Pontifícia Universidade Católica do Rio de Janeiro  
INF 2545 – Sistemas Distribuídos Professora: Noemi 2010.1

**Trabalho 1**Tipos de Servidor

**Alunos:**  
Danilo Moret  
Thiago Manhente de C. Marques

# Experimentos propostos

Nossa proposta original era realizar todos os testes duas vezes, uma com o servidor rodando em ambiente Linux e outro em ambiente Windows. Porém, tivemos problemas para colocar o LuaPosix para funcionar no Windows, e optamos então por realizar ambos os experimentos em ambientes Linux (Ubuntu e Kubuntu)

Os ambientes em que o servidor rodará serão os seguintes:

Virtualbox Ubuntu 64, RAM 2GB sobre host Intel(R) Core(TM)2 Duo CPU P8700 @ 2.53GHz, RAM 8GB

Kubuntu sobre Intel® Core™ 2 Duo CPU T8100 @ 2.10GHz, RAM 2GB

As seguintes configurações de servidor serão utilizadas:

Servidor monoprocesso;

Servidor multiprocesso com alocação sob demanda;

Servidor multiprocesso com pré-alocação de 5 processos; e

Servidor multiprocesso com pré-alocação de 10 processos.

Para cada tipo de servidor, serão testados os seguintes cenários de demanda:

Demanda baixa: 1 e 5 processos clientes acessando o servidor simultaneamente.

Demanda média: 10 processos clientes acessando o servidor simultaneamente.

Demanda alta: 15 processos clientes acessando o servidor simultaneamente.

Para cada par de configurações servidor-demanda acima, testaremos o envio de arquivos com os seguintes tamanhos:

10KB, 100KB, 1MB, 10MB, 100MB.

Teremos assim, no total, 160 experimentos:

4 configurações de servidor \* 5 tamanhos de arquivos\* 4 cenários de demanda \* 2 ambientes.

Em cada teste, as seguintes propriedades serão medidas:

Tempo total para transferência dos arquivos.

Maior tempo de transferência dos arquivos.

Tempo médio de transferência dos arquivos.

Ocorrência de timeouts nos clientes.

Número médio de clientes atendidos por minuto pelo servidor (throughput).

# Resultados esperados

Esperamos que o servidor monoprocessado tenha um bom desempenho com a conexão de poucos clientes, visto que não possui sobrecarga de criação de processos e trocas de contexto. Esperamos, porém, que esse desempenho vá decaindo vertiginosamente conforme o número de clientes cresça, visto que ele só pode atender a um de cada vez.

Esperamos também que os servidores não apresentem grandes diferenças de desempenho para arquivos de tamanho bem reduzido, por ser extremamente rápida a resposta do servidor para o cliente.

# Resultados auferidos

Nas próximas páginas estão as tabelas com os tempos auferidos. Os resultados vem do processamento dos logs gerados pelo programa cliente.

Alguns dos testes propostos não puderam ser realizados devido a travamentos nas máquinas que rodavam o servidor. Os testes que não puderam ser realizados foram:

Servidor multiprocessos sob demanda transferindo arquivos de 100MB para 10 e 15 clientes (em ambos os ambientes de teste)

Servidor multiprocessos pré-alocados (5 e 10 processos) transferindo arquivos de 100MB para 10 e 15 clientes (no ambiente VirtualBox Ubuntu)

Tempos de transferência médios



Tempos de transferência máximos



Tempos de transferência totais



# Conclusões quanto aos resultados

## Avaliação Geral

Em geral, os resultados auferidos confirmaram as expectativas quanto ao desempenho dos servidores para poucos e muitos clientes e quanto à influência do tamanho do arquivo no desempenho do servidor.

Considerando arquivos com mais de 1MB, o servidor monoprocessado teve um melhor desempenho para pouca demanda, apresentando tempos médios de resposta menores que os demais tipos para 1 e 5 clientes. Teve, porém, desempenho inferior aos servidores pré-alocados para 10 e 15 clientes, apresentando tempos médios de resposta maiores.

Os resultados auferidos também comprovaram que, para transações em que a resposta do servidor é bastante rápida, como no caso de arquivos pequenos (até 1MB), a diferença entre os tipos de servidores é imperceptível. Nesses casos, uma implementação mais rebuscada não se justifica, podendo se optar tranquilamente por uma implementação simples de servidor monoprocesso sem nenhuma perda considerável de desempenho.

No caso específico de servidores de arquivos, a maior vantagem de se ter multiprocessos é poder sobrepor o tempo de leitura do arquivo com a execução dos outros processos. Para arquivos pequenos, como o tempo de leitura é bastante reduzido, essa vantagem não é significativa.

## Multiprocessos em computadores com um único processador

Os ambientes de teste possuíam ambos apenas um processador. Embora possuíssem mais de um núcleo, ambos os ambientes observaram queda de desempenho quando o número de processos aumentava além de determinados níveis. Cremos que isso se dê pela grande concorrência que se estabelece sobre o único processador, levando os vários processos a serem escalonados com muita rapidez e gerando uma carga extra de trocas de contexto, justificando assim a perda de desempenho.

Testes adicionais em máquinas com vários processadores seriam recomendados, porém não dispúnhamos de nenhum ambiente com tal característica.

## Criação de processos sob demanda

Ao observarmos a dificuldade que tivemos de executar os casos de teste com grande número de clientes para o servidor sob demanda, concluímos que a criação desses processos num ambiente de alta concorrência pode levar a estagnação do servidor, por gerar uma grande sobrecarga de trabalho para a criação dos processos.

Uma solução possível é misturar os conceitos de alocação sob demanda e pré-alocação, da seguinte maneira: Pré-aloca-se um determinado número de processos no servidor. Ao verificar que a demanda está crescendo e chegando próximo ao limite dos processos alocados, realiza-se uma nova pré-alocação de mais um determinado número de processos para atender a crescente demanda. Analogamente, observando-se a diminuição da demanda, poder-se-ia terminar um bloco de processos para liberar recursos no servidor (em especial, tempo de escalonamento dos processos no processador).

### Questão de processos-zumbis

Durante testes com o servidor multiprocesso sob demanda no ambiente Kubuntu, percebemos que os processos-filhos que terminavam de enviar os arquivos ao invés de serem encerrados passavam para o estado processo-zumbi.

Pesquisamos e vimos que isso pode ser causado em algumas versões do Unix/Linux quando o processo-pai ignora o sinal de término do processo-filho, e que poderia ser revertido usando a função signal() para configurar SIGCHLD para SIG\_IGN. Não fizemos esse tratamento por não descobrirmos um jeito de fazê-lo em Lua, visto que a biblioteca LuaPosix aparentemente não fornece suporte para essa função signal().

Nos testes que realizamos, cremos que a existência desses processos-zumbis não tenha prejudicado tanto o desempenho. Porém, em um ambiente de maior escala, essa questão é importante por abrir a possibilidade de a tabela de processos ser poluída com esses processos.