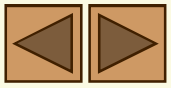
Expressão da Concorrência

Autoria

- Alunos de turma de SOII
- C. Geyer
- Versão
  - V23, 2011-2

Índice



Representação de uma tarefa

- Forma abstrata
  - Independentemente de modelo e linguagem de programação
- Processo
  - conceito ou construção de SO e bibliotecas
  - freqüentemente (↓) associado a uma tarefa
- Thread
  - conceito ou construção de SO, bibliotecas e de linguagens
  - freqüentemente (↑) associado a uma tarefa

Processo x thread

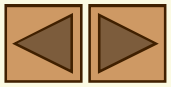
- ✓ Processo
  - Mais antigo
  - Mais lento (criação, ...)
  - Maior consumo de memória
  - Oferece mecanismos de proteção entre processos
  - Não oferece variáveis (memória) compartilhada entre processos
    - Via bibliotecas
  - Mais usado em PC com memória distribuída (redes, ...)

Processo x thread

- Thread
  - Em geral o inverso de processo
    - P.ex.: mais rápido
  - Mais usado em PC com memória compartilhada (1 computador)

Tipos de criação

- ✓ Inicialmente, convém considerar as formas pelas quais a concorrência da execução pode ser especificada.
- ✓ Tipos em função de “quando”
  - estática
    - o programa contém a declaração de um conjunto fixo de processos
    - os quais são ativados simultaneamente, no início da execução do programa
    - mais simples para o programador
    - eventualmente gerência de tarefas mais simples -> mais eficiente



Tipos de criação

- Tipos em função de “quando”
  - dinâmica
    - pois os processos são criados e terminados dinamicamente, durante a execução
    - mais flexibilidade, pois os processos são criados e terminados conforme as necessidades da aplicação
    - quantidade variável conforme necessidade em cada execução e momento

Tipos de criação

- Em função de "quem"
  - implícita
    - sistema cria automaticamente
      - Compilador
      - Máquina virtual (JVM, ...)
    - mais simples para o programador
    - freqüentemente menos flexível
    - nem sempre o "sistema" sabe quando, o que, ...

Tipos de criação

- Em função de “quem”
  - explícita
    - programador deve usar uma primitiva ou construção específica
    - mais flexível
    - trabalho extra para o programador

Tipos de criação

- Freqüentemente
  - estática-implícita
    - MPI (biblioteca para programação paralela)
    - Versão 1: estática em tempo de carga do programa
    - N instâncias do mesmo código onde N é quantidade de cpus
  - dinâmica-explícita
    - fork (Unix)
    - Comando ou procedimento que cria novo processo

Tipos de criação

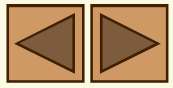
- ✓ Outros casos
  - dinâmica-implícita
    - paralelização automática (via compilação) de linguagens
      - Prolog, Lisp
      - Fortran, C
  - estática-explícita
    - SR (linguagem proposta por [Andrews 1991]
    - declaração de processos por palavra reservada e begin/end
    - ativados no início da execução
    - quantidade fixa em todas as execuções
    - obs: processos são encapsulados em um “resource” o qual é dinâmico
  - Vários outros casos: combinando tipos acima

Tipos de criação

- Outros casos
  - Exercício
    - Considere uma linguagem onde se possa criar uma tarefa via marcação de um trecho de código com uma palavra reservada
    - ...
      Task:
      x := y +1;
      z := fibonacci(x);
      ...
    - É modo implícito ou explícito? Dinâmico ou estático?

Conceito

- Grafo de Precedência
  - grafo (usualmente) acíclico dirigido, no qual cada nó representa a execução de um processo
  - pode ser interpretado como um grafo de fluxo de processos
  - normalmente usado em projeto, especificação, análises, ...
    - editor convencional
    - editores para PC



Grafo de Precedência

- Cada nodo representa um processo;
- Cada arco indica a precedência de execução e restrição na ordem dos processos;
- Exemplo:

  - Pj só pode ser executado após Pi, ou seja, Pj depende de Pi

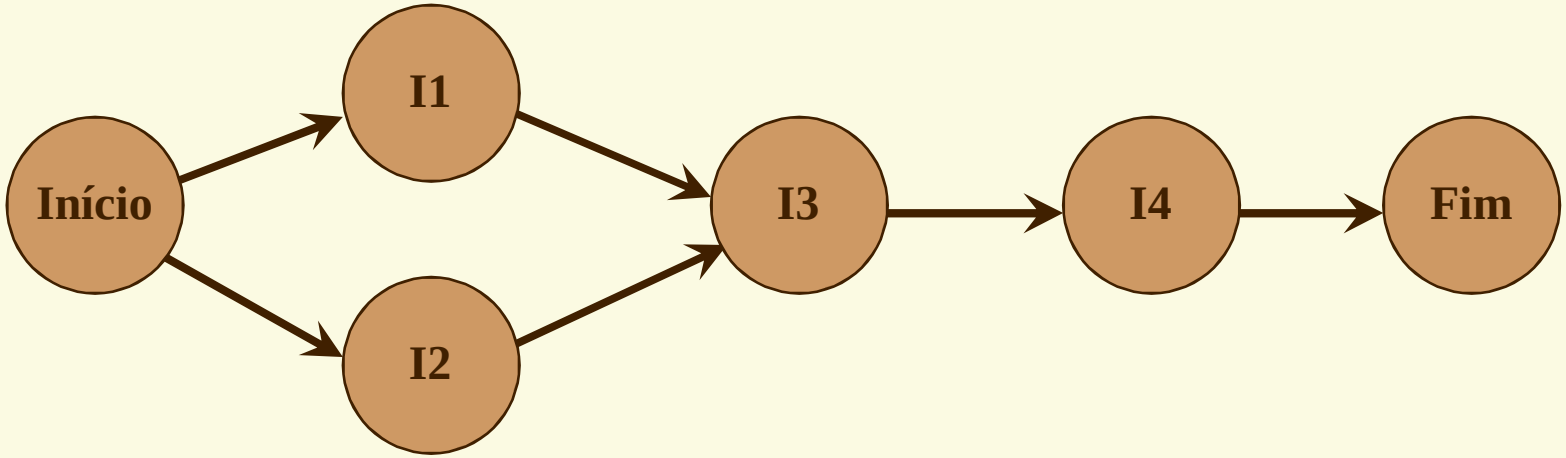
Grafo de Precedência - Exemplo 1

Programa:

```
I1 :: A := X + Y;
I2 :: B := Z + 1;
I3 :: C := A - B;
I4 :: W := C + 1;
```

- **Suponha que queremos executar algumas dessas instruções concorrentemente.**
- **Claramente, é possível perceber que a instrução I3 não pode ser executada antes de I1 e I2, pois I3 necessita dos novos valores de A e B.**
- **E o mesmo acontece com I4, que não pode ser executado antes que o valor de C seja atualizado por I3.**

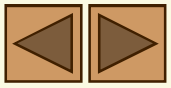
Grafo de Precedência - Exemplo 1



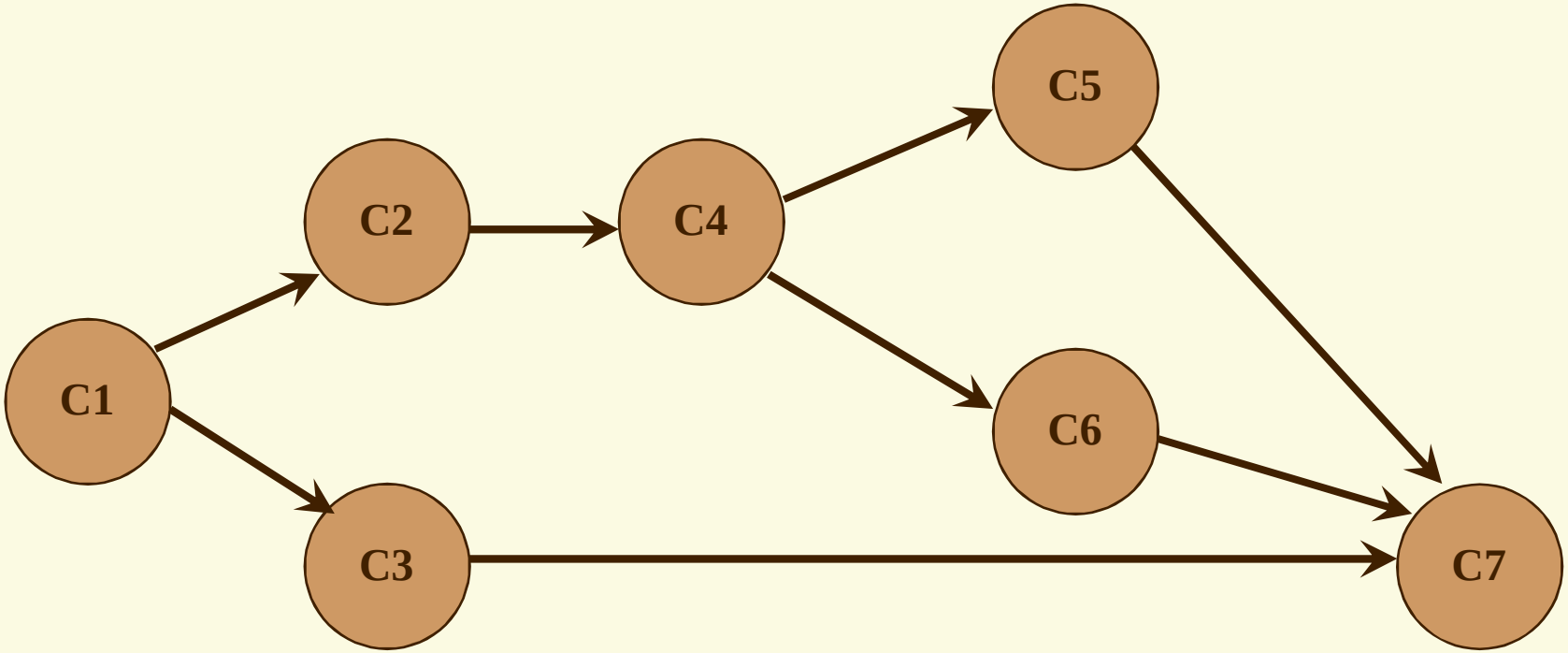
Grafo de Precedência correspondente ao programa apresentado

Grafo de Precedência - Exemplo 2

- Considere um grafo de precedência com as seguintes relações de precedência:
  - C2 e C3 podem ser executados depois da execução de C1;
  - C4 pode ser executado depois da execução de C2;
  - C5 e C6 podem ser executados depois da execução de C4;
  - C7 só pode ser executado depois da execução de C5, C6 e C3;



Grafo de Precedência - Exemplo 2



Grafo de Precedência

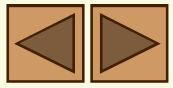
- Limitações
  - não representa comunicação contínua (repetida) entre dois processos
  - não representa comunicação realizada durante a execução (subcaso)
  - não representa repetições de processos
  - na literatura: inúmeros extensões, ou ainda outras propostas, para contornar essas e outras limitações

Usos

- ✓ Em projeto e especificação de programas concorrentes
- ✓ Em análise da complexidade de PCs
- ✓ Em programação de programas paralelos
  - em geral, modelos mais restritos
  - por exemplo, a comunicação entre processos é somente por arquivos
- ✓ No controle da execução de um PC
  - Meta programação ou programação em 2 níveis

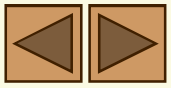
Conceito

- Fork/Join
  - As instruções **fork** e **join** foram introduzidas por Conway [1963] e Dennis e Van Horn [1966];
  - Foram a primeira notação de linguagem para especificação de concorrência;



Fork

- A instrução fork L produz duas execuções concorrentes (fluxo, tarefa) num programa.
  - Uma execução começa na instrução rotulada L,
  - enquanto a outra é a continuação da execução na instrução seguinte à instrução fork
  - Processo (tarefa) filho:
    - Usualmente a nova execução concorrente
  - Processo pai:
    - Usualmente a que executou o fork

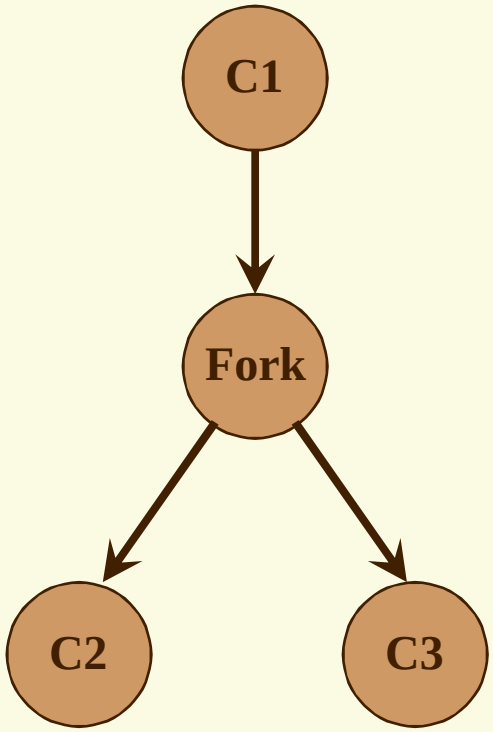


Fork

- Para ilustrar este conceito, considere o seguinte segmento de programa:

```
C1;
fork L;        // L é um rótulo
C2;
.......
L: C3;
```

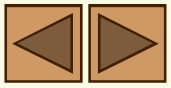
Fork



- Parte do grafo de precedência do programa é apresentado na figura ao lado.
- Quando a instrução **fork** L é executada, uma nova computação é iniciada em C3. Esta nova computação executa concorrentemente com a antiga computação, a qual continua em C2.

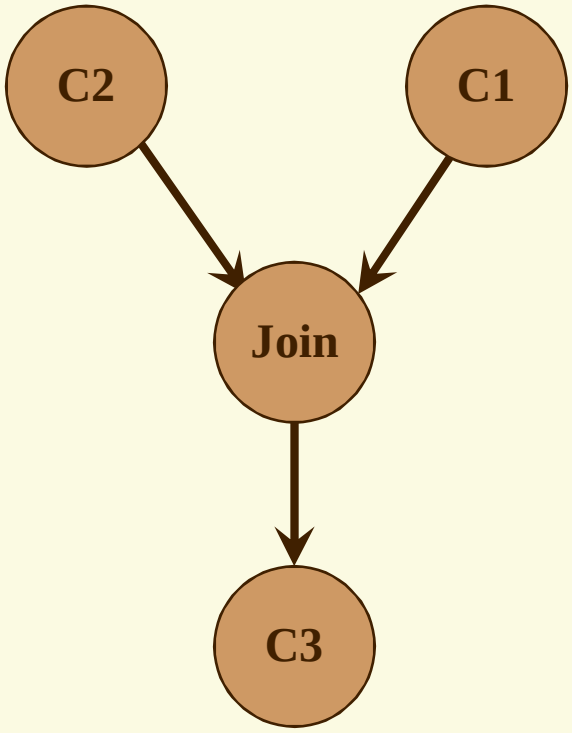
Join

- A instrução **join** agrupa duas computações concorrentes em uma.
  - Cada uma das computações deve requisitar para ser agrupada com a outra.
  - Já que as computações podem ter tempos de execução diferentes, uma pode executar o **join** antes da outra e sua execução é terminada.
- A instrução **join** deve ser executada de forma atômica.
  - Sem interrupções na visão de terceiros



Join - Exemplo 1

```
count := 2;
fork L1;
...
C1;
goto L2;
L1: C2;
L2: join count;
```



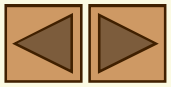
Parte do grafo de precedência para a programa ao lado

Fork/Join - Exemplo

```
C1;
count := 3;
fork L1;
C2;
C4;
fork L2;
C5;
goto L3;
```

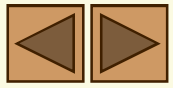
```
L2: C6;
goto L3;
L1: C3;
L3: join count;
C7;
```

Ver grafo de precedência correspondente



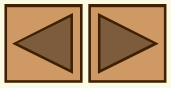
Fork/Join

- É importante notar que a execução de uma instrução **fork** divide uma única computação em duas computações independentes.
- Também é importante salientar que a execução de uma instrução **join** agrupa diversas execuções concorrentes.



Fork- Características

- ✓ criação dinâmica de processos, o que dificulta a correção do programa;
- ✓ quantidade de processos variável em cada execução;
  - Através do uso de comandos ifs, whiles, ...
- ✓ código do novo processo é o mesmo do processo pai;
  - existem variações como por exemplo nas threads ou em OO



Fork- Características

- ✓ variáveis: 2 alternativas em geral;
  - A: compartilhamento de variáveis;
  - B: duplicação das variáveis, como ocorre no Unix;
    - Valor inicial na cópia é idêntico mas depois são independentes
- ✓ estruturas do tipo goto dão pouca legibilidade ao programa;

Fork - exemplos

- SO Unix, Linux
  - Formato:

    process_id fork();
  - process_id:
    - se pai recebe id do filho
    - Se filho, recebe zero
  - Ambos os processos continuam a execução na instrução seguinte ao fork
  - Duplicação de variáveis

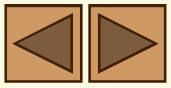
Fork - exemplos

- SO Unix, Linux
  - Uso comum

```
int pid;
pid = fork();
if pid == -1 {
    “código de erro”
} else if pid == 0 {
    “código do filho”
} else {
    “código do pai”
}
```

Join - Variações

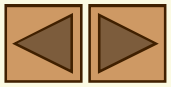
- ✓ Forma geral: agrupamento de 2 processos concorrentes
- ✓ Join “n”
  - n: número de processos a sincronizar ou terminar no join
- ✓ Join Pk
  - espera pelo término do processo Pk;
  - Pk é normalmente um processo filho;
  - é necessário que os processos estejam nomeados para sua identificação;
- ✓ Join-filho
  - espera pelo término de algum filho



Join - Variações

- Join usando exit
  - o processo que executa **join** espera pelo término de **um** filho;
  - processo filho termina por **exit;**
  - Exemplo:

```
...
I1;
fork LI3;
I2;
join;
I4;
GOTO LI5;
LI3: I3;
exit;
```



Exemplos de Implementações

- ✓ SOs Unix, Linux
- ✓ Biblioteca de programação paralela MPI

Thread

- ✓ Variação do fork/join
- ✓ Modelo mais usado atualmente para PC
  - Devido vantagens básicas:
    - Mais eficiente
    - Variáveis compartilhadas
- ✓ Inúmeras implementações
  - SOs: Unix, Linux, Windows, SOs de dispositivos móveis (PDAs, celulares), embarcados, sensores, ...
  - Bibliotecas
  - Linguagens: Java, C#, Python, ...

Thread

- Modelo processo + thread usual
  - Um processo pode conter várias threads
  - O código main (procedure, método) é uma thread
  - Uma thread tem escopo limitado ao de seu processo
    - Processo B não tem acesso a threads de processo A
  - Usualmente o escopo de um processo é limitado por um computador
    - Escopo: hw (cpu, ram, ...) mais o SO
    - Logo também o de uma thread

Thread

- ✓ Criação
  - Primitiva (procedure, método) “create_thread”
  - Argumentos
    - Procedure que contém o código da thread
    - Argumento da procedure
  - Retorno
    - Identificador da thread
  - Semântica
    - Similar à do fork
    - Thread atual continua execução concorrentemente à nova

Thread

- Morte
  - Final da procedure (ou método)
  - Exit

Thread

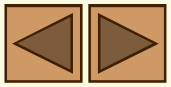
- Modelo usual de variáveis
  - Variáveis estáticas
    - Compartilhadas entre as threads de um mesmo processo
  - Variáveis locais
    - Uma cópia por thread
  - Argumentos
    - Uma cópia por thread

Thread

- ✓ Exemplos
  - Posix threads
    - Biblioteca padronizada
    - SO Unix, Linux e outros
  - Java threads
  - C# threads

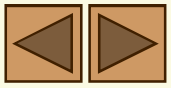
Conceito

- Parbegin/end
  - comando de alto nível para especificação da concorrência;
    - comando estruturado
  - especifica explicitamente uma seqüência de segmentos de programa que poderiam ser executados concorrentemente;
  - normalmente encontrado em algumas linguagens
    - nunca (?) usado em bibliotecas
    - exige compilação

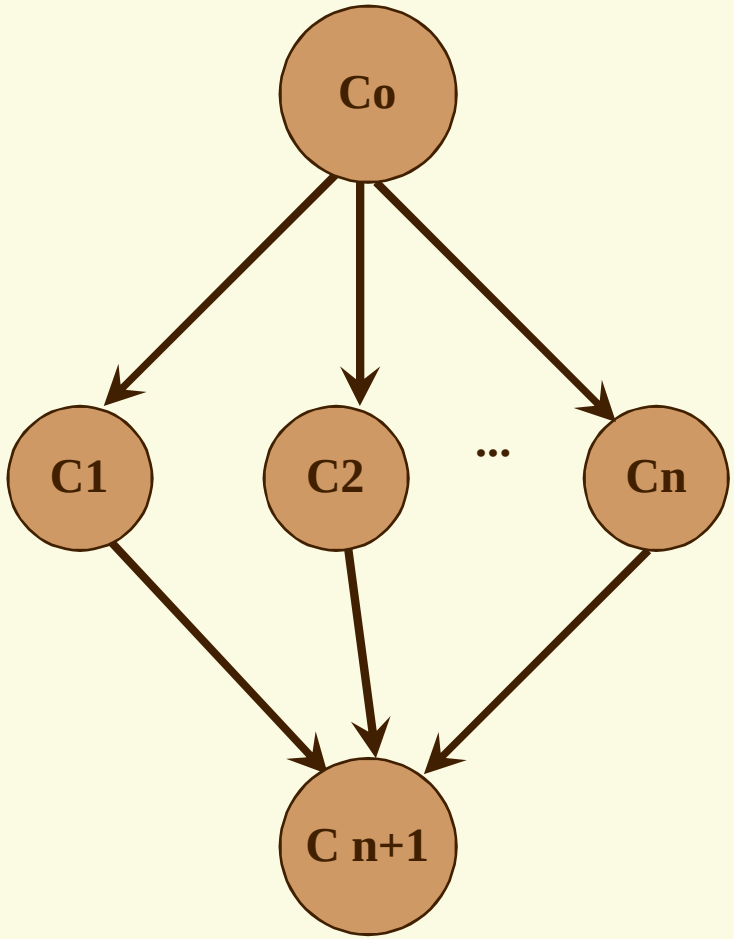


Parbegin/End

- Todos os comandos são encapsulados entre o parbegin e parend e devem ser executados concorrentemente;
- Quantidade fixa de processos (tarefas);
- Sintaxe do comando:
  - **Parbegin C**1;C2;…;Cn **Parend**
- Semântica do comando:
  - Cada Ci é um segmento de código autônomo e é executado concorrentemente.



Parbegin/End

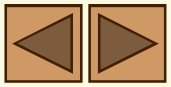


- grafo de precedência que corresponde à sintaxe dada anteriormente
- C0 e Cn+1 são os comandos que aparecem logo antes e após da/a instrução parbegin/parend, respectivamente.
- o comando Cn+1 pode ser executado somente após todos Ci i= 1,2…n , tiverem sido executados

Parbegin/End - Exemplo 1

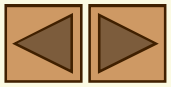
```
C1;
parbegin
    C3;
    begin
      C2;
      C4;
      parbegin
          C5;
          C6;
      parend;
   end;
parend;
C7;
```

Ver grafo de precedência correspondente



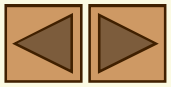
Parbegin/End - Vantagens e Desvantagens

- Vantagens:
  - Mais legível;
  - Maior facilidade na verificação de erros;
- Desvantagens:
  - Todo o grafo de precedências pode ser expresso por **fork/join**, mas nem todo o grafo de precedências pode ser representado por **parbegin/parend**.
    - Com **fork/join** é possível entrar e sair de um subgrafo
  - Dependendo da sofisticação da linguagem, é possível acontecer de não poder usar **goto** para dentro ou para fora do **parbegin/end**



Conceito

- ✓ Vetor de Processos
  - necessita a criação de um vetor de *n* posições
    - também pode trabalhar com matrizes;
  - cada processo (tarefa) executa o mesmo código
    - código é declarado dentro de um laço cujas instruções são executadas concorrentemente;
    - obs: concorrência entre tarefas!



Vetor de processos - Exemplo 1

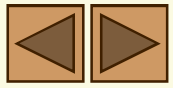
- Declaração do vetor:

```
var a[1:n]: real;          // sintaxe SR
```

- Código:

```
process inicializa (i: 1 to n)
  a[i] := 0
```

- Código de cada processo: a[i] := 0
- Função geral: inicializar o vetor “a” de forma concorrente



Vetor de processos - Exemplo 1

- Equivale a

```
fa i := 1 to n → a[i] := 0 af      // sintaxe SR
```

ou

```
for (i := 1 to n) {a[i] := 0}
```

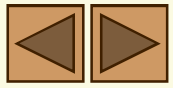
Vetor de processos - Exemplo 2

- Declaração das matrizes: (sintaxe SR)

```
var a[1:n,1:n], b[1:n,1:n]: real;
var c[1:n, 1:n]: real := ( [n], ([n] 0.0))
```

- Inicialização das matrizes:

```
fa i:= 1 to n,  j:= 1 to n →
   read(filea, a[i,j]); read(fileb, b[i,j])
af
```

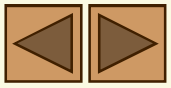


Vetor de processos - Exemplo 2 (cont.)

- Laço de execução:

```
process compute (i := 1 to n, j := 1 to n)
   fa k:= 1 to n →
      c[i,j]) := c[i,j] + a[i,k]*b[k,j]
   af
```

- Comentários:
  - criação de n^2 processos (2 laços de ordem *n*)
  - matrizes são inicializadas seqüencialmente
  - produtos são computados em paralelo



Vetor de processos - caso

- ✓ Biblioteca OpenMP
  - Padronizada
  - Para programação paralela
  - Para certos tipos de programas
  - Exige compilador especial
  - Adicionada às linguagens C, Fortran, ...

Considerações Finais

- ✓ Vários modelos para criação da concorrência
  - Modelos de programa concorrente
  - APIs para criação e gerência (join, ...) das tarefas
  - Variações na semântica das APIs
  - Variações na sintaxe das APIs (menos importante)
- ✓ Diferentes modelos afetam os outros aspectos da PC
  - Sincronização e comunicação

Considerações Finais

- ✓ Existem outros modelos
  - Nas linguagens OO
    - E com APIs distintas, ...
  - Na paralelização de linguagens seqüenciais (Fortran, Lisp, Prolog, ...)
    - Normalmente usando os modelos acima na implementação

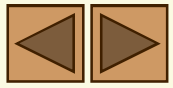
Resumo

- ✓ Tarefa, processo e thread
- ✓ Tipos: estático x dinâmico, implícito x explícito
- ✓ Modelos
  - Grafo de precedências (ou de comunicação)
  - Fork/join
  - Thread
  - Parbegin/parend
  - Array de processos

Exercício 1

- Reescreva o seguinte programa usando instruções parbegin/parend. Certifique-se de que será explorado o máximo de paralelismo e que o resultado produzido será *sempre* o mesmo da execução seqüencial.

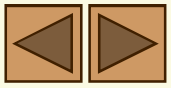
```
W := X1 * X2;
V := X3 * X4;
Y := V  * X5;
Z := V  * X6;
Y := W  * Y ;
ANS := Y + Z;
```



Exercício 1-a

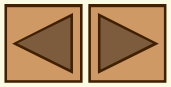
- Reescreva o seguinte programa usando instruções parbegin/parend. Certifique-se de que será explorado o máximo de paralelismo e que o resultado produzido será *sempre* o mesmo da execução seqüencial.

```
W := X1 * X2;
V := X3 * X4;
T := Y1 + Y2;
Y := V  * T;
Z := V  * X6;
W := W  * Y ;
ANS := Y + Z;
```



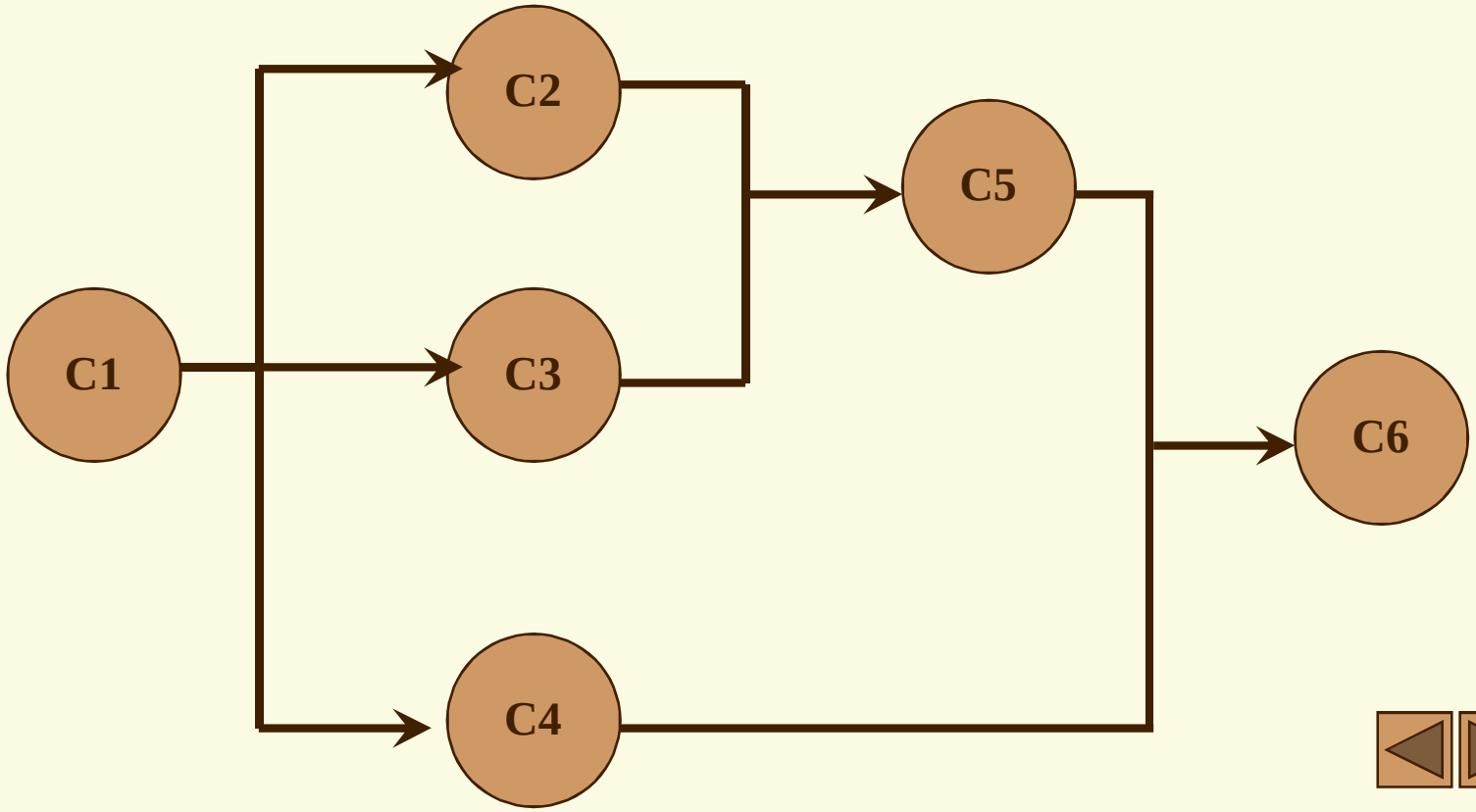
Exercício 1 - Resposta

```
parbegin
  W := X1 * X2;
  begin
    V := X3 * X4;
    parbegin
      Y := V  * X5;
      Z := V  * X6;
    parend;
  end;
parend;
Y := W  * Y ;
ANS := Y + Z;
```



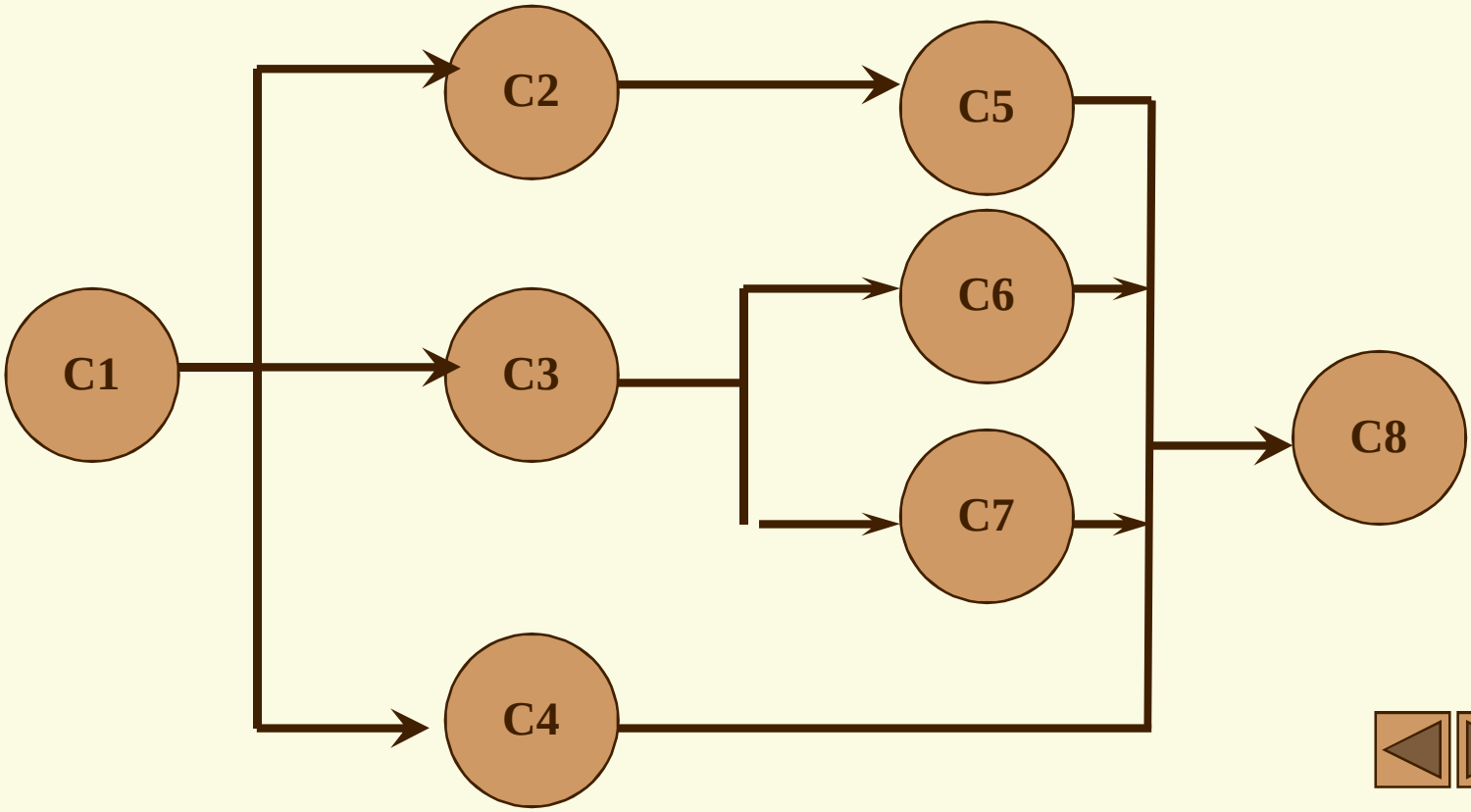
Exercício 2

- Transforme o grafo de precedência abaixo em um programa usando fork/join:



Exercício 2-a

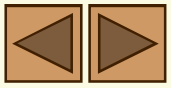
- Transforme o grafo de precedência abaixo em um programa usando fork/join:



Exercício 2 - Resposta

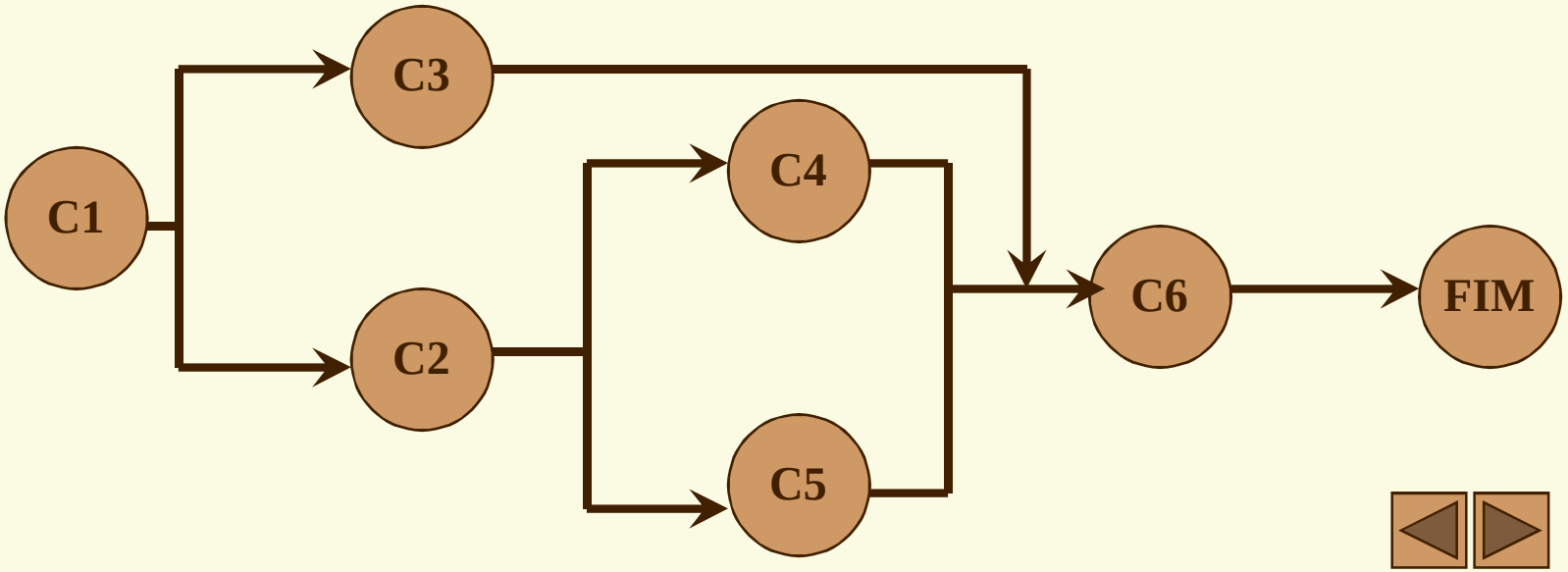
```
C1;              LC3: C3;
fork LC3;             goto LJ3;
fork LC4;        LC4: C4;
C2;                   goto LJ4;
LJ3: join;
C5;
LJ4: join;
C6;
```

Transformação do grafo de precedência em um programa utilizando instruções fork/join



Exercício 3

- Transforme o grafo de prcedência abaixo em um programa fork/join e em parbegin/parend. Se o grafo não puder ser representado por parbegin/parend, explique porquê.

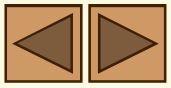


Exercício 3 - Resposta

```
C1;
fork LI3;
C2;
fork LI5;
C4;
LJ5: join;
LJ3: join;
C6;
```

```
LC3: C3;
     goto LJ3;
LI5: C5;
     goto LJ5;
```

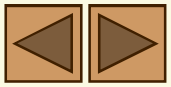
Transformação do grafo de precedência em um programa utilizando instruções fork/join



Exercício 3 - Resposta

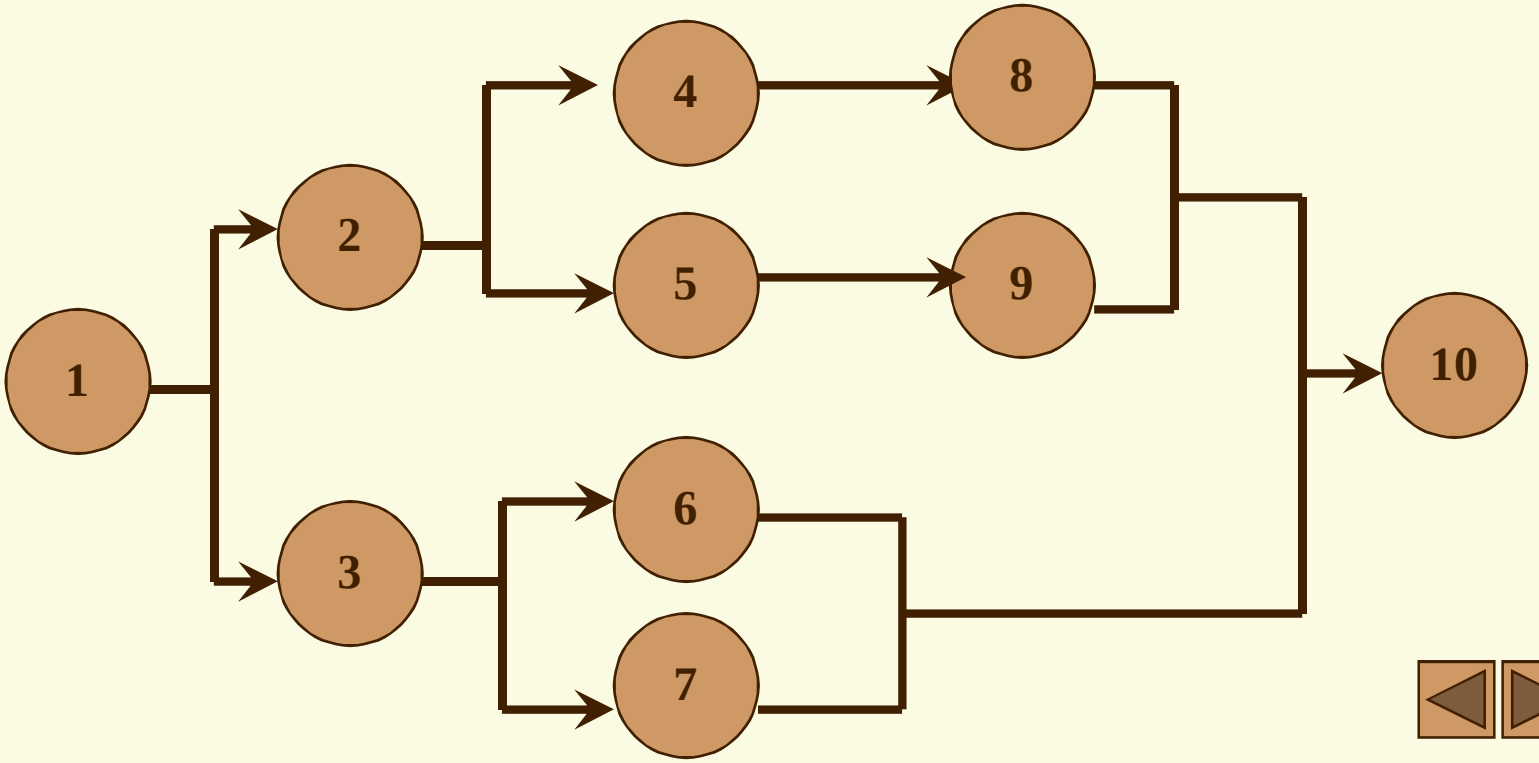
```
C1;
parbegin
   C3;
   begin
     C2;
     parbegin
        C4;
        C5;
     parend;
   end;
parend;
C6;
```

Transformação do grafo de precedência em um programa utilizando instruções parbegin/parend



Exercício 4

- ✓ Implemente o grafo de precedência abaixo utilizando fork/join. Explique a semântica do fork e do join: início do processo, quantidade de processos...



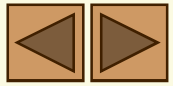
Exercício 4 - Resposta

```
1;
Fork L3;
2;
Fork L5;
4;
8;
J2: Join
J3: Join;
    10;
    exit;
```

```
L3: 3;
    Fork L7;
    6;
J1: Join;
    goto J3;
```

```
L7: 7;
    goto J1;
L5: 5;
    9;
    goto J2;
```

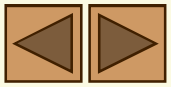
O início do processo se dá com o processo pai (1). O comando fork cria processos filhos. A quantidade de processos criados é variável. Já a semântica do processo join é esperar o término de 2 processos (join padrão) para somente depois disso passar para a próxima instrução. O código com fork/join não fica legível, mas a criação dos processos é feita de forma dinâmica . Exemplo: Fork L3 cria ó processo 3. O join de rótulo J1 espera o término da execução de 6 e 7.



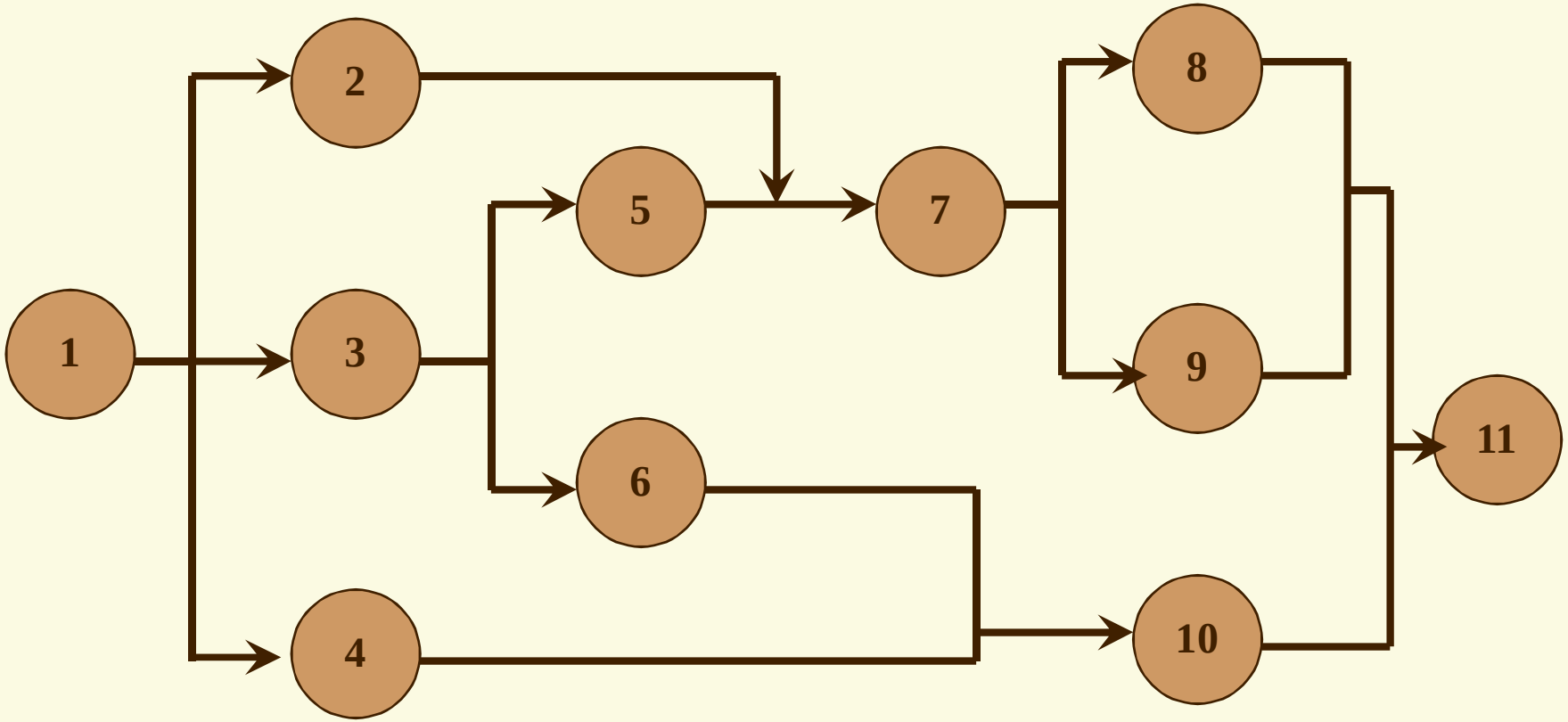
Exercício 5

- A partir do programa abaixo que utiliza fork/join, implemente o grafo de precedência correspondente.

```
    1;
    Fork L3;
    Fork L4;
    2;
    goto J1;

L3: 3;
    Fork L6;
    5;
J1: Join;
    7;
    Fork L9;
    8;
J2: Join;
J3: Join;
    11;
exit;

L4: 4;
    goto J4;

L9: 9;
    goto J2;

L6: 6;
J4: Join;
    10;
    goto J3;
```



Exercício 5 - Resposta



Revisão

- ✓ Tipos quanto a “quando”?
- ✓ Tipos quanto a “quem”?
- ✓ Combinações usuais?
- ✓ Tipos mais simples para programador?
- ✓ Tipos mais eficientes?
- ✓ Tipos mais flexíveis?

Revisão

- ✓ Conceito de grafo de precedência
  - nó?
  - arco?
  - 2 nós ligados por 1 arco?
  - 1 nó com 2 ou mais arcos de saída?
  - 1 nó com 2 ou mais arcos de entrada?

Revisão

- ✓ Conceito de fork clássico
  - função básica?
  - código?
  - variáveis?
  - início?
- ✓ Variações no fork Unix?
  - sobre acima?
  - retorna?
- ✓ Problemas de eng. de sw

Revisão

- ✓ Conceito de join clássico
  - versão 2 processos
  - versão n (2 ou mais) processos
- ✓ Variações no Unix
  - nome
  - tipo 1 (anônimo)
  - tipo 2 (com identificador)

Revisão

- ✓ Parbegin/parend
  - estático ou dinâmico?
  - implícito ou estático?
  - quais funções do modelo fork/join?
  - qual o modelo de código?
  - qual o modelo de dados?
  - o que pode ser proibido no parbegin/parend?
  - quais as vantagens?

Revisão

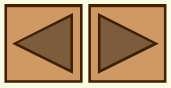
- Vetor de processos
  - estático ou dinâmico?
  - implícito ou explícito?
  - qual o código?
  - qual o modelo de dados?
  - qual a similaridade com o parbegin/parend?
  - qual a diferença com …?

Revisão

- ✓ Outras primitivas Unix
  - sobre identificadores
  - término
  - código

Bibliografia

- Andrews, G.R.; *The SR Programming Language* - Benjamin/Cummings, 1993, 1 ed., Redwood City, 344p
- Bic, L.; Shaw, A.C.; *The Logical Design of Operating Systems* - Prentice Hall, 1988, 2 ed., New Jersey, 370p.
- Peterson, J.L.; Silberschatz,A.; *Operating System Concepts* - Addison-Wesley, 1985, 2 ed., 584p.
- Toscani,S.S.; *Introdução aos Sistemas Operacionais* - Apostila, Instituto de Informática, 1996, Porto Alegre, 184p.



Especificações

Autores

Ingrid de Vargas Mito ingrid@aton.inf.ufrgs.br

Maíra Colling Wenzel maira@aton.inf.ufrgs.br

Local

Instituto de Informática

UFRGS

Disciplina: Sistemas Operacionais II

Ano: 1998/2

Revisão: 2001-2 (C. Geyer)