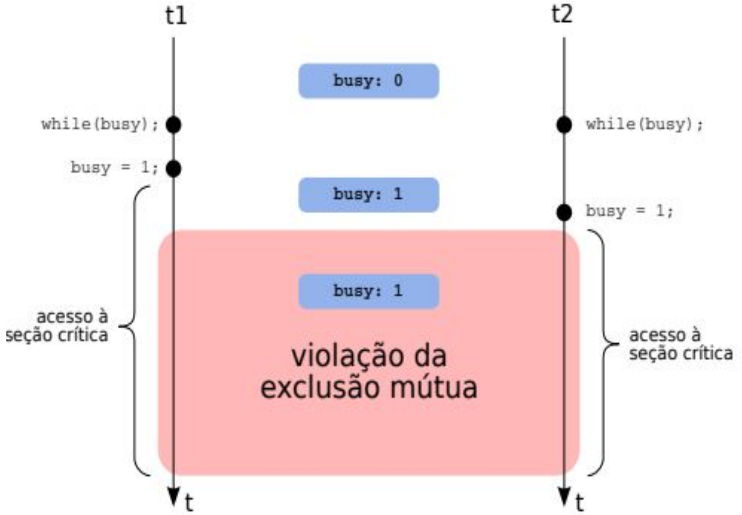
INSTITUTO FEDERAL
Santa Catarina

Para garantir que não haja conflito de entrelaçamento entre seções críticas, é necessário realizar a exclusão mútua, a qual seria bloquear o acesso de múltiplas tarefas a mesma seção crítica definindo o começo e fim de cada seção, utilizando das primitivas enter (bloqueia o acesso após a entrada de uma tarefa) e leave (libera o acesso para outra tarefa). Para que haja sucesso ao realizar a exclusão mútua alguns critérios tem que levados em consideração, somente uma tarefa pode acessar uma seção crítica por vez, a tarefa que está em espera tem que ter o seu acesso garantido (sem inanição), a decisão de uso das seções críticas deve depender somente das tarefas que estão tentando entrar na mesma e não devem depender de fatores físicos sendo necessário uma solução puramente lógica. Algumas soluções podem ser vistas abaixo:

- **Inibição de interrupções:** Uma solução simples que consiste em impedir as trocas de contexto e interrupções ao entrar em uma seção crítica, e as reativa ao sair da mesma. Dificilmente usada pois só funciona em sistemas monoprocessados e possui diversos problemas.
- **Alternância de uso:** Nesta solução é atribuído uma variável turno na qual cada tarefa aguarda seu turno de usar a seção crítica em uma sequência circular $t0 \rightarrow t1 \rightarrow t2 \rightarrow \cdots \rightarrow tn-1 \rightarrow t0$ garantindo a independência de fatores externos porém não cumprindo com os outros critérios.

Exclusão mútua

- **A solução trivial:** Utiliza uma variável busy para indicar se a seção crítica está ocupada ou disponível.



Porém não funciona pois a atribuição e teste da variável busy são realizados em momentos distintos, ocorrendo erro cajo haja troca de contexto e consumindo muito processador com o teste contínuo da variável busy

- **O algoritmo de Peterson:** Para realizar uma exclusão mútua entre duas tarefas ao entrar em uma seção crítica foi desenvolvido por Gary Peterson um algoritmo que possibilitou uma solução correta, sendo usado para tarefas n<2.

- **Operações atômicas :** Para contornar o problema do uso da variável busy, foram criadas instruções em código de máquina que permitem testar e atribuir valor a uma variável “ao mesmo tempo” sem que haja a possibilidade de troca de contexto, ou seja, de forma atômica.

Problemas

INSTITUTO FEDERAL
Santa Catarina

O acesso concorrente de muitas tarefas ao mesmo recurso pode provocar problemas de chamadas disputa,sendo assim uma forma de evitar estes tipos de problemas é forçar o acesso aos recurso por exclusão mútua mas mesmo assim essa solução sofre de problemas que não permite o uso em larga escala em aplicações.

- Ineficiência:Tarefas que aguardam o acesso em uma seção crítica ficam realizando um teste de condição continuamente, impactando no consumo de tempo de processador para solucionar isto seria adequado suspender essas tarefas até a seção crítica libere por solicitação.

- Injustiça:Quando não existe garantia de ordem no acesso à seção crítica que depende da duração do quantum e da política de escalonamento, uma tarefa pode entrar e sair da seção crítica várias vezes,antes que outras tarefas consigam acessá-la.

- Dependência:Tarefas que querem acessar a seção crítica podem ter acesso recusado por tarefas que não têm interesse na seção crítica naquele instante de tempo.

2. Explique o que são condições de disputa, mostrando um exemplo real.

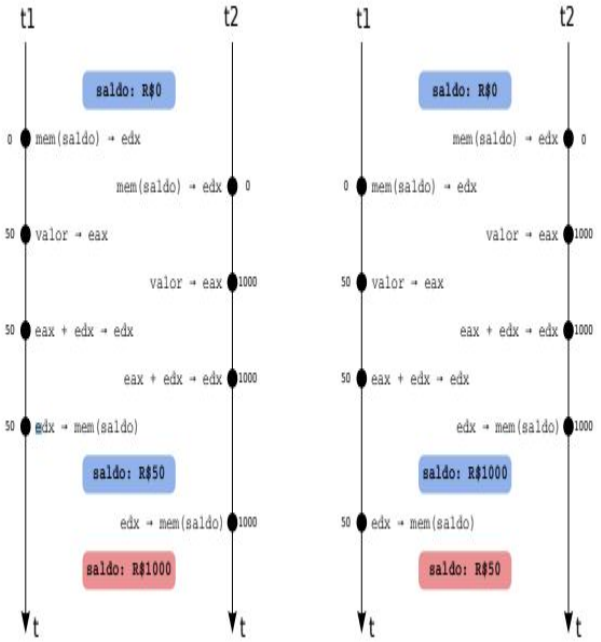
INSTITUTO FEDERAL
Santa Catarina

Condições de disputa em sistemas onde várias tarefas acessam de forma concorrente recursos compartilhados,sob certas condições,sendo assim podem gerar erros dinâmicos de difícil detecção.

Um exemplo onde duas tarefas t1 e t2 estão executando ao mesmo tempo, onde que estas duas tarefas querem chamar a função depositar em seus terminais.

1. t1 acessa memória de aplicação e guarda na memória o valor da variável saldo em edx em sua memória.
2. t2 acessa memória de aplicação e guarda na memória o valor da variável saldo em edx em sua memória.
3. t1 adiciona valor na variável eax.
4. t2 adiciona valor na variável eax.
5. t1 realiza soma de eax(valor de depósito) e edx(valor de saldo) resultando em 50 e armazenando em edx.
6. t2 realiza soma de eax(valor de depósito) e edx(valor de saldo) resultando em 1000 e armazenando em edx.
7. t1 envia para memória de aplicação valor que está na variável edx que é 50, sendo assim memória de aplicação possui variável saldo igual a 50.
8. t2 envia para memória de aplicação valor que está na variável edx que é 50, sendo assim memória de aplicação possui variável saldo igual a 1000.

   Como as tarefas ocorreram simultaneamente a tarefa 1 foi ignorado,pois o resultado deveria ser 1050,pois os valores dos registradores das tarefas não estavam sincronizados com o valor de memória da aplicação



Campus São José

3. Sobre as afirmações a seguir, relativas aos mecanismos de coordenação, indique quais são incorretas, justificando sua resposta:

INSTITUTO FEDERAL
Santa Catarina

(a) A estratégia de inibir interrupções para evitar condições de disputa funciona em sistemas multi-processados.
FALSO: esse tipo de estratégia funciona somente em sistemas monoprocessados,pelo fato que em multi-processados duas tarefas concorrentes podem executar simultaneamente em processadores separados à mesma seção crítica do código fonte.
(b) Os mecanismos de controle de entrada nas regiões críticas provêem exclusão mútua no acesso às mesmas.
(c) Os algoritmos de busy-wait se baseiam no teste contínuo de uma condição.
(d) Condições de disputa ocorrem devido às diferenças de velocidade na execução dos processos.
FALSO: as condições de disputa ocorre em sistemas onde várias tarefas acessam de forma concorrente recursos compartilhados.
(e) Condições de disputa ocorrem quando dois processos tentam executar o mesmo código ao mesmo tempo.
FALSO:se dois processos tentassem executar a mesma seção crítica do código teria condições de disputa.
(f) Instruções do tipo Test&Set Lock devem ser implementadas pelo núcleo do SO.
FALSO:as instruções deste tipo são executadas pelo processador do sistema.
(g) O algoritmo de Peterson garante justiça no acesso à região crítica.
(h) Os algoritmos com estratégia busy-wait otimizam o uso da CPU do sistema.
FALSO: o algoritmo faz o contrário ele faz a CPU gastar mais recursos de processamento.
(i) Uma forma eficiente de resolver os problemas de condição de disputa é introduzir pequenos atrasos nos processos envolvidos.
FALSO: os atrasos não resolvem os problemas de disputa,sendo que o atraso não influencia no acesso aos dados simultâneos.

4. Explique o que é espera ocupada e por que os mecanismos que empregam essa técnica são considerados ineficientes.

INSTITUTO FEDERAL
Santa Catarina

É o fenômeno que ocorre ao se realizar um teste de uma variável diversas de vezes de forma sequencial até que alguma condição se satisfaça para cessar o teste contínuo, levando a um alto consumo do processador.

5. Em que circunstâncias o uso de espera ocupada é inevitável?

Em operações que ocorrem rapidamente e que troca de contexto e suspender o processo não é viável como por exemplo inicializar um conversor analógico digital.

INSTITUTO FEDERAL
Santa Catarina

6. Considere ocupado uma variável inteira compartilhada entre dois processos A e B (inicialmente, ocupado = 0). Sendo que ambos os processos executam o trecho de programa abaixo, explique em que situação A e B poderiam entrar simultaneamente nas suas respectivas regiões críticas.

```
while (true) {
        regiao_nao_critica();
        while (ocupado) {};
        ocupado = 1;
        regiao_critica();
        ocupado = 0;
}
```

O trecho tem semelhança com o código da variável busy, com A e B executando ao mesmo tempo ambos poderiam ler a região como desocupada e entrar nela as duas antes que haja uma leitura novamente.