UFMG

CURSO DE ESPECIALIZAÇÃO EM AUTOMAÇÃO INDUSTRIAL

Fundamentos e Aplicações de Sistemas de Automação

Módulo 3: Exclusão Mútua

Ao avaliar desenvolvimentos da matemática, devemos sempre ter em mente que as idéias atrás das notações são de longe a melhor metade

Carl B. Boyer, História da Matemática

Prof: Constantino Seixas Filho

sexta-feira, 25 de março de 2005

1

Seção Crítica

UFMG

- Problema fundamental da Programação Concorrente:
  - As instruções envolvidas no acesso a recursos globais compartilhados são intercaladas pelo escalonador fora do controle do programador
- Ideal: todo acesso a recursos compartilhados realizado de forma atômica

Modelo de Programa:

```
{  ***
   SeçãoNãoCrítica;
   ProtocoloDeEntrada;
   SeçãoCrítica;
   ProtocoloDeSaída;
   ****
}
```

2

Seção Crítica

- **Seção Crítica:**
  Trecho de código de um programa onde as instruções não podem ser intercaladas com a de outro programa
- **A seção crítica é o trecho de um programa que deve ser executado sob exclusão mútua**
- **Exclusão Mútua é a propriedade de que apenas uma unidade de execução (thread) estará na sua seção crítica de cada vez.**
- **Quando a exclusão mútua é provada dizemos que a seção crítica é uma região crítica.**
- **O protocolo de entrada serializa a entrada na seção crítica.**

3

Seção Crítica

- P1) Exclusão mútua
  Duas threads nunca poderão estar executando instruções de suas seções críticas concorrentemente
- P2) Ausência de impasse (*deadlock)*
  O impasse acontece quando nenhuma thread consegue entrar em sua seção crítica. Quando várias threads disputam a entrada na seção crítica, temos que assegurar que pelo uma tenha êxito
- P3) Ausência de inanição (*starvation*)
  O critério de entrada na seção crítica deve ser tal que assegure que nenhuma thread seja excluída da oportunidade de entrar. A inanição também é conhecida como adiamento indefinido
- P4) Acesso facilitado em caso de ausência de contenção
  Se apenas uma thread deseja entrar em sua seção crítica, ela deverá conseguir faze-lo com um mínimo de *overhead*

4

Premissas

- Uma thread nunca pode ficar em *loop* ou interromper sua execução dentro da seção crítica, ou dos protocolos de entrada ou saída. Se isto ocorresse, nenhuma outra thread poderia continuar sua execução, o que traria conseqüências desastrosas para todo o sistema

5

Características de um bom algoritmo

- É desejável que o algoritmo funcione para um número ilimitado de threads e não apenas para duas.
- É desejável que a espera para entrar na seção crítica não seja ativa (espera ocupada), isto é, que o programa não fique executando um loop tentando acessar insistentemente o recurso.
- É desejável que o algoritmo seja simples, constituído de poucas instruções para facilitar o seu uso. Os protocolos de entrada e saída não devem, de preferência, utilizar variáveis que dependam de qual thread deseja entrar na seção crítica, a fim de serem mais facilmente encapsulados em bibliotecas.
- É desejável que o mecanismo utilizado minimize os erros dos programadores. Mecanismos complexos, ou de baixo nível de abstração geralmente induzem os usuários a erros.

6

Família de Soluções

a) Técnicas que assumem a não existência de uma instrução especial, no conjunto de instruções de um processador, capaz de ler e escrever em um registro de memória de forma atômica. Esta instrução é denominada de *test_and_set*

b) Técnicas baseadas na instrução *test_and_set* do processador e portanto independentes do sistema operacional

c) Técnicas baseadas na não preemptibilidade do kernel do sistema operacional.

Os métodos usados na práticas pertencem à categoria c)

7

Pecados Mortais

1. Uso de recurso global sem Exclusão Mútua
2. Espera ocupada

8

Prova formal de teoremas em programação concorrente

- Lógica proposicional
- Lógica temporal
- Demonstração por absurdo
- Loop Invariants
- PIF = Princípio da Indução Finita

9

Demonstração por absurdo

- *Reductio ad Absurdum* ou Prova por contradição criada por Hipócrates de Chios, matemático grego no século VI A.C.
- Começamos por negar uma determinada propriedade e depois chegamos à conclusão de que esta suposição leva a uma contradição com relação aos valores assumidos por variáveis do problema e portanto a propriedade considerada como falsa, é verdadeira
- Tem como base a lei do terceiro excluído: “se uma proposição não pode ser verdadeira então deve ser falsa”
- Exemplo:

  Não existe um número racional maior que 0 que seja o mínimo

  Vamos supor que existe um número racional q maior que 0 que seja o mínimo do conjunto dos números racionais

  Seja r = q/2. r é positivo, é menor que q e é racional.

  Logo q não é o mínimo.

10

Problema

- Num certo país os cavaleiros sempre dizem a verdade e os valetes sempre mentem. Um dia um espião de nome Murdoch entrou no país. O rei prendeu três pessoas. Um é cavaleiro, um é valete e o outro é Murdoch. O espião é o único dos três que se chama Murdoch. Os três prestaram os seguintes depoimentos no tribunal:

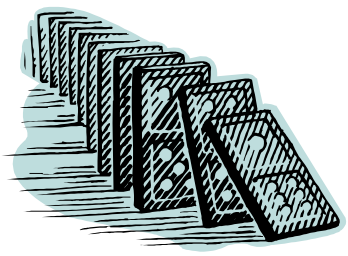
  A: Meu nome é Murdoch
  B: É verdade
  C: Eu sou Murdoch

  Quem é o espião ?

11

Princípio da Indução Finita

- Para provar que uma propriedade se verifica para todos os membros de uma série infinita:
  a) Prova-se a propriedade para n = 1
  b) Admite-se que a propriedade se verifica para n = N e demonstra-se que continua verdadeira para n= N+1
- Princípio Intuitivo:



12

Princípio da Indução Finita

- Exemplo:

  Seja a série: .: 1, 2, 3, 4, 5, 6, ...

  Demonstrar que $\sum_{n=1}^{n} n = S(n) = \frac{n(n+1)}{2}$

  Para n = 1 $\sum_{n=1}^{1} n = S(1) = \frac{1(1+1)}{2} = 1$

  Logo a lei se verifica

  - Vamos admitir que a expressão é V para n = N:

    $\sum_{n=1}^{N} n = S(N) = \frac{N(N+1)}{2}$ logo

    $\sum_{n=1}^{N+1} n = \sum_{n=1}^{N} n + (N+1) = \frac{N(N+1)}{2} + (N+1) = \frac{(N+1)(N+2)}{2} = S(N+1)$

    A quem se deve a descoberta desta expressão ?

    Qual o raciocínio por ele utilizado ?

13

Problemas

Demonstre que:

a) n^3 – n é divisível por 3

b) n^5 – n é divisível por 5

c) A soma dos quadrados dos N números naturais é:

$$\sum_{j=1}^{n} j^2 = \frac{n(n+1)(2n+1)}{6}$$

d) A soma dos cubos dos primeiros N números racionais é um quadrado. Qual o valor deste quadrado ?

14

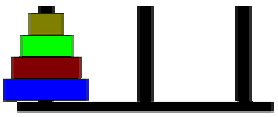
Torres de Hanói

- O Objetivo é mover as peças da coluna da esquerda para a da direita no menor número de passos de forma a que em nenhum instante uma peça maior fique sobre uma menor
- Modelagem do problema:
  Para n=1 necessitamos de apenas um movimento logo:
  T(1) = 1, onde T(n) é o tempo para mover a torre no menor número de passos possível

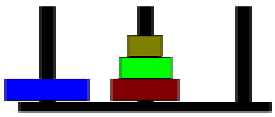
15

Torres de Hanói

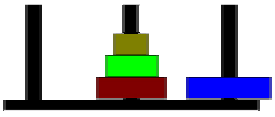
- Parece claro que o algoritmo para solucionar o problema consiste em se mover a torre de N-1 peças para a coluna central, depois mover a peça maior para o destino e finalmente mover a torre intermediária para o destino



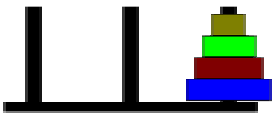
Posição Inicial



1) Mover pilha N-1 para Aux

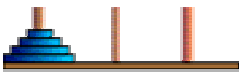


2) Mover Base



3) Mover pilha N-1 para Dest

- Logo: T(N)=2T(N-1)+1



16

Torres de Hanói

- O número de passos necessários para realizar a transferência é:
  .: 1, 3, 7, 15, ...
- Nós poderíamos sugerir uma expressão geral para a(n):
  $A(n) = 2^n - 1$
- Demostre que a expressão anterior é verdadeira usando PIF:

17

Loop Invariant

- **No nosso caso específico, um invariante é uma proposição matemática relacionando valores de variáveis e a posição do contador de instruções de um programa, o qual podemos provar ser verdadeira no início do programa e que continua verdadeira para qualquer transição de instruções do mesmo, ou de qualquer outro programa do sistema**
- **Encontre um invariante no programa seguinte**

18

Loop Invariants

```
// Variáveis globais:
int s1 = 1;
int s2 = 1;

Thread T1;
{   int x = 0;
    loop {
    a1:   s1 = s1 - 1 ;
    b1:   if (s1 >= 0) x= s1;
    c1:   s1 = s1 + 1;
    } // end_loop
}   // ThreadT1

Thread T2;
{   int x = 0;
    loop {
    a2:   s2 = s2 -1;
    b2:   if (s2 >= 0) x= s2;
    c2:   s2 = s2 +1;
    } // end_loop
}   // ThreadT2
```

Procure invariantes neste programa

Proposição de Invariantes:

a) s1 >= 0

b) s1 = 0 v s1 = 1

c) x = 0

d) s2 >= 0

e) s2=0 v s2 = 1

19

Demonstração

Voltando ao exemplo anterior, vamos demonstrar a propriedade:
Q ≡ (s1 = 0 v s1 = 1).
No início a afirmativa é verdadeira:
a1: (s1 = 0 v s1 = 1) s1 = 0 pela semântica da inicialização

Vamos examinar a primeira transição válida:
a1 → b1: s1 = s1 -1;
A expressão se torna: (s1 = -1 v s1 = 0)

A demonstração FALHOU

20

Demonstração – Tentativa 2

Vamos tomar uma propriedade auxiliar:

P ≡ (at(a1) ⊃ s1 = 1) ^ ((at(b1) v at(c1)) ⊃ s1 = 0) é um invariante

Transição	Demonstração
em a1	at(a1) é V, at(b1) é F, at(c1) é F (s1 = 1) é V e (s1 = 0) é F pela semântica da inicialização Usando o raciocínio da lógica proposicional: P ≡ (at(a1) ⊃ s1 = 1) ^ (at(b1) v at(c1) ⊃ s1 = 0) (V ⊃ V) ^ (F v F ⊃ F) V ^ V V
a1 → b1	at(a1) é F, at(b1) é V, at(c1) é F (s1 = 1) é F e (s1 = 0) é V Usando o raciocínio da lógica proposicional: P ≡ (at(a1) ⊃ s1 = 1) ^ (at(b1) v at(c1) ⊃ s1 = 0) (F ⊃ F) ^ (V v F ⊃ V) V ^ V V
b1 → c1	at(a1) é F, at(b1) é F, at(c1) é V (s1 = 1) é F e (s1 = 0) é V Usando o raciocínio da lógica proposicional: P ≡ (at(a1) ⊃ s1 = 1) ^ (at(b1) v at(c1) ⊃ s1 = 0) (F ⊃ F) ^ (F v V ⊃ V) V ^ V V

Transição	Demonstração
c1 → a1	at(a1) é V, at(b1) é F, at(c1) é F (s1 = 1) é V e (s1 = 0) é F Usando o raciocínio da lógica proposicional: P ≡ (at(a1) ⊃ s1 = 1) ^ (at(b1) v at(c1) ⊃ s1 = 0) (V ⊃ V) ^ (F v F ⊃ F) V ^ V V
a2 → b2	O invariante não é afetado
b2 → c2	O invariante não é afetado
c2 → a2	O invariante não é afetado

21

Demonstração – Tentativa 2

O invariante:

P ≡ (at(a1) ⊃ s1 = 1) ^ ((at(b1) v at(c1)) ⊃ s1 = 0)

Foi demonstrado utilizando PIF

Ora, como o programa tem três instruções podemos afirmar:

Q = at(a1) V at(b1) V at(c1)

Das duas afirmativas deduzimos:

R = (s1 =1) V (s1 = 0)

c.q.d.

22

Exercícios

a) Complete as tabelas verdades abaixo:

a	b	a v b	a ^ b	a ⊕ b	a ⊃ b	a ≡ b	~a	~b	~a^b	~b^a	(~a^b) v (~b^a)	a ⊕ b = (~a^b) v (~b^a)
V	V											
V	F											
F	V											
F	F											

b) Demonstre que:

Se ((a ⊃ b) ^ a) ⊃ b (*modus ponens*)

23

Exclusão Mútua sem instruções especiais – Tentativa 1

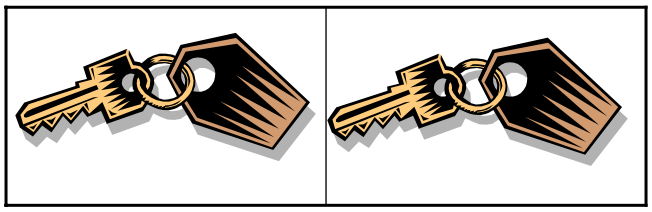
```
// Variáveis globais
BOOL solicita1 = FALSE;
BOOL solicita2 = FALSE;

Thread T1;
{
    loop {  ***
    a1:     while (solicita2 ) ;          // espera
    b1:     solicita1 = TRUE;             //  anuncia que deseja entrar
    c1:     SeçãoCrítica;
    d1:     solicita1 = FALSE;            //  libera direito para outras threads
    e1:     SeçãoNãoCrítica;
    } // end_loop
}   //  ThreadT1

Thread T2;
{
    loop {  ***
    a2:     while (solicita1 ) ;          // espera
    b2:     solicita2 = TRUE;             //  anuncia que deseja entrar
    c2:     SeçãoCrítica;
    d2:     Solicita2 = FALSE;            //  libera direito para outras threads
    e2:     SeçãoNãoCrítica;
    } // end_loop
}   //  ThreadT2
```

24

Modelo para Entendimento



Solicita1 Solicita2

25

Tentativa 1

Imagine a seguinte seqüência:

- T1 faz o loop em a1 e passa para b1
- T2 faz o loop em a2 e passa para b2
- T1 faz solicita1 = TRUE
- T2 faz solicita2 = TRUE
- T1 entra em sua seção crítica.
- T2 entra em sua seção crítica.

A exclusão mútua foi violada!

Este algoritmo falhou por excesso de agressividade ou de gentileza ?

26

Tentativa 2

```
// Variáveis globais
BOOL solicita1 = FALSE;
BOOL solicita2 = FALSE;
Thread T1;
{
        loop {      ***
        a1:         solicita1 = TRUE;          //  anuncia que deseja entrar
        b1:         while (solicita2 ) ;       // espera
        c1:         SeçãoCrítica;
        d1:         solicita1 = FALSE;         //  libera direito para outras threads
        e1:         SeçãoNãoCrítica;
        } // end_loop
}       //  ThreadT1

Thread T2;
{
        loop {      ***
        a2:         solicita2 = TRUE;          //  anuncia que deseja entrar
        b2:         while (solicita1 ) ;       // espera
        c2:         SeçãoCrítica;
        d2:         solicita2 = FALSE;         //  libera direito para outras threads
        e2:         SeçãoNãoCrítica;
        } // end_loop
}       //  ThreadT2
```

27

Demonstração

Temos que demonstrar quatro propriedades:

1. Exclusão Mútua
2. Ausência de *deadlock*
3. Ausência de inanição
4. Facilidade de acesso em caso de ausência de contenção

28

Demonstração – Exclusão Mútua

Por absurdo: Vamos supor que não ocorra exclusão mútua

Então é possível que as duas threads sejam flagradas em suas seções críticas ao mesmo tempo.

Vamos supor: T1 at(c1) e T2 at(b2) → at(c2)
Se T1 at(c1) ⊃ solicita1 = TRUE (Invariante)
Logo T2 não consegue fazer a transição: at(b2) → at(c2)

Analogamente:
Vamos supor: T2 at(c2) e T1 at(b1) → at(c1)
Se T2 at(c2) ⊃ solicita2 = TRUE (Invariante)
Logo T1 não consegue fazer a transição: at(b1) → at(c1)

Conclusão: A exclusão mútua se verifica

29

Demonstração – Ausência de deadlock

Imagine a seguinte seqüência de escalonamento das instruções das duas threads:

- T1 faz solicita1 = TRUE
- T2 faz solicita2 = TRUE
- T1 fica em loop aguardando solicita2 ser falso.
- T2 fica em loop aguardando solicita1 ser falso

Ocorreu Deadlock !!

Ou melhor: Livelock já que o loop é de espera ocupada !

- Deadlock é a condição em que uma thread fica bloqueada, esperando por uma condição que nunca ocorre
- Livelock é definido como a condição em que uma thread fica presa em um loop infinito esperando por uma condição que nunca irá ocorrer

30

Tentativa 3

```
// Variáveis globais:
int vez = 1;

Thread T1;
{
    loop { ***
    a1:    while (vez == 2 ) ;    // espera sua vez
    b1:    SeçãoCrítica;
    c1:    vez = 2;               //  libera direito para outra thread
    d1:    SeçãoNãoCrítica;
    } // end_loop
}   //  ThreadT1

Thread T2;
{
    loop { ***
    a2:    while (vez == 1) ;     // espera
    b2:    SeçãoCrítica;
    c2:    vez = 1;               //  libera direito para outra thread
    d2:    SeçãoNãoCrítica;
    } // end_loop
}   //  ThreadT2
```

31

Tentativa 3

Este algoritmo falha na ausência de contenção.

Por que ?

32

Algoritmo de Peterson

```
// Variáveis globais
int vez = 0;
BOOL solicita1 = FALSE;
BOOL solicita2 = FALSE;

Thread T1 {
loop {      ***
    a1:     solicita1 = TRUE;                 //  anuncia que deseja entrar
    b1:     vez = 2;                          //  cede direito à outra thread
    c1:     while (solicita2 && (vez == 2)) ;  // espera ocupada
    d1:     SeçãoCrítica;
    e1:     solicita1 = FALSE;                //  libera direito para outras threads
    f1:     SeçãoNãoCrítica;
    } // end_loop
}   //  ThreadT1

Thread T2 {
loop {      ***
    a2:     solicita2 = TRUE;                 //  anuncia que deseja entrar
    b2:     vez = 1;                          //  cede direito a outra thread
    c2:     while (solicita1 && (vez == 1)) ;  // espera ocupada
    d2:     SeçãoCrítica;
    e2:     solicita2 = FALSE;                //  libera direito para outras threads
    f2:     SeçãoNãoCrítica;
    } // end_loop
}   //  ThreadT2
```

33

Demonstração

- Todas as quatro propriedades fundamentais se verificam e podem ser demonstradas

Exemplo: Ausência de deadlock

Demonstração por absurdo:

- Se existe deadlock, então nenhuma thread consegue entrar na seção crítica e ficam presas em c1 e c2 respectivamente.
- Se T1 at(c1) ⊃ (solicita1 = TRUE) e (solicita2 and (vez=2)) é True
- Se T2 at(c2) ⊃ (solicita2 = TRUE) e (solicita1 and (vez=1)) é True
- Logo (vez=1) e (vez=2) é True, o que é absurdo

34

Algoritmo de Dekker

```
int c1, c2 = 1;
int vez = 1;
Thread T1 {
      loop {
      SeçãoNãoCrítica;
      c1 = 0;                          // T1 quer entrar na seção crítica
      while (c2 != 1)                  // protocolo de entrada: espera T2 liberar
               if (vez == 2) {         // detecta a contenção
                  c1 = 1;              // desiste da tentativa
                  while (vez != 1) ;   // espera sua vez de insistir caso empate
                  c1 = 0;              // ocupa C1
               }  // if
      }  // end_loop
      SeçãoCrítica;
      c1 = 1;                          // protocolo de Saída: libera T2
      vez = 2;                         // é a vez de T2 insistir em caso de empate
} // ThreadT1;

Thread T2 {
      loop {
      SeçãoNãoCrítica;
      c2 = 0;                          // T1 quer entrar na seção crítica
      while (c1 != 1)                  // protocolo de entrada: espera T1 liberar
               if (vez == 2) {         // detecta a contenção
                  c2 = 1;              // desiste da tentativa
                  while (vez != 2) ;   // espera sua vez de insistir caso empate
                  c2 = 0;              // ocupa C1
               }  // if
      }  // end_loop
      SeçãoCrítica;
      c2 = 1;                          // protocolo de Saída: libera T1
      vez =1;                          // é a vez de T1 insistir em caso de empate
} // ThreadT1;
```

35

Algoritmo de Dekker

Como o algoritmo de Dekker se compara como o algoritmo de Peterson ?

- O Algoritmo de Dekker é mais complexo e possui duas esperas ocupadas

Quais as deficiências dos dois algoritmos ?

- Ambos foram concebidos para 2 processos apenas
- Sua generalização para n processos é complexa
- Possuem esperas ocupadas
- Os protocolos de entrada e saída são difíceis de serem encapsulados em bibliotecas

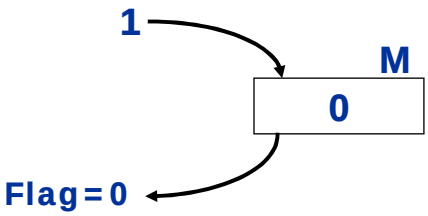
36

Exclusão Mútua com Instruções Especiais de Hardware

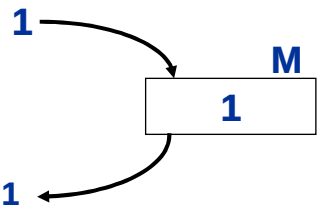
```
TAS(M)      if (M==0)
               flag = 0
               else flag = 1;
            M = 1;
```

M = Memória Global

Memória Livre



Memória Ocupada



Exclusão Mútua com Instruções Especiais de Hardware

```
int Global;
void ProtocoloEntrada() {
   #assembly
TasLoop:       tas(Global)      // espera ocupada
jnz TasLoop
}

Thread T1
{  loop {
         SeçãoNãoCrítica;
         ProtocoloEntrada();    // espera seção crítica livre
         SeçãoCrítica;
         Global = 0;            // libera seção crítica
   } // end loop
} // end T1
```

Evitando-se a espera ocupada no WNT

```
DWORD WaitForSingleObject(
HANDLE hHandle,          //  Handle para um objeto do kernel
DWORD dwMilliseconds     //  Tempo máximo que desejamos esperar
);
```

Status	Interpretação
WAIT_OBJECT_0	Objeto foi sinalizado
WAIT_TIMEOUT	Ocorreu timeout
WAIT_ABANDONED	Uma thread proprietária de um Mutex terminou (realizou ExitThread) sem liberá-lo. O objeto Mutex será estudado ainda neste capítulo.
WAIT_FAILED	A função falhou

39

Esperando por um só objeto

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <process.h>                    // _beginthreadex() e _endthreadex()
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>                      // _getch
#define _CHECKERROR         1           // Ativa função CheckForError
#include "CheckForError.h"

// Casting para terceiro e sexto parâmetros da função _beginthreadex
typedef unsigned (WINAPI *CAST_FUNCTION)(LPVOID);
typedef unsigned *CAST_LPDWORD;
DWORD WINAPI TestFunc(LPVOID);          // declaração da função

int main()
{
    HANDLE hThreads[3];
    DWORD dwThreadId;
    DWORD dwExitCode = 0;
    DWORD dwRet;
    int i;
```

40

Esperando por um só objeto

```
    for (i=0; i<3; ++i) {          // cria 3 threads
        hThreads[i] = (HANDLE) _beginthreadex(
                    NULL,
                    0,
                    (CAST_FUNCTION)TestFunc,
                    (LPVOID)i,
                    0,
                    (CAST_LPDWORD)&dwThreadId);
                    if (hThreads[i])
                       printf("Thread %d criada com Id= %0x \n", i, dwThreadId);
    }       // for
    for (i=0; i<3; ++i) {
        dwRet = WaitForSingleObject(hThreads[i], INFINITE);
        CheckForError(dwRet == WAIT_OBJECT_0);
        GetExitCodeThread(hThreads[i], &dwExitCode);
        printf("thread %d terminou com codigo de saida %d\n", i, dwExitCode);
        CloseHandle(hThreads[i]);        // apaga referência ao objeto
    }
    printf("\nAcione uma tecla para terminar\n");
    _getch(); // Pare aqui, caso não esteja executando no ambiente MDS
    return EXIT_SUCCESS;
}   // main
```

41

Esperando por um só objeto

```
DWORD WINAPI TestFunc(LPVOID index)
{   int i;
    Sleep(50);
    for(i=0; i<50; ++i) {
        printf("%d ", index);
        // Sleep(10);                    // delay de 10 ms
    }
    printf("\n");
    _endthreadex((DWORD) index);
    return(0);
} // TestFunc
```

42

WaitForMultipleObjects

DWORD WaitForMultipleObjects (

```
DWORD nCount,                  // Número de handles a esperar,  limitado por MAXIMUM_WAIT_OBJECTS
CONST HANDLE *lpHandles,       // Vetor de handles para objetos do kernel.
BOOL bWaitAll,                 // TRUE: retorna se todos os handles forem sinalizados.
                               // FALSE:  retorna se qualquer handle for sinalizado
DWORD dwMilliseconds           // Tempo máximo que desejamos esperar
);
```

bWaitAll	Status	Interpretação
TRUE FALSE	WAIT_OBJECT_0 Valor	Todas threads retornaram Valor-WAIT_OBJECT_0 = índice da thread que retornou
	WAIT_ABANDONED_0 + x	Uma thread proprietária de um Mutex o abandona sem liberá-lo x é o índice da thread.
	WAIT_TIMEOUT	Ocorreu *timeout*
	WAIT_FAILED	Função falhou

43

Uso de WaitForMultipleObjects

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>                    // _beginthreadex() e _endthreadex()
#include <conio.h>                      // _getch
#define _CHECKERROR          1          // Ativa função CheckForError
#include "CheckForError.h"
// Casting para terceiro e sexto parâmetros da função _beginthreadex
typedef unsigned (WINAPI *CAST_FUNCTION)(LPVOID);
typedef unsigned *CAST_LPDWORD;
DWORD WINAPI TestFunc(LPVOID);          // declaração da função

int main()
{
HANDLE hThreads[3];
DWORD dwThreadId;
DWORD dwExitCode = 0;
DWORD dwRet;
int i;
```

44

Uso de WaitForMultipleObjects

```
    for (i=0; i<3; ++i) {          // cria 3 threads
            hThreads[i] = (HANDLE) _beginthreadex(
            NULL,
            0,
            (CAST_FUNCTION)TestFunc,
            (LPVOID)i,
            0,
            (CAST_LPDWORD)&dwThreadId);
            if (hThreads[i]) printf("Thread %d criada com Id= %0x \n", i, dwThreadId);
    }  // for

    dwRet = WaitForMultipleObjects(3,hThreads,TRUE,INFINITE);
    CheckForError((dwRet >= WAIT_OBJECT_0) && (dwRet < WAIT_OBJECT_0 + 3));
    for (i=0; i<3; ++i) {
            GetExitCodeThread(hThreads[i], &dwExitCode);
            printf("thread %d terminou: codigo=%d\n",i,dwExitCode);
            CloseHandle(hThreads[i]);        // apaga referência ao objeto
    }  // for

    printf("\nAcione uma tecla para terminar\n");
    _getch();   // Pare aqui, caso não esteja executando no ambiente MDS
    return EXIT_SUCCESS;
} // main
```

45

Uso de WaitForMultipleObjects

```
DWORD WINAPI TestFunc(LPVOID index)
{   int i;
    for(i=0; i<50; ++i) {
            printf("%d ", index);
    // Sleep(10);      // delay de 10 ms
    }
    printf("\n");
    _endthreadex((DWORD) index);
    return(0);
} // TestFunc
```

46

Objetos do Kernel

OBJETO	DESCRIÇÃO
Thread	Sinalizado quando a thread termina
Process	Sinalizado quando a última thread termina
Change notification	Sinalizado quando um tipo particular de mudança ocorre em um diretório
Console input	Sinalizado quando uma entrada está disponível no buffer da console
Event	Estado controlado diretamente pela aplicação através de *SetEvent()*, *PulseEvent()* e *ResetEvent()*
Mutex	Está sinalizado quando não é possuído por nenhuma thread
Semaphore	Está sinalizado quando seu valor é maior que zero e não sinalizado quando seu valor é zero
File	Está sinalizado quando a operação de I/O termina
Timer	Está sinalizado quando o valor setado é atingido, ou o intervalo expira

47

CriticalSection

```
VOID InitializeCriticalSection(
LPCRITICAL_SECTION lpCriticalSection    // Apontador  para variável do tipo CRITICAL_SECTION.
);

VOID EnterCriticalSection(
LPCRITICAL_SECTION lpCriticalSection    // Apontador  para variável do tipo CRITICAL_SECTION.
);

VOID LeaveCriticalSection(
LPCRITICAL_SECTION lpCriticalSection    // Apontador  para variável do tipo CRITICAL_SECTION.
);

VOID DeleteCriticalSection(
LPCRITICAL_SECTION lpCriticalSection    // Apontador  para variável do tipo CRITICAL_SECTION.
);
```

48

Exemplo – CriticalSection

```
DWORD WINAPI MyThread(LPVOID);          // declaração da função
// Variáveis Globais
CRITICAL_SECTION CriticalSection;

int main()  {
    HANDLE hThread;
    DWORD dwThreadId;
    DWORD dwExitCode = 0;
    DWORD dwRet;
    InitializeCriticalSection(&CriticalSection);
    hThread = (HANDLE) _beginthreadex(
                        NULL,
                        0,
                        (CAST_FUNCTION)MyThread,
                        (LPVOID)i,
                        0,
                        (CAST_LPDWORD)&dwThreadId
              );
    *****************
    EnterCriticalSection(&CriticalSection);
    SeçãoCrítica;
    LeaveCriticalSection(&CriticalSection);
    *****************
```

49

Exemplo – CriticalSection

```
    // Espera a thread terminar
    dwRet = WaitForSingleObjects(hThread, INFINITE);
    CheckForError(dwRet);
    GetExitCodeThread(hThread, &dwExitCode);
    CloseHandle(hThread);       // apaga referência ao objeto
    DeleteCriticalSection(&CriticalSection);
    return EXIT_SUCCESS;
}   // main

DWORD WINAPI MyThread(LPVOID index)
{
    EnterCriticalSection(&CriticalSection);
    SeçãoCrítica;
    LeaveCriticalSection(&CriticalSection);
    _endthreadex((DWORD) index);
    return();
} // MyThread
```

50

Mutex

- O nome Mutex vem de Mutual Exclusion
- Mutex são objetos do Kernel especializados na sincronização para acesso a uma seção crítica
- O uso do Mutex para controlar o acesso a uma seção Crítica implica em maior overhead que o uso de CriticalSections já que é necessária uma chamada ao sistema operacional

51

CreateMutex

```
HANDLE CreateMutex(
LPSECURITY_ATTRIBUTES lpMutexAttributes,   // Apontador para atributos de segurança.
BOOL bInitialOwner,                        // TRUE: thread que criou é a proprietária.
LPCTSTR lpName                             // Nome do objeto.
);
```

Retorno:

Status	Interpretação
Handle para o Mutex criado	Sucesso
NULL	Falha

52

OpenMutex

HANDLE OpenMutex(

DWORD dwDesiredAccess,	// Atributos de segurança: MUTEX_ALL_ACCESS: o handle pode ser usado em qualquer função. SYNCHRONIZE: o handle pode ser usado apenas nas funções *Wait… e ReleaseMutex().*
BOOL bInheritHandle,	// TRUE: handle será herdável.
LPCTSTR lpName	// Nome do Mutex a ser aberto.
);	

Retorno:

Status	Interpretação
Handle para o Mutex aberto	Sucesso
NULL	Falha

53

Wait / ReleaseMutex

Operação de Wait:

DWORD WaitForSingleObject(	
HANDLE hHandle,	// Handle para o Mutex
DWORD dwMilliseconds	// Tempo máximo que desejamos esperar
);	

Operação de Release:

HANDLE ReleaseMutex(HANDLE hMutex);

Retorno:

Status	Interpretação
TRUE	Sucesso
FALSE	Falha

54

Mutex - Exemplo

```
// Variáveis Globais
HANDLE hMutex;

int main()
{
    HANDLE hThread;
    DWORD dwThreadId;
    DWORD dwExitCode = 0;
    DWORD dwRet;
    hMutex = CreateMutex(NULL, FALSE, "AcessaMapa");
    hThread = (HANDLE) _beginthreadex(
                                NULL,
                                0,
                                (CAST_FUNCTION)MyThread,
                                (LPVOID)i,
                                0,
                                (CAST_LPDWORD)&dwThreadId
                                );
    ***************************
    WaitForSingleObject(hMutex, INFINITE);
    SeçãoCrítica;
    ReleaseMutex(hMutex);
    ***************************
```

55

Mutex - Exemplo

```
    // Espera a thread terminar
    dwRet  = WaitForSingleObjects(hThread, INFINITE);
    CheckForError(dwRet);
    GetExitCodeThread(hThread, &dwExitCode);
    CloseHandle(hThread);                    // apaga referência ao objeto
    CloseHandle(hMutex);
    return EXIT_SUCCESS;
}   // main


DWORD WINAPI MyThread(LPVOID index)
{
    WaitForSingleObject(hMutex, INFINITE);
    SeçãoCrítica;
    ReleaseMutex(hMutex);
    _endthreadex((DWORD) index);
    return();
} // MyThread
```

56

Propriedades do Mutex

- Quando a Seção Crítica está livre, o Mutex fica no estado sinalizado
- Quando uma thread faz um Wait em um Mutex sinalizado ele passa ao estado não sinalizado e dizemos que a thread é proprietária do Mutex
- Se uma thread terminar dentro da seção crítica, todas as demais threads presas em uma operação de Wait neste Mutex retornarão com o valor:

WaitForSingleObject()	WAIT_ABANDONED_0
WaitForMultipleObjects()	WAIT_ABANDONED_0 + X, onde X é o índice da thread que abandonou a Seção Crítica

- Ao esperar por um Mutex podemos estipular um tempo de timeout
- Tanto a instrução de Wait como a de Release devem pertencer a uma mesma thread
- Para sincronizar threads diferentes use Semáforos binários

57

Funções complementares

```
LONG InterlockedIncrement(
LPLONG lpAddend

);
```

// Endereço de posição de memória de 32 bits alinhada em uma fronteira de 32 bits. A posição será incrementada e o resultado comparado com zero.

```
LONG InterlockedDecrement(
LPLONG lpAddend

);
```

// Endereço de posição de memória de 32 bits, alinhada em uma fronteira de 32 bits. A posição será decrementada e o resultado comparado com zero.

Retorno:

Status	Interpretação
ret	Valor incrementado / decrementado

```
LONG InterlockedExchange(
LPLONG lpTarget,
LPLONG Value
);
```

// Endereço de posição de memória de 32 bits a ser trocada, alinhada em uma fronteira de 32 bits

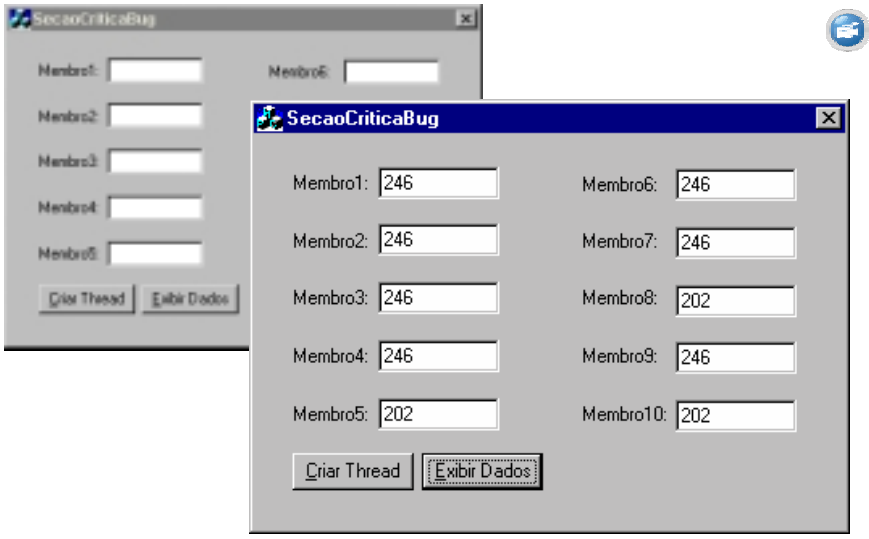
// Valor a ser carregado na posição apontada por lpTarget

Retorno:

Status	Interpretação
ret	Valor anterior de lpTarget

58

Exemplos MFC – Seções Críticas



59

Exemplos MFC – Seções Críticas - Antes

```
UINT Thread (LPVOID Param)  {
    DWORD ID;
    ID = GetCurrentThreadId();
    while (TRUE)  {
          Registro.Membro1 = ID;
          Registro.Membro2 = ID;
          Registro.Membro3 = ID;
          Registro.Membro4 = ID;
          srand ((unsigned) time (NULL));
          Sleep (rand() / 100);
          Registro.Membro5 = ID;
          Registro.Membro6 = ID;
          Registro.Membro7 = ID;
          srand ((unsigned) time (NULL));
          Sleep (rand() / 100);
          Registro.Membro8 = ID;
          Registro.Membro9 = ID;
          Registro.Membro10 = ID;
    }
    return 0;
}
```

60

Exemplos MFC – Seções Críticas Depois

```
UINT Thread (LPVOID Param) {
    DWORD ID;
    ID = GetCurrentThreadId ();
    while (TRUE)    {
          EnterCriticalSection (&csCriticalSection);
          Registro.Membro1 = ID;
          Registro.Membro2 = ID;
          Registro.Membro3 = ID;
          Registro.Membro4 = ID;
          srand ((unsigned) time (NULL));
          Sleep (rand() / 100);
          Registro.Membro5 = ID;
          Registro.Membro6 = ID;
          Registro.Membro7 = ID;
          srand ((unsigned) time (NULL));
          Sleep (rand() / 100);
          Registro.Membro8 = ID;
          Registro.Membro9 = ID;
          Registro.Membro10 = ID;
          LeaveCriticalSection (&csCriticalSection);
    }
    return 0;
}
```

61

Exemplos MFC – Seções Críticas

```
void CSecaoCriticaDlg::OnExibir()
{
    char szAux[5];

    /* esta região também deve ser protegida pelo mutex */
    EnterCriticalSection (&csCriticalSection);
    Membro1->SetWindowText (itoa (Registro.Membro1, szAux, 10));
    Membro2->SetWindowText (itoa (Registro.Membro2, szAux, 10));
    Membro3->SetWindowText (itoa (Registro.Membro3, szAux, 10));
    Membro4->SetWindowText (itoa (Registro.Membro4, szAux, 10));
    Membro5->SetWindowText (itoa (Registro.Membro5, szAux, 10));
    Membro6->SetWindowText (itoa (Registro.Membro6, szAux, 10));
    Membro7->SetWindowText (itoa (Registro.Membro7, szAux, 10));
    Membro8->SetWindowText (itoa (Registro.Membro8, szAux, 10));
    Membro9->SetWindowText (itoa (Registro.Membro9, szAux, 10));
    Membro10->SetWindowText (itoa (Registro.Membro10, szAux, 10));
    LeaveCriticalSection (&csCriticalSection);
}
```

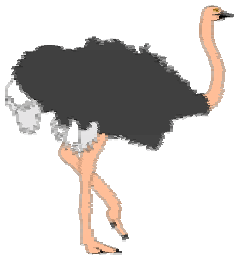
62

Insight x Método

Existem avestruzes e girafas em um cercado.
No total são 22 cabeças e 60 pernas.
Quantas são as avestruzes e as girafas ?
Não vale montar sistema de equações.

63

Nova visão do problema



64

Invariante

Quebra cabeça dos 15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Este movimento é possível ?

Qual o invariante deste jogo ?

65

Qual o invariante deste jogo ?

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

1	2	3	4
5	6	7	8
9	10	11	
13	14	15	12

1	2	3	4
5	6	7	8
9	10		11
13	14	15	12

1	2	3	4
5	6	7	8
9		10	11
13	14	15	12

1	2	3	4
5	6	7	8
9	14	10	11
13		15	12

1	2	3	4
5	6	7	8
9	14	10	11
13	15	12	

A=Conjunto de pares fora de ordem

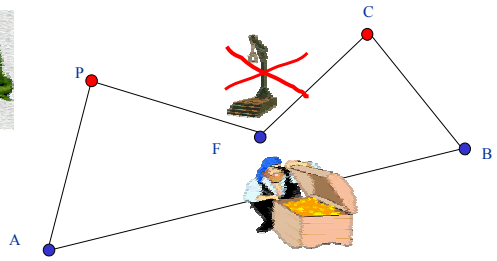
A = { (14,10), (14, 11), (14, 13), (14, 12), (13, 12), (15, 12)}

#A = 6 que é PAR

66

Invariante

Um pirata escondeu um tesouro em uma ilha e construiu um mapa com indicações:



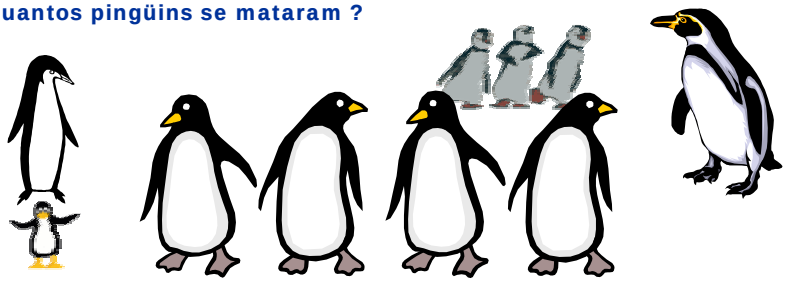
- Chegando à ilha vá até o poste da forca indicado na figura pela letra F e de lá ande reto até um pinheiro P, medindo a distância percorrida.
- Chegando ao pinheiro, vire à esquerda e ande igual distância em angulo reto com a trajetória anterior. Marque o ponto A.
- Ainda a partir da forca ande em direção ao carvalho C e lá chegando vire à direita, caminhando em angulo reto a mesma distancia anterior. Marque o ponto B. O tesouro está enterrado no ponto médio do segmento AB.
- Ao chegar à ilha constataram que a forca havia sido demolida e já não existia mais nem traço de onde estava localizada, mas o pinheiro e o carvalho ainda estavam lá.
- Mesmo assim eles encontram o tesouro. Descubra como e qual o raciocínio empregado.

From: The kinematic method in geometrical problems – Yu.I.Lyubich and L.A.Shor, Mir Publishers

67

Exercício P.I.F.

- Era uma vez uma pequena ilha habitada por pingüins.
- Um determinado dia, Frigidaire, Deus dos pingüins, anunciou que uma maldição estava prestes a se realizar. Exatamente na próxima segunda-feira, uma terrível doença se abateria sobre pelo menos um de seus súditos. O único sintoma da doença seria uma mancha vermelha na testa (lembra-se de Gorbachev ?).
- O pingüim que estivesse doente deveria se matar até o final do dia em que soubesse de sua terrível condição. Ao final do quinto dia, houve morte e ranger de bicos.
- Quantos pingüins se mataram ?



68

Pequeno Teorema de Fermat

Se p é um número primo então para qualquer inteiro a, a^p - a é divisível por p

A demonstração foi proposta por Euler em 1736 (Boyer, pg 336):

Usando o princípio da indução finita:

1) **Para a = 1: $1^p - 1 = 0$. Logo a propriedade se verifica**

2) **Vamos assumir a propriedade verdadeira para a = k: $k^p - k = np$**

3) **Para a = k+1 temos que provar que a propriedade continua verdadeira**

   Euler usou o teorema binomial:

   $(k+1)^p = k^p + mp + 1$, onde m é um inteiro (Por que ?)

   $(k+1)^p - (k+1) = K^p + mp - k = mp + k^p - k = mp + np = (m+n)p$

   Logo o número resultante é divisível por k

69

Muito Obrigado

Perguntas?

Constantino Seixas Filho

constantino.seixas@task.com.br

70