



- Entrega:** 14 de março de 2007
Apresentação: 14, 19, 21 de março de 2007 (a confirmar)
Metodologia: Trabalho de implementação extra-classe, a ser realizado em grupos (2 a 4 integrantes).

Tema: *Linguagem para programação concorrente*

Motivação: Os limites tecnológicos atingidos com os recursos atuais na criação de processadores mais velozes motivou pesquisas na área das arquiteturas paralelas. Destas pesquisas, hoje, encontramos no mercado computadores construídos segundo o modelo de arquitetura multiprocessada a preços bastante competitivos. Em particular, computadores SMPs (Symetric MultiProcessors) e processadores multi-core são opções de compra oferecidas até mesmo a usuários domésticos. Como resultado prático, uma nova gama de aplicações deve ser desenvolvida, de forma a tirar proveito destes recursos computacionais. Neste sentido, uma nova geração de ferramentas de programação paralela deve ser desenvolvida, refletindo esta nova e crescente demanda.

Definição: Especificar e implementar uma linguagem para programação concorrente em arquiteturas com memória compartilhada. Esta linguagem pode ser original ou ser uma extensão de uma linguagem de programação imperativa existente (como C ou Pascal). Deve ser implementado um compilador capaz de traduzir o código da linguagem especificada em uma outra linguagem de alto nível, este código intermediário gerado deve ser compilável pelo compilador desta linguagem intermediária. **Atenção:** no programa intermediário, a concorrência pode ser simulada através de chamadas de funções ou implementada de fato, utilizando alguma ferramenta de programação concorrente.

Especificação: Esta linguagem deve oferecer o seguinte conjunto mínimo de recursos de programação:

Integer	tipo de dado inteiro
Double	tipo de dado ponto flutuante
Fluxo	tipo de dado fluxo
[]	array unidimensional
if – then – else	estrutura de condição simples
while	estrutura de laço
Split	criação de novo(s) fluxo(s) de execução
Merge	junção de fluxos de execução

Observe que ainda devem ser oferecidos recursos para operações aritméticas (somadas, divisões, atribuições, etc.). Também deve ser oferecida a possibilidade de definir sub-programas (funções) que possam ser identificadas para criação de novos fluxos de execução. Sugere-se que funções sejam definidas através da seguinte sintaxe:

```
TIPO NOME( TIPO par )
CORPO
```

Fluxo: Tipo de dado que armazena informações sobre o novo fluxo criado. Uma variável deste tipo deve ser utilizada para manter informações sobre os fluxos de execução e somente deve ser manipulada pelas operações **Split** e **Merge**.

Split: Operação de criação de novo(s) fluxo(s) de execução. A sintaxe deste operador pode ser exemplificada como segue (também esta sendo exemplificado o **Merge**):

1	Fluxo t; Integer dados, res; t = Split(func, dados); ... Merge(t, res);
2	Fluxo t[4]; Integer dados[4], res[4]; t = Split(func, dados); ... Merge(t, res);
3	Fluxo t[4]; Integer dados[4], res; t = Split(func1, func2, func3, func4, dados); ... Merge(t, res);
4	Fluxo t[4]; Integer dados1, dados2, dados3, dados4, res1, res2, res3, res4; t = Split(func1, dados1, func2, dados2, func3, dados3, func4, dados4); ... Merge(t, res1, res2, res3, res4);

Nestes exemplos, a operação **Split** se dá da seguinte forma:

1	<p>t corresponde a um identificador de fluxo. dados a um dado inteiro. func ao identificador de um subprograma que recebe um inteiro como entrada. A função Split cria um novo fluxo de execução e coloca em t a identificação do fluxo criado.</p>
2	<p>t corresponde a um array de identificadores de fluxos. dados a um array inteiros. func ao identificador de um subprograma que recebe um array de inteiros como entrada. A função Split cria 4 novos fluxos de execução, todos executando o mesmo código definido por func, e coloca em t a identificação do fluxo criado, um sobre cada posição.</p>
3	<p>t corresponde a um array de identificadores de fluxos. dados a um array inteiros. func[1-4] aos identificadores de um subprograma que recebem um array de inteiros como entrada. A função Split cria 4 novos fluxos de execução, executando, respectivamente, func[1-4], e coloca em t a identificação do fluxo criado, um sobre cada posição. Cada novo fluxo recebe uma cópia do dado fornecidos como argumento.</p>
4	<p>t corresponde a um array de identificadores de fluxos. dados[1-4] valores inteiros. func ao identificador de um subprograma que recebe um valor inteiro como entrada. A função Split cria 4 novos fluxos de execução, executando, respectivamente, func[1-4], e coloca em t a identificação do fluxo criado, um sobre cada posição. Cada novo fluxo recebe uma cópia do dado fornecidos como argumento.</p>

Merge: Operação de união de dois (ou mais) fluxo de execução. A sintaxe deste operador encontra-se exemplificada junto ao operador **Split**. Nestes exemplos, a operação se dá da seguinte forma:

1	<p>res corresponde a um dado inteiro. A função Merge sincroniza o fluxo de execução que executou esta operação com o fluxo de execução identificado por t e coloca o resultado da computação de t em res.</p>
2	<p>res corresponde a um array de valores inteiros. A função Merge sincroniza o fluxo de execução que executou esta operação com todos os fluxos de execução identificados pelo array t e distribui os resultados da computação de t[i] em res[i].</p>
3	<p>res corresponde a um dado inteiro. A função Merge sincroniza o fluxo de execução que executou esta operação com todos os fluxos de execução identificado pelo array t e coloca em res o resultado da operação += de C. Como resultado final, res conterá a acumulação dos valores retornados pelos fluxos de execução sincronizados.</p>
4	<p>res1, res2, res3, res4 correspondem a valores inteiros. A função Merge sincroniza o fluxo de execução que executou esta operação com todos os fluxos de execução identificado pelo array t e coloca em res1, res2, res3, res4, respectivamente, o resultado da computação de cada um dos fluxos de execução sincronizados.</p>

Material a ser entregue:

- Documentação:
 - Manual de programação da linguagem especificada;
 - Exemplos de programas e seus respectivos resultados;
 - Expressões regulares para os lexemas da linguagem;
 - Regras semânticas;
 - Manual de desenvolvimento do compilador construído:
 - Fluxos de dados entre as partes;
 - Estratégias empregadas para tabela de símbolos;
 - Ações semânticas adotadas.
- Arquivos:
 - Compilador;
 - Programas exemplos.

Recursos a serem utilizados:

- Lex/Yacc (Flex/Bison) ou JavaCC
- Threads POSIX

Avaliação:

A presente proposta de trabalho apresenta uma especificação genérica para uma linguagem de programação concorrente, os grupos que refletirem em suas implementações esta especificação e apresentarem uma documentação adequada serão considerados aprovados neste trabalho. Diversas questões, no entanto, estão deixadas em aberto: adição de diferentes tipos de dados, inclusão de diferentes estruturas de controle de fluxo de execução seqüencial como switch e for, e ainda outras formas de manipulação dos fluxos de execução e dos dados transferidos entre estes. Toda extensão a presente especificação será considerada e valorizada.