URI | UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES

30-769

# Linguagem de Programação IV

MSc. Fernando Schubert

# CRIAÇÃO ESTÁTICA E DINÂMICA DE PROCESSOS

- Os processos de um programa concorrente podem ser criados de forma estática ou dinâmica.
  - No primeiro caso, o programa contém a declaração de um conjunto fixo de processos, os quais são ativados simultaneamente, no início da execução do programa.
  - No segundo caso, os processos são criados dinamicamente, durante a execução, através de instruções especiais para esse fim.
- Os mecanismos vistos até aqui (fork, join e quit) realizam criação dinâmica (e término dinâmico), pois os processos (ou threads) são criados somente quando instruções especiais são executadas.

# CRIAÇÃO ESTÁTICA

- No caso da criação estática, os processos são declarados explicitamente no programa fonte e vão existir desde o início da execução do programa concorrente.
- Normalmente, as linguagens de programação permitem especificar esses processos de duas maneiras: como processos individuais ou como um array de processos.

# CRIAÇÃO ESTÁTICA

- Especificação de processos individuais: Neste caso, cada processo é especificado de forma individual, conforme exemplificado a seguir:

```
V4program
    process P1;
    k: integer init 0;
    while k < 10 do
    {   write(1);
        k:=k+1
    };
    process P2;
    k: integer init 0;
    while k < 10 do
    {   write(2);
        k:=k+1
    }
endprogram
```

- O programa define 2 processos, denominados P1 e P2, que não compartilham variáveis (não existem variáveis globais no programa).
- Cada processo utiliza uma variável local, denominada k.
- O primeiro imprime 10 vezes o número 1 e o segundo, em paralelo, imprime 10 vezes o número 2.
- O resultado da execução pode ser qualquer seqüência de tamanho 20, contendo 10 vezes o número 1 e 10 vezes o número 2, embaralhados. Teoricamente, são possíveis 20!/(10!*10!) resultados diferentes.

# CRIAÇÃO ESTÁTICA

- Array de processos:
  - Neste caso, uma única declaração específica um grupo de processos semelhantes, que se distinguem apenas pelo valor de uma variável local inteira, que é especificada no cabeçalho do processo, conforme é ilustrado a seguir, considerando o mesmo exemplo anterior.

```
V4program
    process P (i := 1 to 2);
    k: integer init 0;
    while k < 10 do
    {   write(i);
        k:=k+1
    }
endprogram
```

- Para o primeiro processo, a sua variável local i vale 1 e para o segundo, a sua variável local i vale 2. Este programa é equivalente ao anterior.

# CRIAÇÃO DINÂMICA

- É possível declarar explicitamente um modelo (uma "forma") para criar processos durante a execução, conforme é ilustrado a seguir.
- Neste caso tem-se o que se denomina criação dinâmica com declaração explícita de processos.

# CRIAÇÃO DINÂMICA

```
V4program
    process type P (i: integer);
    k: integer init 0;
    while k < 10 do
    {   write(i);
        k:=k+1
    };
    process Q;
    { new P(1); new P(2) }
endprogram
```

- Através da especificação process type, explicita-se um modelo (template) de processo, o qual é utilizado para criar exemplares (cópias, clones) desse processo durante a execução.
- A criação de um novo exemplar se dá através do comando new, o qual permite passar parâmetros para o processo criado.

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- Compartilhamento de um procedimento:
  - Cada processo chama imprime fornecendo como argumento o número a ser impresso. Como nos exemplos anteriores, o resultado da execução desse programa é imprevisível, sendo possíveis (teoricamente) 10 C20 resultados distintos.

```
V4program
    procedure imprime(i: integer);
    k: integer init 0;
    while k < 10 do
    {   write(i);
        k:=k+1
    };
    process P1; imprime(1);
    process P2; imprime(2)
endprogram
```

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- Compartilhamento de um procedimento:
  - Observação sobre as variáveis de um procedimento
    - No exemplo anterior, existem duas execuções concorrentes do procedimento imprime, uma efetuada por P1 e outra por P2.
    - Cada uma dessas execuções utiliza variáveis i (argumento) e k (variável local do procedimento).
    - A observação importante é que cada processo utiliza cópias independentes dessas variáveis.
    - O código do procedimento é compartilhado pelos dois processos, mas os dados (variáveis i e k) são privativos de cada processo.
    - Não se pode esquecer que, na chamada de um procedimento, os argumentos e as variáveis locais desse procedimento são alocados na pilha do processo chamador.
    - Como cada processo (ou thread) trabalha com uma pilha própria, as execuções não interferem uma com a outra.
    - Na programação concorrente é sempre assim, todo procedimento é automaticamente reentrável (ou puro).
      - Isto significa que o mesmo código pode ser executado simultaneamente por vários processos sem que haja interferência entre eles, pois cada processo utiliza um conjunto separado de parâmetros e de variáveis locais.

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- Compartilhamento de uma variável:
  - O programa a seguir implementa um sistema concorrente onde dois processos compartilham uma variável global S. Cada processo incrementa S de uma unidade, 100 vezes.

```
V4program
   S : integer init 0;
   process  p1;
   k: integer init 0;
   {   loop
            S:= S+1;
            k:= k+1;
            exit when k = 100
      endloop;
      nl; write('p1'); tab(2); write(S)
   };
```

```
process  p2;
k: integer init 0;
{   loop
            S:= S+1;
            k:= k+1;
            exit when k = 100
      endloop;
      nl; write('p2'); tab(2); write(S)
   }
endprogram
```

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- O problema da exclusão mútua:
  - No programa anterior, tem-se 2 processos manipulando a variável global S. Cada processo executa 100 vezes o comando S:=S+1. Este comando é compilado para a seguinte seqüência de código de máquina:

```
push   S          % coloca o valor de S na pilha
push   $1         % coloca a constante 1 na pilha
add               % soma os dois últimos valores colocados na pilha
pop    S          % guarda o resultado em S
```

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- O problema da exclusão mútua:
  - Como os pontos em que a UCP passa de um processo para outro são imprevisíveis, pode acontecer de um processo perder a UCP no meio da seqüência acima.
  - Vamos supor que o valor de S carregado na pilha seja 10 e que o processo perca a UCP.
  - Nesse caso, quando este processo receber a UCP de volta, ele vai concluir a seqüência acima e armazenar o valor 11 em S.
  - Todos os acréscimos a S feitos pelo outro processo nesse ínterim são perdidos.
  - Como conseqüência, embora S inicie com o valor zero e cada processo some 1 a S cem vezes, o valor final de S dificilmente será igual a 200.
  - Para o resultado ser 200, deve haver exclusão mútua no acesso à variável S, isto é, enquanto um processo estiver manipulando S, o outro não pode acessar S.
  - A exclusão mútua é um requisito muito importante nos sistemas concorrentes.
  - Em geral, é necessário garantir o acesso exclusivo aos dados compartilhados.
  - A exclusão mútua só não é necessária quando os dados são compartilhados na modalidade "apenas leitura", isto é, quando os dados não são alterados pelos processos.
  - Os trechos dos processos onde os dados compartilhados são manipulados, são denominados trechos críticos ou regiões críticas.

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- Criação dinâmica de processos
  - A criação dinâmica (e recursiva) de processos é ilustrada através do problema da torre de Hanói, descrito a seguir.
  - Tem-se 3 torres (pilhas) e n discos de tamanhos diferentes.
  - Inicialmente os n discos estão empilhados na torre 1, na ordem certa (maior na base, menor no topo).
  - O problema consiste em movimentar todos os discos para uma determinada torre, sem que nunca um disco maior fique sobre um disco menor.
  - Os discos devem ser transportados de um em um e a terceira torre pode ser usada como intermediária nessas movimentações.

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- Criação dinâmica de processos
  - O programa inicia com o processo P criando um filho (escravo) Hanoi para movimentar 3 discos da torre 1 para a torre 2, usando a torre 3 como intermediária.
  - Para movimentar n discos de a para b, usando c como torre intermediária, um escravo Hanoi faz o seguinte.
  - Se n = 1 (só tem um disco para movimentar), então o trabalho é fácil e direto: o escravo movimenta o único disco de a para b (e mostra essa movimentação).
  - Caso contrário, o escravo cria um escravo_1 para movimentar n-1 discos de a para c, usando b como torre intermediária.

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- Criação dinâmica de processos
  - Quando esse serviço é concluído (escravo_1 termina o seu trabalho), o escravo original movimenta o disco que lhe sobrou (que é o maior) de a para b (mostra essa movimentação) e cria um escravo_2 para concluir o serviço, que é movimentar n-1 discos de c para b usando a como torre intermediária.

# EXEMPLOS DE PROGRAMAS CONCORRENTES

- Criação dinâmica de processos

```
V4program
    process type Hanoi(n, a, b, c: integer);
    id, m: integer;
        if n = 1
        then { nl; write(a); write(' --> '); write(b) }
        else { m:= n-1;
                    id:= new Hanoi(m, a, c, b); join(id);
                    nl; write(a); write('  -->  '); write(b);
                    id:= new Hanoi(m, c, b, a); join(id)
                };
    process P; new Hanoi(3, 1, 2, 3)
endprogram
```