Sistemas Operacionais

Othon Oliveira

Fatec – Faculdade de Informática — PE

1 de março de 2016

Espaço de endereçamento

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Espaço de endereçamento

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Definição

Isso é quase uma definição de processo, exceto pelo espaço de endereçamento

Espaço de endereçamento

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Definição

Isso é quase uma definição de processo, exceto pelo espaço de endereçamento

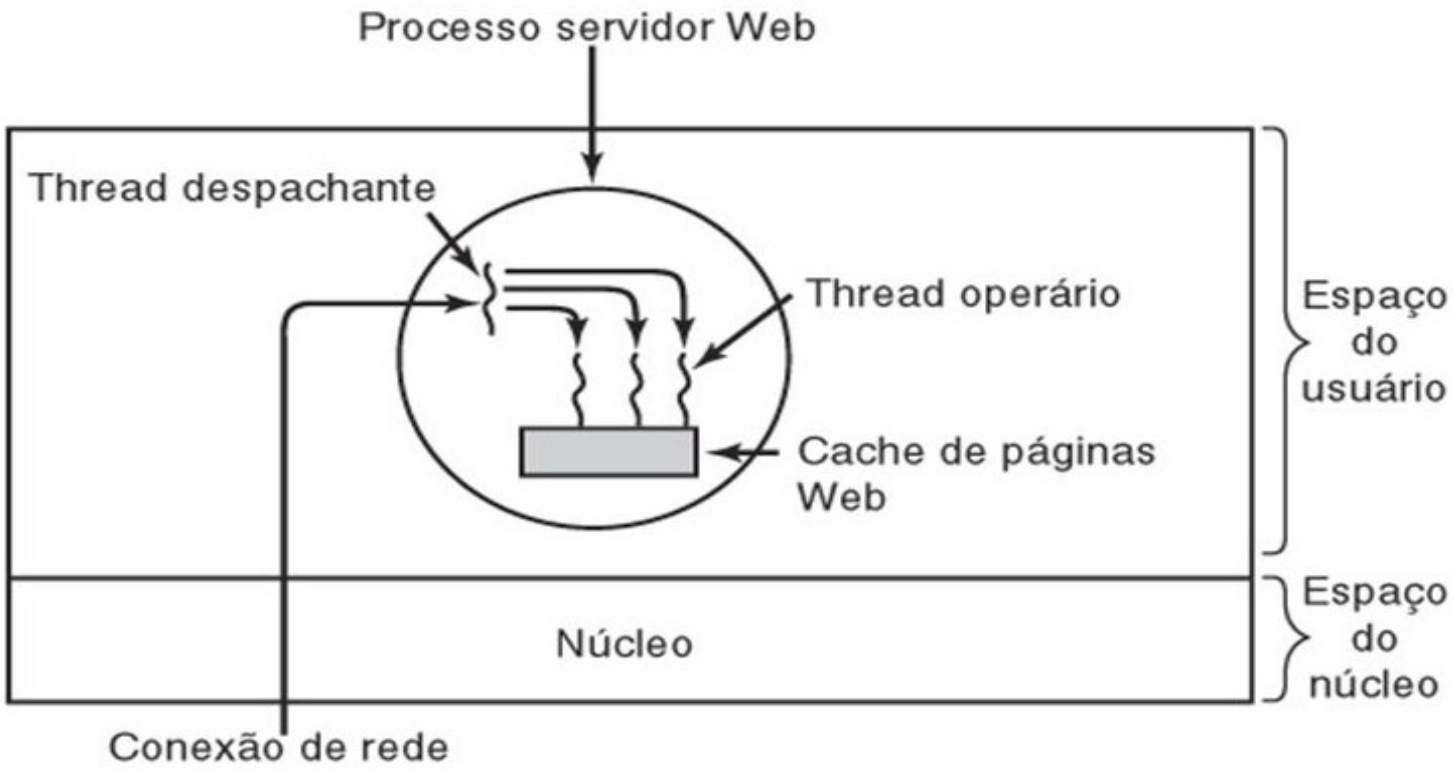
Contudo é frequente querer ter múltiplos threads em um único espaço de endereçamento executando em quase-paralelamente, como se fossem processos separados.

Exemplo

Um servidor Web

Um modo de organizar o servidor Web é mostrado na figura acima. Na figura, um thread, o **despachante**, lê as requisições que chegam à rede. Depois de examinar a requisição, ele escolhe um thread **operário** ocioso (isto é bloqueado) e entrega-lhe a requisição, possivelmente com um ponteiro associado a mensagem e uma palavra especial para cada thread. Na prática o despachante "acorda" o operário que está descansando, tirando-o do estado bloqueado e colocando-o no estado pronto.

Um servidor Web



Um servidor web com múltiplos threads

Thread Despachante

C,C++

```
1 while(true){\\
2   // Uma implementacao da figura anterior
3   // um bloco, em C:
4   get_next_request(&buf);\\
5   handoff_work(&buf);\\
6 }
```

Thread Operário

C,C++

```
1  while(true){
2     wait_for_work(&buf);
3     look_for_page_in_cache(&buf, &page);
4     if(page_not_in_cache(&page))
          read_page_from_disk(&buf, &page);
5     return_page(&page);
6  }
```

Thread Operário

C,C++

```
1  while(true){
2    wait_for_work(&buf);
3    look_for_page_in_cache(&buf, &page);
4    if(page_not_in_cache(&page))
         read_page_from_disk(&buf, &page);
5    return_page(&page);
6  }
```

Pergunta

Este exemplo é multithreads ou monothread ??

Explicando o código

Um servidor Web

Esse modelo permite que o servidor seja escrito como uma coleção de threads sequenciais. O programa do despachante consiste num laço infinito para obter requisições de trabalho e entregá-las a um operário. Cada código de operário consiste em um laço infinito que acata uma requisição de um despachante e verifica se a página está presente na cache de páginas Web. Se estiver, entrega-a ao cliente e bloqueia esperando uma nova requisição.

Explicando o código

Um servidor Web

Esse modelo permite que o servidor seja escrito como uma coleção de threads sequenciais. O programa do despachante consiste num laço infinito para obter requisições de trabalho e entregá-las a um operário. Cada código de operário consiste em um laço infinito que acata uma requisição de um despachante e verifica se a página está presente na cache de páginas Web. Se estiver, entrega-a ao cliente e bloqueia esperando uma nova requisição.

monothread

Porém o código anterior é monothread

Espaço do Usuário

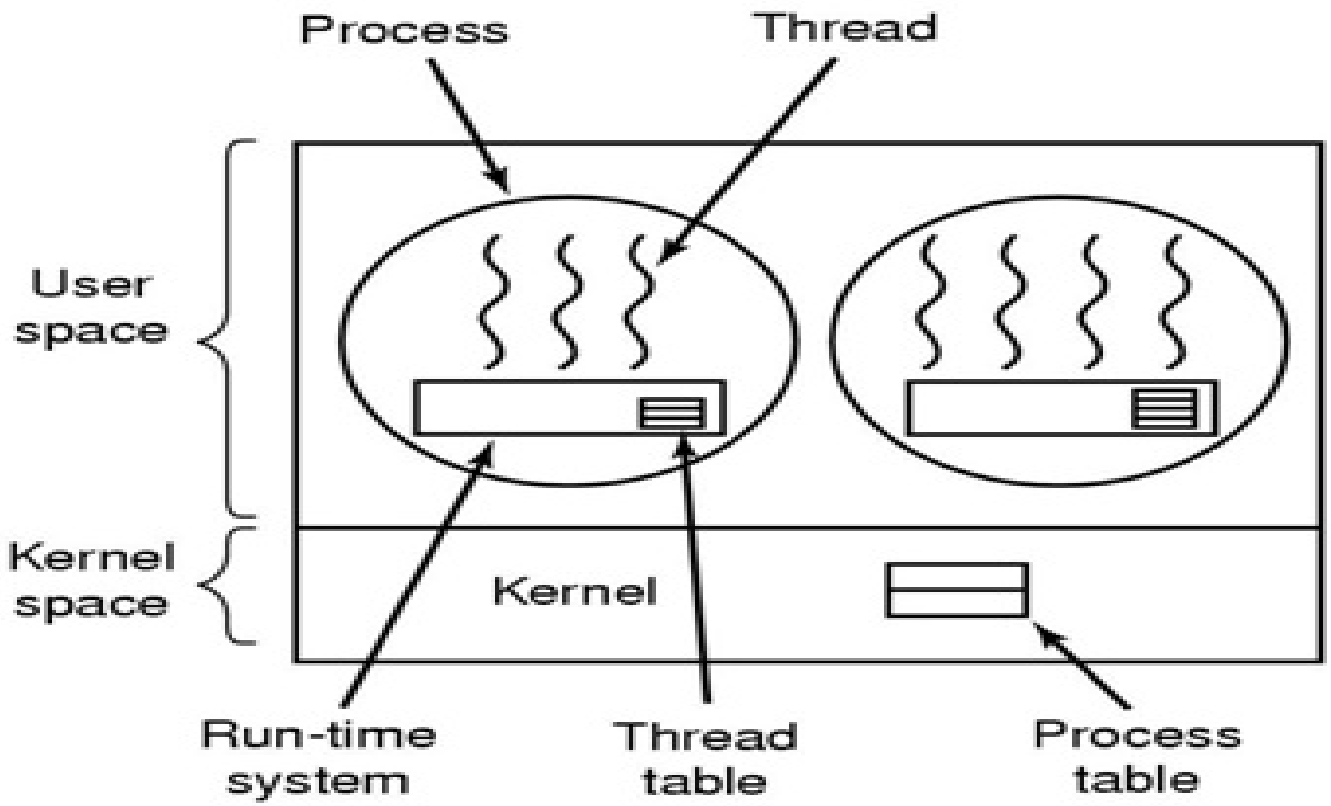
Há muitos de implementar um pacote de threads: no espaço do usuário e no núcleo. Essa escolha é um pouco controversa. Exite também a implementação híbrida.

Thread de usuário

O primeiro método é inserir o pacote de threads totalmente dentro do espaço do usuário (threads de usuário). O núcleo não é inforamdo sobre eles. O que compete ao núcleo é gerir os processos.

Thread de usuário

Thread em modo usuário



Espaço do kernel

Considere que o núcleo saiba sobre os threads e os gerencie. Não é necessário um sistema de supervisor como mostrado na figura anterior

Espaço do kernel

Considere que o núcleo saiba sobre os threads e os gerencie. Não é necessário um sistema de supervisor como mostrado na figura anterior

Definição

Não há também uma tabela de threads em cada processo. Em vez disso o núcleo tem uma tabela de threads que acompanha todos os threads do sistema.

Espaço do kernel

Considere que o núcleo saiba sobre os threads e os gerencie. Não é necessário um sistema de supervisor como mostrado na figura anterior

Definição

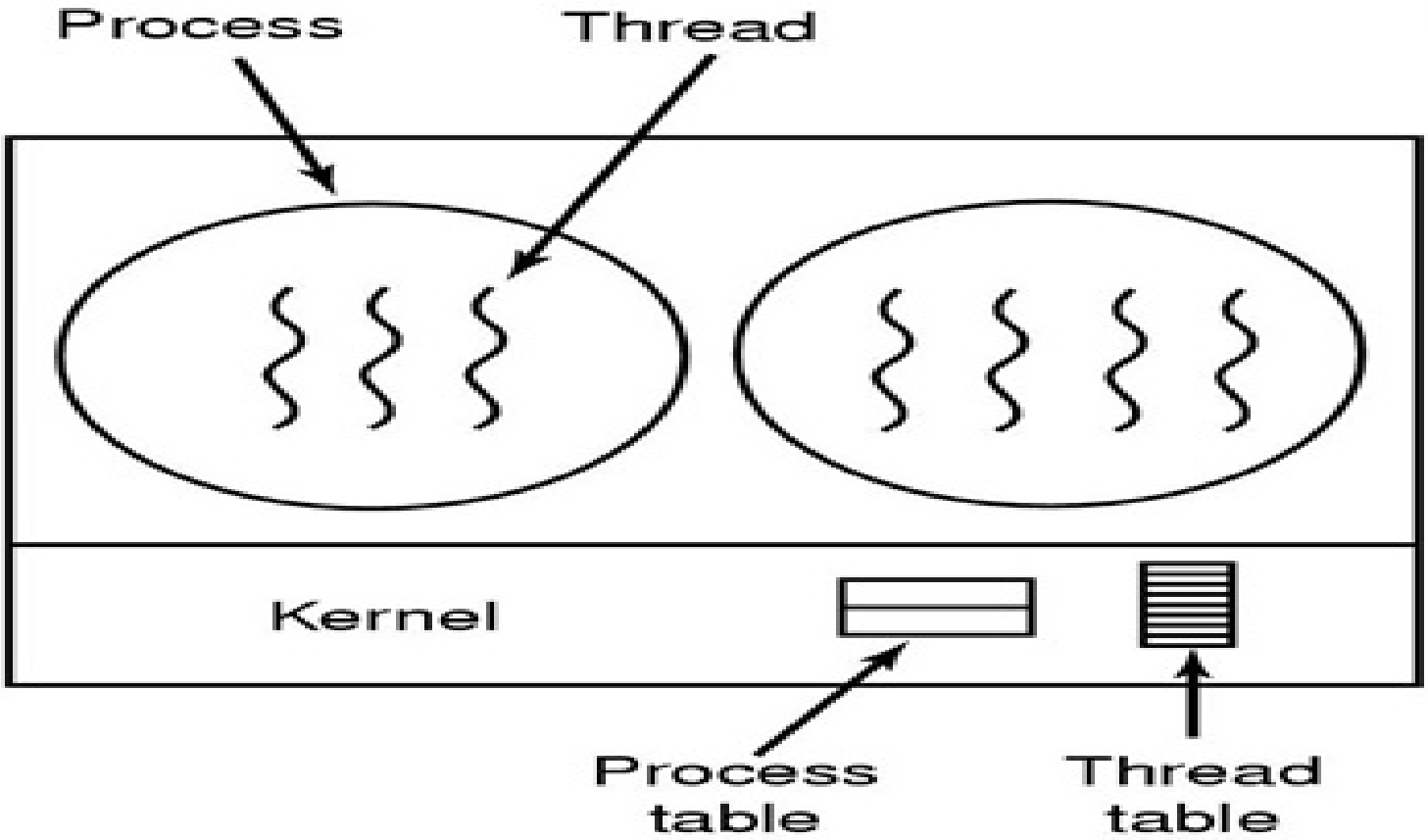
Não há também uma tabela de threads em cada processo. Em vez disso o núcleo tem uma tabela de threads que acompanha todos os threads do sistema.

Definição

Quando um thread que criar um novo thread ou destruir um já existente, faz uma chamada ao núcleo, que realiza então a criação ou a destruição atualizando a tabela de threads do núcleo.

Threads de núcleo – kernel

Thread em modo kernel



Threads híbridas

Vários modos de tentar combinar as vantagens dos threads de usuário com os threads de núcleo têm sido investigados.

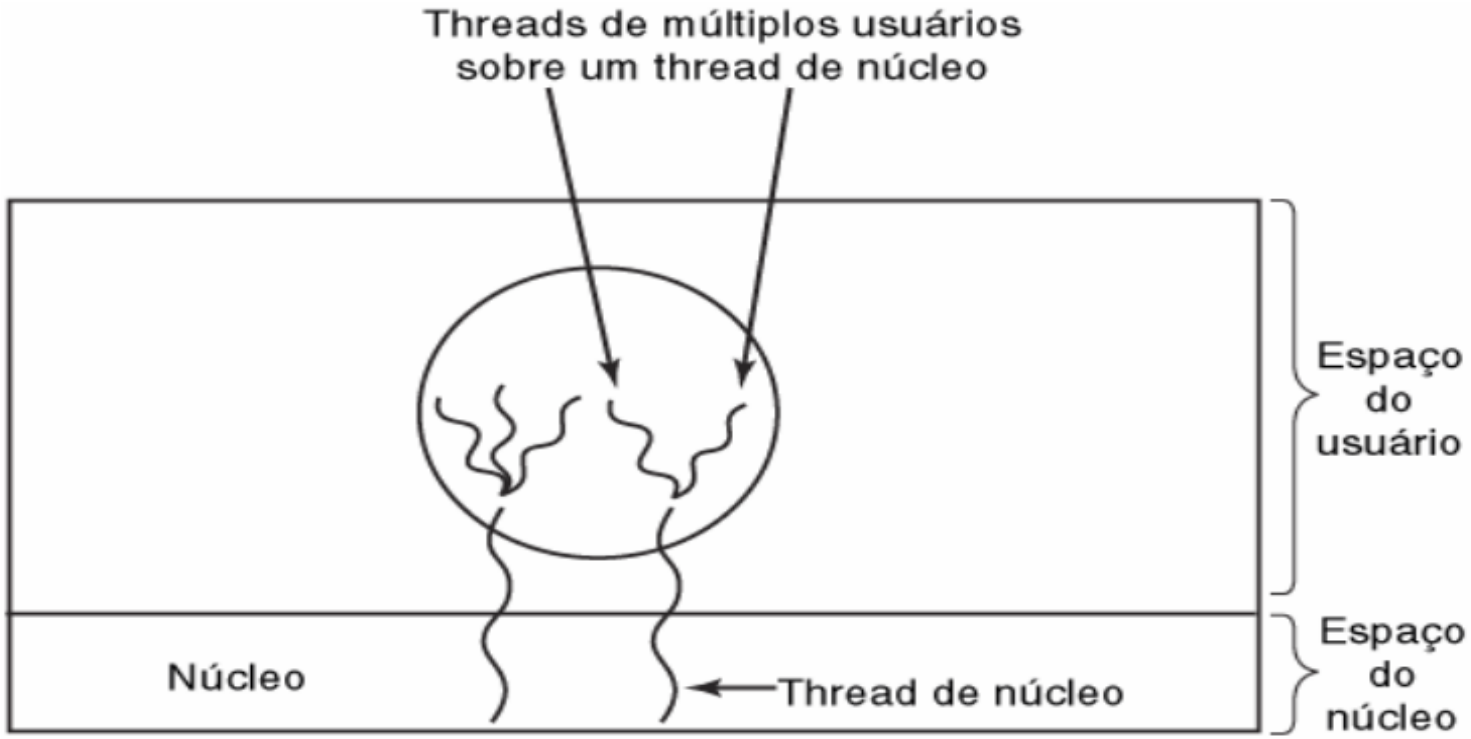
Threads híbridas

Vários modos de tentar combinar as vantagens dos threads de usuário com os threads de núcleo têm sido investigados.

Multiplexar threads

Um deles é usar threads de núcleo e então Multiplexar threads de usuários sobre algum ou todos os threads de núcleo.

Threads híbridas



Multiplexação de threads de usuário sobre threads de núcleo

A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

Tipos

- Condição de disputa

A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

Tipos

- Condição de disputa
- Regiões críticas

A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

Tipos

- Condição de disputa
- Regiões críticas
- Exclusão mútua com espera ociosa

A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

Tipos

- Condição de disputa
- Regiões críticas
- Exclusão mútua com espera ociosa
- Dormir e acordar

A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

Tipos

- Condição de disputa
- Regiões críticas
- Exclusão mútua com espera ociosa
- Dormir e acordar
- Semáforos

A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

Tipos

- Condição de disputa
- Regiões críticas
- Exclusão mútua com espera ociosa
- Dormir e acordar
- Semáforos
- Mutexes

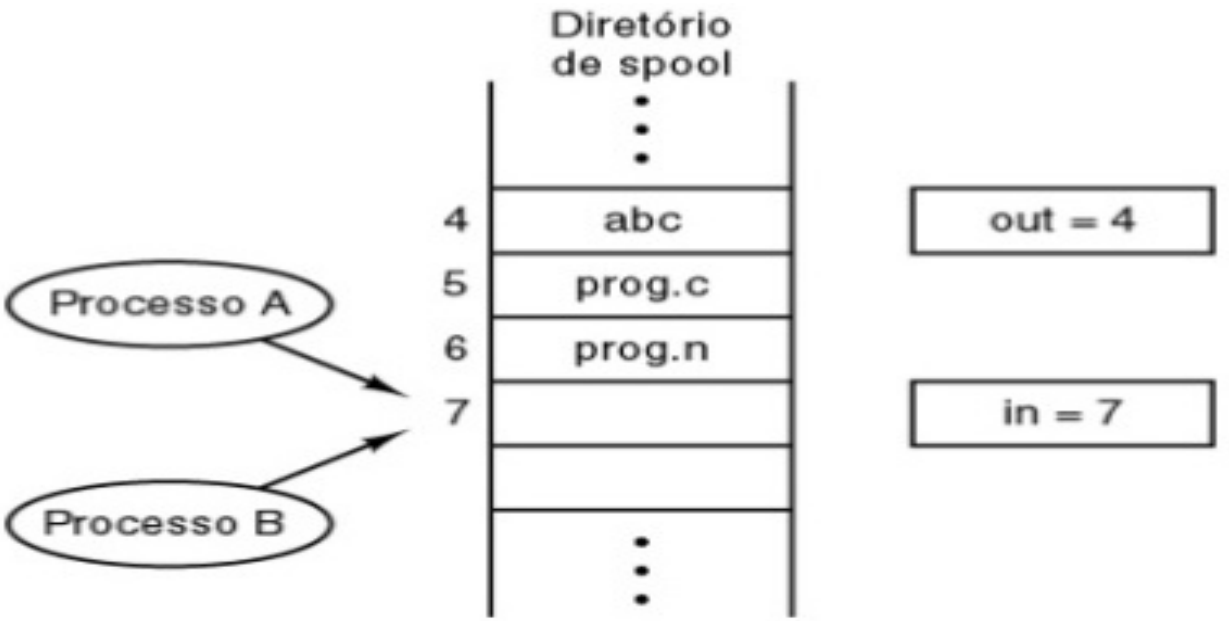
A Comunicação entre processos

Em um pipeline do interpretador de comandos, a saída do primeiro processo deve ser passada para o segundo processo, e isso prossegue até o fim da linha de comando. Assim há uma necessidade de comunicação entre processos

Tipos

- Condição de disputa
- Regiões críticas
- Exclusão mútua com espera ociosa
- Dormir e acordar
- Semáforos
- Mutexes
- Monitores

Em alguns sistemas operacionais, processos que trabalham juntos podem compartilhar algum armazenamento comum. A seguir dois processos tentar escrever arquivos na fila de impressão



O que fazer para evitar condições de disputa? Exclusão mútua

A resposta para evitar disputa em memória compartilhada é evitar de alguma maneira que processos leiam e escrevam ao mesmo tempo na memória compartilhada.

Região de Exclusão

Em outras palavras uma maneira de assegurar que outros processos sejam impedidos de usar uma variável ou um arquivo compartilhado que já estiver em uso por outro processo é criar uma Região de Exclusão.

Isso é suficiente??

Embora essa solução impeça as condições de disputa, isso não é suficiente Precisamos satisfazer quatro condições

Região Críticas

- Nunca dois processos podem estar simultaneamente em suas regiões críticas.

Isso é suficiente??

Embora essa solução impeça as condições de disputa, isso não é suficiente Precisamos satisfazer quatro condições

Região Críticas

- Nunca dois processos podem estar simultaneamente em suas regiões críticas.
- Nada pode ser afirmado sobre a velocidade ou sobre o número de CPUs.

Isso é suficiente??

Embora essa solução impeça as condições de disputa, isso não é suficiente Precisamos satisfazer quatro condições

Região Críticas

- Nunca dois processos podem estar simultaneamente em suas regiões críticas.
- Nada pode ser afirmado sobre a velocidade ou sobre o número de CPUs.
- Nenhum processo executando fora da região crítica pode bloquear outros processos.

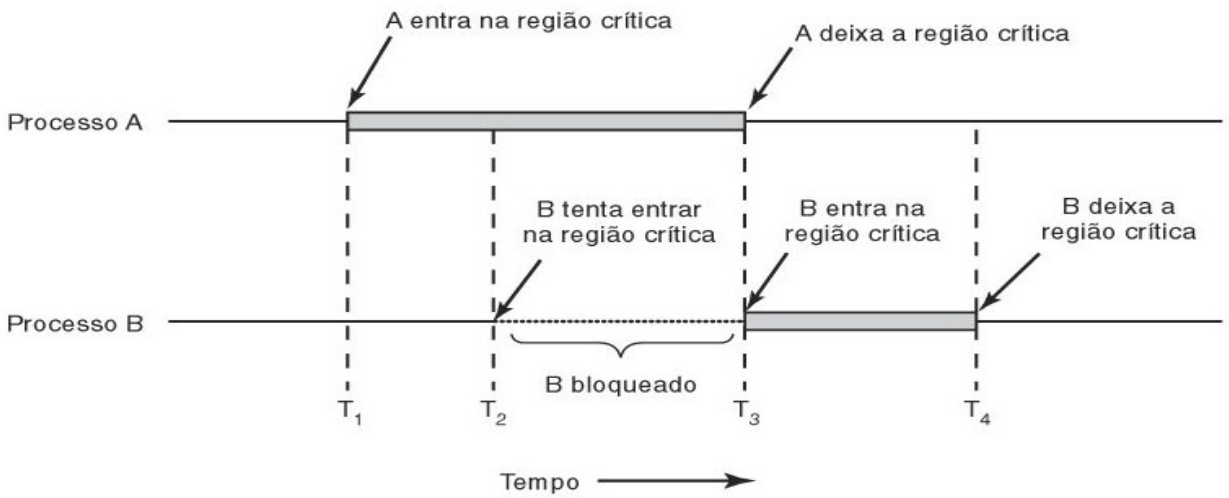
Isso é suficiente??

Embora essa solução impeça as condições de disputa, isso não é suficiente Precisamos satisfazer quatro condições

Região Críticas

- Nunca dois processos podem estar simultaneamente em suas regiões críticas.
- Nada pode ser afirmado sobre a velocidade ou sobre o número de CPUs.
- Nenhum processo executando fora da região crítica pode bloquear outros processos.
- Nenhum processo deve esperar eternamente para entrar em sua região crítica.

Exclusão mútua usando Regiões Críticas



Exclusão mútua usando regiões críticas

Desabilitando interrupções

Estudaremos várias alternativas para realizar Exclusão Mútua de modo que nenhum outro processo invada a região de memória compartilhada.

Desabilitando interrupções da CPU

A solução simples é aquela que cada processo desabilita todas as interrupções logo depois de entrar em sua região crítica e reabilita-as antes de sair dela.

Desabilitando interrupções

Estudaremos várias alternativas para realizar Exclusão Mútua de modo que nenhum outro processo invada a região de memória compartilhada.

Desabilitando interrupções da CPU

A solução simples é aquela que cada processo desabilita todas as interrupções logo depois de entrar em sua região crítica e reabilita-as antes de sair dela.

Desabilitando interrupções da CPU

Pode ser necessário o próprio núcleo desabilitar as interrupções, para umas poucas instruções enquanto estiver alterando Variáveis ou listas.

Variáveis de impedimento(lock variables)

Uma solução por software

Variáveis de impedimento(lock variables)

Uma solução por software

Definição

Para entrar em sua região crítica, um processo testa antes se a variável *lock* é zero (0) o processo altera altera para 1 e entra na região crítica. Infelizmente essa técnica apresenta a mesma falha que vimos no diretório de spool.

Caso outro processo ao mesmo tempo leia a variável *lock* alterando para 1 haverão 2 processos em suas Regiões Críticas.

Alternância obrigatória

Pesquisa

- Pesquisar sobre Alternância obrigatória

Alternância obrigatória

Pesquisa

- Pesquisar sobre Alternância obrigatória
- Solução de Paterson

Alternância obrigatória

Pesquisa

- Pesquisar sobre Alternância obrigatória
- Solução de Paterson
- A instrução TSL

Solução de Paterson

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Definição

Isso é quase uma definição de processo, exceto pelo espaço de endereçamento

Contudo é frequente querer ter múltiplos threads em um único espaço de endereçamento executando em quase-paralelamente, como se fossem processos separados.

Alternância obrigatória

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Definição

Isso é quase uma definição de processo, exceto pelo espaço de endereçamento

Contudo é frequente querer ter múltiplos threads em um único espaço de endereçamento executando em quase-paralelamente, como se fossem processos separados.

A instrução TSL

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Definição

Isso é quase uma definição de processo, exceto pelo espaço de endereçamento

Contudo é frequente querer ter múltiplos threads em um único espaço de endereçamento executando em quase-paralelamente, como se fossem processos separados.

Problema da inversão de prioridade

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Definição

Isso é quase uma definição de processo, exceto pelo espaço de endereçamento

Contudo é frequente querer ter múltiplos threads em um único espaço de endereçamento executando em quase-paralelamente, como se fossem processos separados.

O Problema produtor-consumidor

Em sistemas tradicionais, cada processo tem um espaço de endereçamento e um único Thread (fluxo) de controle.

Definição

Isso é quase uma definição de processo, exceto pelo espaço de endereçamento

Contudo é frequente querer ter múltiplos threads em um único espaço de endereçamento executando em quase-paralelamente, como se fossem processos separados.