**Ideias Utilizadas**

No que se refere à implementação do código da disciplina de Sistemas Operacionais do Trabalho com o objetivo de simular um servidor web que atenda requisições, seguem-se as idéias utilizadas para a implementação do mesmo.

No método man do programa é utilizado um for que roda 60 vezes como o intuito de simular os sessenta segundos desejados na descrição do trabalho. A cada interação é realizada uma solicitação para que se possa gerar uma nova requisição, e na sequência é realizada a chama de uma Thread.Sleep com parâmetro de 1000 milésimos, ou seja 1 segundo.

Ao realizar a chamada para uma nova requisição o programa conta com um pool de 15 Threads na classe servidor afim de atender a cada chamada realizada. Na classe Servidor foram utilizadas os atributos threadsIniciadas, threadsConcluidas, threadsAtendidas e threadsNegadas visando contabilizar as informações desejadas no trabalho. Além de um random para determinar o tempo de CPU da requisição e um object locker para controlar o Monitor utilizado as sessões criticas do programa. Os métodos NovaRequisicao( inicia uma nova requisição ), GestorDeRequisicoes (verifica se há espaço para a requisição no vetor de Threads, se sim é realizada a gravação no arquivo de texto out.txt) e o método ImprimeVetor (mostra as requisições sendo inicializadas e terminadas).

Dentro do método GestorDeRequisicoes há um laço de repetição for que percorre todo o pool (vetor de Requisições) procurando uma posição vazia ou uma Requisição que possua o atributo *ativa* igual a *false*, caso ele percorra todo o vetor e não encontre uma posição em uma destas condições esta Thread é considerada negada, caso contrário ela é considerada atendida e ganha seu atributos, como identificação, e tempo que deve durar, então é invocado o método para escrever estas informações no arquivo de saída solicitado (*out.txt*). Após isso a Thread é suspensa pelo tempo de sua execução. Ao fim deste tempo ela invoca o método de escrita novamente para registrar seu fim no arquivo de saída, então ela atualiza os contadores, atualiza o atributo *ativa* para indicar que a execução foi concluída e que já é possível atribuir outra Thread à posição em que ela se encontrava.

Por fim o comando *break* encerra a execução do laço de repetição para evitar que ele continue procurando colocação depois de já ter conseguindo um encaixe.

A classe de Requisição conta com os atributos id, contador e ativa. E os métodos de GetID, e Get e Set Ativa (formas de verificar se a Thread esta ativa ou não em determinado momento pelo servidor).

**Bibliotecas Pesquisadas**

As bibliotecas utilizadas no trabalho foram: Threading e IO.

De modo que da biblioteca de Threading foram utilizados as classes Monitor e Threads, e destas os seguintes métodos:

* Monitor.Enter: para bloquear um recurso informando que outras Threads não podem acessá-lo até que a execução termine
* Monitor.Exit: para desbloquear um recurso informando que não há nenhum outro processo atuando sobre ele no momento e, por tanto, pode ser acessado por quem necessitar
* New Thread: Instancia uma nova Thread no sistema
* Thread.Sleep: para paralisar o processamento de uma Thread durante um tempo determinado.
* Thread.Star: para iniciar a execução de um método/processo, associado à determinada Thread.

No caso da biblioteca System.IO utilizada com o objetivo de gerar o arquivo de saída (*out.txt*). Para tanto foram aplicados os seguintes recursoso:

* FileStream: que cria um fluxo para um arquivo definido
* StreamWriter: que estabelece que informações serão incluídas no referido arquivo
* StreamWriter.WriteLine: insere as informações fornecidas no conteúdo do arquivo
* Funções *.close*: para fechar os recursos abertos

**Dificuldades Experimentadas**

Podemos perceber que pelo fato da maioria das aulas serem teóricas e só havendo tido uma aula de laboratório, foram muitas as duvidas apresentadas ao realizar a implementação do código, não pela complexidade do problema, mas, pelo fato de compreender a exatidão do código estar de fato implementado da maneira correta. Neste quesito os testes (próximo tópico a ser abordado) realizados nos guiaram na realização do trabalho.

A falta de familiaridade ao implementar um pool de Threads, em conjunto da falta de experiência de entender de fato se o trabalho estaria sendo feito da maneira correta. O fato de nunca termos trabalhado com computação paralela também foi um fator que destacamos nas dificuldades apresentadas, visto que apesar da compreensão teórica do assunto o contato com o código agrega um conhecimento diferencial aos estudos, que por assim dizer ocasionou tantas duvidas na realização do mesmo. Acreditamos que se tais questões fossem também exercícios praticados no decorrer das aulas, teria contribuído para uma melhor compreensão e execução deste trabalho.

**Testes**

Para validar o código foram realizados testes que intentavam causar falhas na execução, aumentamos o número de chamadas, removemos o intervalo entre as chamadas, alteramos o tamanho do pool de Requisições para verificar se em algum momento as variáveis de controle apresentariam valores inconsistentes.

Ao encontrar alguma inconsistência elevávamos o nível de limitação no paralelismo para evitar que voltasse a ocorrer.

Implementamos o método que imprime o vetor de Requisições com a identificação da Requisição e o estado dela para verificar o comportamento do mesmo, assim pudemos detectar o funcionamento dos algoritmos com mais facilidade.

Implementamos a escolha da forma de visualização do console para permitir que fosse possível alternar facilmente entre a listagem das ações das Threads criadas e a impressão do estado do vetor para facilitar a conferência do funcionamento.