BCC202 – Estruturas de Dados I (2015-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG

Trabalho Prático 01 (TP01)

- **Data de entrega: 13/12/2015 até 23:55. O que vale é o horário do Moodle, e não do *seu*, ou do *meu*, relógio!!!**

- O padrão de entrada e saída deve ser respeitado **exatamente** como determinado no enunciado.

- Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Bom trabalho!

1 Objetivos

Este trabalho prático tem como objetivo principal fundamentar os conceitos de Tipos Abstratos de Dados (TADs), alocação dinâmica de memória e recursividade.

2 Execírcio 1 - Definição do Problema

O laboratório **HPC**, do **DECOM**, trabalha com computação de alto desempenho. Um dos problemas abordados é o escalonamento de processos entre diferentes computadores, gerenciando os recursos demandados por cada processo e disponíveis no sistema.

Simulação é uma técnica muito utilizada para avaliação de desempenho de sistemas de computação. Um simulador cria uma abstração do sistema alvo, capturando seus aspectos mais relevantes para o estudo em questão. Neste trabalho prático, o seu objetivo é implementar as estruturas de dados que serão utilizadas no gerenciamento de recursos em um sistema distribuído chamado **ProSys**. A fim de validar as estruturas de dados definidas, uma estratégia de gerenciamento simples será implementada.

O sistema **ProSys** é uma plataforma distribuída para armazenamento e processamento de dados. Ele é composto de `N` **computadores** interligados por uma rede de alta velocidade. Cada computador contém um conjunto de **recursos** (*CPU*, *memória* e *discos*). Os **processos** a serem executados no sistema são compostos de **tarefas**. Cada tarefa demanda uma certa quantidade de recursos específicos.

Primeiramente, você deverá implementar as TADs que representam o sistema ProSys, descritas na Seção 2.1. Em seguida, você deve implementar um escalonador de processos simples, descrito na Seção 2.2.

2.1 TADs do sistema

Cada TAD necessário para a implementação do sistema é definida a seguir. Você pode implementar campos e funções adicionais nestes TADs, ou mesmo implementar outros TADs complementares, que julgar necessários para uma implementação mais eficiente e correta do ProSys.

`TRecurso`: Representa um recurso de um computador. Pode ser de três tipos:

1. **CPU**: recurso capaz de realizar o processamento de instruções. A capacidade de CPU é a quantidade de processos que podem estar em espera ou em execução.
2. **Memória**: recurso capaz de armazenar os dados manipulados durante a execução de um processo. A capacidade de memória é definida pela quantidade de Giga Bytes que podem ser armazenados.
3. **Disco**: recurso utilizado para armazenar dados de entrada e saída da execução de um processo. A capacidade do disco é definida pela quantidade de Giga Bytes que podem ser armazenados.

O TAD **`TRecurso`** deve possuir os seguintes campos:

- **`Tipo`**: inteiro que define o tipo de recurso (de acordo com a numeração anterior).
- **`Capacidade`**: double que define a capacidade do recurso em questão.
- **`EmUso`**: double que define a quantidade do recurso atualmente em uso.
- **`Velocidade`**: double que indica o tempo de acesso/execução para uma unidade de medida do recurso (tempo necessário para fazer acesso a **um byte** de armazenamento ou o tempo de execução de uma única instrução).

E as seguintes operações:

- **`Inicializa`**: aloca dinamicamente memória para um recurso e define valores iniciais para seus campos.
- **`Get`*`Campo`*** e **`Set`*`Campo`***: funções *get* e *set* para cada campo.
- **`EstaDisponivel`**: verifica a disponibilidade de atendimento do recurso. Recebe um inteiro com a demanda pelo recurso e retorna `1` caso haja capacidade para atender à demanda e `0` caso contrário.
- **`Reservar`**: realiza a reserva por uma determinada quantidade de recurso. Recebe um inteiro com a demanda pelo recurso e retorna `1` caso consiga realizar a reserva e `0` caso contrário. Realizar a reserva significa aumentar o valor do campo **`EmUso`** de acordo com a demanda informada.
- **`Liberar`**: libera a reserva de uma determinada quantidade de recurso. Recebe um inteiro com a demanda pelo recurso e retorna `1` caso consiga liberar a reserva e `0` caso contrário. Realizar a liberação significa reduzir o valor do campo **`EmUso`** de acordo com a demanda informada.
- **`CalcularTempo`**: calcula o tempo necessário de uso do recurso para uma determinada demanda recebida como argumento: **`Velocidade`** * *demanda*.

`TTarefa:` Representa uma tarefa a ser executada em um processo. Deve possui os seguintes campos:

- **`Recurso`**: inteiro que determina o recurso necessário para a execução da tarefa. Utiliza os mesmos códigos definidos para o campo **`Tipo`** do TAD **`TRecurso`**.
- **`Demanda`**: double que define a quantidade de recurso necessário para a execução da tarefa. A *Demanda* de uma tarefa da CPU corresponde ao número de intruções do processo a ser executado.
- **`Concluido`**: determina se a tarefa já foi executada. Recebe o valor `1` caso já tenha sido concluída e valor `0` caso contrário.

E as seguintes operações:

- **`Inicializa`**: aloca dinamicamente memória para uma tarefa e define valores iniciais para seus campos.
- **`Get`*`Campo`*** e **`Set`*`Campo`***: funções *get* e *set* para cada campo.
- **`Executar`**: executa a tarefa, alterando adequadamente o valor do campo **`Concluido`**.

`TProcesso:` Representa um processo a ser executado. Deve possuir os seguintes campos:

- **`Tarefas`**: vetor de ponteiros para tarefas (**`TTarefa**`**) que constituem o processo. A ordem de execução das tarefas é a mesma definida na entrada e não pode ser alterada durante a execução.
- **`Concluido`**: determina se o processo já foi executado. Recebe o valor `1` caso já tenha sido concluído e valor `0` caso contrário.

E as seguintes operações:

- **`Inicializa`**: aloca dinamicamente memória para um processo e define valores iniciais para seus campos. Recebe como argumento a quantidade de tarefas que compõem o processo.
- **`Get`*`Campo`*** e **`Set`*`Campo`***: funções *get* e *set* para cada campo.
- **`AddTarefa`**: adiciona uma nova tarefa ao vetor de tarefas. Recebe um **`TTarefa*`** como argumento e a insere na primeira posição vazia do vetor de tarefas.

- **`Executar`**: executa sequencialmente todas as tarefas que compõem o processo. Ao término da execução de todas as tarefas o processo deve ser marcado como concluído (alteração do campo **`Concluido`**). Cada tarefa deve ser manipulada adequadamente para o registro de sua execução, ou seja, as funções **`Iniciar`** e **`Concluir`** das tarefas devem ser invocadas adequadamente.

`TComputador:` Representa um computador do ProSys. Deve possuir os seguintes campos:

- **`Recursos`**: vetor de ponteiros para recursos (**`TRecurso**`**) disponíveis no computador.
- **`Processos`**: vetor de ponteiros para processos (**`TProcesso**`**) escalonadas para serem executadas no computador.

E as seguintes operações:

- **`Inicializa`**: aloca dinamicamente memória para um computador e define valores iniciais para seus campos. Recebe como argumento a quantidade de recursos e processos que podem ser associados ao computador.
- **`addRecurso`**: adiciona um novo recurso ao vetor de recursos. Recebe um **`TRecurso*`** como argumento e a insere na primeira posição vazia do vetor de recursos.
- **`addProcesso`**: adiciona um novo processo ao vetor de processos. Recebe um **`TProcesso*`** como argumento e o insere na primeira posição vazia do vetor de processos. Também realiza a reserva dos recursos demandados para a execução de cada tarefa do novo processo.
- **`Executar`**: executa sequencialmente todos os processos escalados para o computador. A cada processo executado deve liberar os recursos demandados por suas tarefas.

2.2 Escalonador do sistema

O escalonador do sistema determina em qual computador um processo deverá ser executado. Esta tarefa será realizada da seguinte maneira:

1. A busca por um computador é sequencial, na ordem em que os computadores forem definidos na entrada.
2. O processo é associado ao primeiro computador que seja capaz de executá-lo:
   (a) Existe uma CPU com capacidade de atender a mais um processo.
   (b) Há memória suficiente para realizar todas as tarefas do processo.
   (c) Há disco suficiente para realizar todas as tarefas do processo.

2.3 Entrada e Saída

A seguir, um exemplo de entrada e saída, os comentários foram inseridos apenas para entendimento, não fazem parte do conteúdo do arquivo de entrada a ser utilizado nem da saída a ser gerada.

Arquivos de entrada e saída para realização de testes serão fornecidos no site da disciplina.

Entrada	Saida

2 // Numero de computadores
 	
// Computador 1
3 5 // Qtd recursos e qtd processos
1 5.0 0.5 // CPU, capacidade, velocidade
2 8.0 0.8 // Memoria
3 20.0 1.6 // Disco
 	
// Computador 2
5 10 // Qtd recursos e qtd processos
1 5.0 0.3 // CPU1, capacidade, velocidade
1 5.0 0.3 // CPU2, capacidade, velocidade
2 8.0 0.5 // Memoria
3 20.0 1.1 // Disco 1
3 20.0 1.5 // Disco 2
 	
3 // Quantidade de processos
 	
// Processo 1
3 // Quantidade de tarefas do processo
2 2.0 // Tarefa 1: Memoria, demanda (GB)
1 1000.0 // Tarefa 2: CPU, demanda
3 15.0 // Tarefa 3: Disco, demanda (GB)
 	
// Processo 2
4 // Quantidade de tarefas do processo
2 3.5 // Tarefa 1: Memoria, demanda (GB)
1 5000.0 // Tarefa 2: CPU, demanda
3 10.0 // Tarefa 3: Disco, demanda (GB)
3 5.0 // Tarefa 4: Disco, demanda (GB)
 	
// Processo 3
3 // Quantidade de tarefas do processo
2 4.0 // Tarefa 1: Memoria, demanda (GB)
1 15000.0 // Tarefa 2: CPU, demanda
3 18.0 // Tarefa 3: Disco, demanda (GB)
	// Computador: qtd. proc., tempo total
	Computador 1: 1, XXX.XXXX
	Computador 2: 2, WWWW.WWWW
	
	// Processo: Alocacao, tempo de execucao
	Processo 1: Computador 1, XXX.XXXX
	Processo 2: Computador 2, YYYY.YYYY
	Processo 3: Computador 2, ZZ.ZZZZ

3 Exercício 2

Uma imagem discreta de largura w e altura h, pode ser representada em um computador através de uma matriz I$[i, j]$, de ordem wxh, que armazena em cada posição um número inteiro entre 0 e 255, o qual especifica uma certa cor em uma paleta de cores. Em pacotes de pintura interativos, é muito comum a operação que efetua o preenchimento de certa área de uma imagem com uma cor cant com uma nova cor c. Esta operação pode ser realizada de forma simples através de um método denominado Boundaryfill. O procedimento em questão recebe como entrada um ponto no interior da região especificado por índices (x,y) e a cor de preenchimento c. O algoritmo inicialmente detecta a cor cant no ponto (x,y) e começa pintando tal posição com a cor c1 caso c ! = cant. O processo é repetido recursivamente para os vizinhos acima I$[x+1, y]$, abaixo I$[x1, y]$, à esquerda I$[x, y1]$ e à direita I$[x, y+1]$ desde que estejam dentro da imagem e possuam cor igual a cant, isto é, igual a cor a ser substituída. Escreva o procedimento que implemente tal algoritmo.

4 Imposições e comentários gerais

- É fundamental que sejam respeitados os conceitos de: TAD.
- Seu programa não pode ter "*memory leaks*", ou seja, toda memória alocada deve ser liberada pelo seu código.
- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.
- Trabalhos copiados (e FONTE) terão nota zero, além de os alunos envolvidos no plágio perderem toda a nota atribuída a participação e pontos extras, entre outros (...). **Isto vale para qualquer entregável** (seção 5).
- Caso seja necessário, alunos poderão ser convocados para entrevista.

5 Entregáveis

Deverão ser entregues para avaliação do trabalho:

- **Código fonte**: código fonte do programa em `C++` (dois diretórios, um contendo o código fonte para o 'Exercício 1'e, outro, o código fonte do 'Exercício 2'.
- **Arquivos de entrada e saída**: arquivos de entrada e saída utilizados para testar cada um dos seus programas. Procure elaborar conjuntos de testes abrangentes, envolvendo cenários variados.
- **Documentação**: Para cada uma das duas questões, a documentação deve ser entregue em um único arquivo `PDF` , e conter:
  1. **Implementação**: descrição da implementação do programa. Não faça "print screens" de telas e não inclua o código fonte. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  2. **Impressões gerais**: descreva o seu processo de implementação deste trabalho. Aponte coisas que gostou bem como aquelas que o desagradou. Avalie o que o motivou, conhecimentos que adquiriu, entre outros.
  3. **Conclusão**: conclusões e comentários gerais sobre o trabalho.

5.1 Como fazer a entrega

Verifique se seus dois programas compilam e executam na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via Moodle um arquivo `.ZIP` com o padrão de nome **PrimeiroNome-UltimoNome-TP1.zip** contendo, para cada questão ('Exercício 1'e 'Exercício 2'), uma pasta de mesmo nome (sem a extensão *.zip*), com o seguinte conteúdo:

- Pasta **fonte**: esta pasta deve conter apenas os arquivos `.c` e `.h`, utilizados na implementação, e os arquivos `.TXT` de entrada e saída, utilizados para testar seu programa.
- Arquivo **PrimeiroNome-UltimoNome-TP1QuestaoX.pdf**: este arquivo `.PDF` deve conter a documentação do questão (Exerício 1 ou Exercício 2).