Sistemas Operacionais

Othon Oliveira

Fatec – Faculdade de Informática — PE

14 de março de 2016

Espera ociosa

A Solução de Paterson com base na **TSL** está correta, mas ambas apresentam o defeito da **espera ociosa**

Espera ociosa

A Solução de Paterson com base na **TSL** está correta, mas ambas apresentam o defeito da **espera ociosa**

A essência

Em essência o que as instruções fazem é: quando quer entrar em sua região crítica um processo verifica se sua entrada é permitida. Se não for, o processo ficará em um loop ocioso esperando até que seja permitida a entrada.

Espera ociosa

A Solução de Paterson com base na **TSL** está correta, mas ambas apresentam o defeito da **espera ociosa**

A essência

Em essência o que as instruções fazem é: quando quer entrar em sua região crítica um processo verifica se sua entrada é permitida. Se não for, o processo ficará em um loop ocioso esperando até que seja permitida a entrada.

Qual o problema dessa solução?

Espera ociosa

A Solução de Paterson com base na **TSL** está correta, mas ambas apresentam o defeito da **espera ociosa**

A essência

Em essência o que as instruções fazem é: quando quer entrar em sua região crítica um processo verifica se sua entrada é permitida. Se não for, o processo ficará em um loop ocioso esperando até que seja permitida a entrada.

Qual o problema dessa solução?

CPU

Não só gasta tempo da CPU como pode, pode gerar outro problema.

A Inversão de prioridade

As prioridades podem se inverter?

Considere um computador com dois processos: H, com alta prioridade, e L, com baixa prioridade. As regras de escalonamento são tais que H é executado sempre que estiver no estado pronto. Em certo momento, com L em sua região crítica, H torna-se pronto para execuar (ex: terminou uma E/S).

A Inversão de prioridade

As prioridades podem se inverter?

Considere um computador com dois processos: H, com alta prioridade, e L, com baixa prioridade. As regras de escalonamento são tais que H é executado sempre que estiver no estado pronto. Em certo momento, com L em sua região crítica, H torna-se pronto para execuar (ex: terminou uma E/S).

O loop infinito

Agora H inicia uma espera ociosa, como L nunca é escalonado enquanto H está executando, L nunca tem a oportunidade de deixar sua região crítica e, assim H entra em um laço infinito. Essa situação é chamada de **problema da inversão de prioridade**.

"Sleep" ou "Wakeup"

O bloqueio em vez da Espera Ociosa

Quando não é permitido a um processo não entrar em sua região crítica, ele pode ser posta a dormir. Com o par de intruções **sleep** e **wakeup**.

"Sleep" ou "Wakeup"

O bloqueio em vez da Espera Ociosa

Quando não é permitido a um processo não entrar em sua região crítica, ele pode ser posta a dormir. Com o par de intruções **sleep** e **wakeup**.

Sleep

Sleep é uma chamada ao sistema que faz com que quem a chama durma, isto é, fica suspenso até que outro processo o desperte.

"Sleep" ou "Wakeup"

O bloqueio em vez da Espera Ociosa

Quando não é permitido a um processo não entrar em sua região crítica, ele pode ser posta a dormir. Com o par de intruções **sleep** e **wakeup**.

Sleep

Sleep é uma chamada ao sistema que faz com que quem a chama durma, isto é, fica suspenso até que outro processo o desperte.

Wakeup

Wakeup tem um parâmetro, qual processo a ser despertado e outro parâmetro um endereço de memória usado para comparar wakeups e seus respectivos sleeps.

Buffer limitado

Compartilhando buffer

Dois processos compartilham um buffer comum de tamanho fixo.

Buffer limitado

Compartilhando buffer

Dois processos compartilham um buffer comum de tamanho fixo. Um deles o **produtor**, põe informação dentro do buffer e o outro,

Buffer limitado

Compartilhando buffer

Dois processos compartilham um buffer comum de tamanho fixo. Um deles o **produtor**, põe informação dentro do buffer e o outro, o **consumidor** a retira informações desse buffer.

Buffer limitado

Compartilhando buffer

Dois processos compartilham um buffer comum de tamanho fixo. Um deles o **produtor**, põe informação dentro do buffer e o outro, o **consumidor** a retira informações desse buffer.

É possível generalizar para M produtores e N consumidores.

A solução: produtor – consumidor

Quando produzir?

O produtor quer colocar um novo item em um buffer cheio ?

A solução: produtor – consumidor

Quando produzir?

O produtor quer colocar um novo item em um buffer cheio ?

Quando consumir?

Ou o consumidor quer retirar de um buffer vazio ?

A solução: produtor – consumidor

Quando produzir?

O produtor quer colocar um novo item em um buffer cheio ?

Quando consumir?

Ou o consumidor quer retirar de um buffer vazio ?

A solução: uma varável *count*, os processos verificam essa variável antes de consumir ou produzir

O produtor

```
#Define N 100 /* número lugares no buffer */
  int count = 0; /* número de itens no buffer */

void producer(void){
  int item;
  while(true){
    item = produzirItem(); /* gera o próximo item */
    if (count == N) sleep();/* se o buffer estiver cheio vá dormir
 */
     inserirItem(item) /* ponha um item no buffer */
    count = count + 1; /* incrementa o contador de itens */
    se (count == 1) /* o buffer está vazio? */
     wakeup(consumer()); /* acorda e chama consumer */
  }
```

O consumidor

```
void consumer(void){
  int intem;

  while(true){
    if (count == 0) sleep();/* se o buffer estiver vazio vá dormir
*/
    item = removeItem(); /* retire um item do buffer */
    count = count - 1; /* decrementa o contador de itens */
    if (count == N-1) /* o buffer está cheio ? */
     wakeup(producer()); /* acorda e chama produtor */
    consumirItem(item); /* imprima o item */
  }
```

O consumidor

```
void consumer(void){
  int intem;

  while(true){
    if (count == 0) sleep();/* se o buffer estiver vazio vá dormir
*/
    item = removeItem(); /* retire um item do buffer */
    count = count - 1; /* decrementa o contador de itens */
    if (count == N-1) /* o buffer está cheio ? */
     wakeup(producer()); /* acorda e chama produtor */
    consumirItem(item); /* imprima o item */
  }
```

Qual o problema do algoritmo consumidor – produtor ??

Um problema na solução produtor – consumidor

O problema do acesso irrestrito da variável *count*

O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0.

Um problema na solução produtor – consumidor

O problema do acesso irrestrito da variável *count*

O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. O escalonador decide parar de executar o consumidor temporariamente e começa a executar o produtor.

Um problema na solução produtor – consumidor

O problema do acesso irrestrito da variável *count*

O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. O escalonador decide parar de executar o consumidor temporariamente e começa a executar o produtor.

E o produtor

O produtor faz inserirItem(item) e incrementa *count* e percebe que seu valor agora é 1, inferindo que o valor era 0.

Um problema na solução produtor – consumidor

O problema do acesso irrestrito da variável *count*

O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. O escalonador decide parar de executar o consumidor temporariamente e começa a executar o produtor.

E o produtor

O produtor faz inserirItem(item) e incrementa *count* e percebe que seu valor agora é 1, inferindo que o valor era 0. Ou seja o consumidor deveria ir dormir, o produtor chama *wakeup* para acordar o consumidor.

Um problema na solução produtor – consumidor

O problema do acesso irrestrito da variável *count*

O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. O escalonador decide parar de executar o consumidor temporariamente e começa a executar o produtor.

E o produtor

O produtor faz inserirItem(item) e incrementa *count* e percebe que seu valor agora é 1, inferindo que o valor era 0. Ou seja o consumidor deveria ir dormir, o produtor chama *wakeup* para acordar o consumidor.

Infelizmente, o consumidor ainda não está adormecido; então, o sinal de acordar é perdido.

Uma solução para o sinal perdido

Perdeu um sinal?

A essência é não perder um sinal de acordar para um processo que (ainda) não adormeceu. O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0.

Uma solução para o sinal perdido

Perdeu um sinal?

A essência é não perder um sinal de acordar para um processo que (ainda) não adormeceu. O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. Se esse sinal não fosse perdido tudo funcionaria, em suma adicionar um controle que um **bit de espera de sinal de acordar**

Uma solução para o sinal perdido

Perdeu um sinal?

A essência é não perder um sinal de acordar para um processo que (ainda) não adormeceu. O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. Se esse sinal não fosse perdido tudo funcionaria, em suma adicionar um controle que um **bit de espera de sinal de acordar**

Um bit de acorda

Quando um sinal de "acordar" for enviado a um processo que ainda não está acordado, esse sinal acionaria um bit **ligado**

Uma solução para o sinal perdido

Perdeu um sinal?

A essência é não perder um sinal de acordar para um processo que (ainda) não adormeceu. O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. Se esse sinal não fosse perdido tudo funcionaria, em suma adicionar um controle que um **bit de espera de sinal de acordar**

Um bit de acorda

Quando um sinal de "acordar" for enviado a um processo que ainda não está acordado, esse sinal acionaria um bit **ligado** Depois, quando o processo tentar dormir, se o bit de espera pelo sinal de acordar estiver ligado, ele será desligado.

Uma solução para o sinal perdido

Perdeu um sinal?

A essência é não perder um sinal de acordar para um processo que (ainda) não adormeceu. O buffer está vazio e o consumidor acabou de ler a variável *count* para verificar se seu valor é 0. Se esse sinal não fosse perdido tudo funcionaria, em suma adicionar um controle que um **bit de espera de sinal de acordar**

Um bit de acorda

Quando um sinal de "acordar" for enviado a um processo que ainda não está acordado, esse sinal acionaria um bit **ligado** Depois, quando o processo tentar dormir, se o bit de espera pelo sinal de acordar estiver ligado, ele será desligado.

O bit de espera de sinal de acordar na verdade é um "cofrinho" que guarda sinais de acordar.