UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2013/1
TRABALHO 3: COMUNICAÇÃO INTER-PROCESSOS USANDO SOCKETS UNIX

---

1. ESPECIFICAÇÃO DO TRABALHO:

O trabalho consiste em implementar em C (não em C++) uma aplicação de chat usando a API de sockets Unix. O programa deverá executar obrigatoriamente em ambientes Unix (Linux) mesmo que o mesmo tenha sido desenvolvido sobre outras plataformas.

Uma descrição sobre comunicação entre processos utilizando sockets é apresentada em vários livros de sistemas operacionais e de programação concorrente. Em caso de dúvidas, é possível consultar os livros do Silberchatz (Operating System Concepts, 8th edition, seção 3.6) e Coulouris (Distributed Systems Concepts and Design, 5th edition, seção 4.2). Ainda, na Internet se encontram diversos tutoriais (alguns deles mencionados nas notas de aula) que descrevem em detalhes a API de sockets Unix.

2. SERVIDOR DE CHAT

Um servidor de chat deve permitir até um certo número de aplicações cliente se conectarem ao servidor, e facilitar a troca de mensagens entre o grupo de clientes que estejam utilizando o chat simultaneamente. O programa deve **obrigatoriamente** ser implementado utilizando a **API de sockets Unix**. O tipo de socket a ser utilizado deve **obrigatoriamente** ser **orientado a conexão (TCP)**, e o servidor deve **obrigatoriamente** ser um **servidor concorrente** (capaz de tratar simultaneamente requisições de vários clientes). O servidor deverá implementar o comportamento típico de um servidor de chat, ou seja, quando um cliente enviar uma mensagem de chat para o servidor, o servidor deverá reenviar essa mensagem para todos os outros clientes conectados ao chat naquele momento.

Para implementar o servidor concorrente você deverá criar um novo processo ou thread para tratar cada nova conexão. Sugere-se no entanto a **implementação com threads** para facilitar o compartilhamento de memória entre as diferentes unidades de execução (lembre-se que threads compartilham o espaço de endereçamento do processo que as criou). Os detalhes de como o servidor será implementado ficam a critério do aluno, inclusive a **especificação de qualquer estrutura de dados** para manter os dados dos clientes conectados ao chat. Lembre-se do que estudamos desde o início do semestre a respeito de sincronização: o acesso concorrente a dados compartilhados pode causar inconsistências, e você deverá **garantir sincronização entre as unidades de execução** através de primitivas como **mutex, semáforos, e variáveis de condição**, quando for necessário.

Além da funcionalidade básica descrita no primeiro parágrafo dessa seção, a avaliação do seu trabalho também irá incluir as suas decisões e justificativas a respeito de como implementar concorrência, de como especificar dados, e de como garantir sincronização no acesso a dados. Além disso, você é encorajado a incluir funcionalidades extras no servidor de chat. Alguns exemplos de funcionalidades adicionais incluem:

- Suporte a mais de uma sala de chat, especificada pelo cliente.
- Permitir que o cliente envie comandos ao servidor, por exemplo, “JOIN chat” e “LEAVE chat”

Note que você deverá definir alguma forma de delimitar esses comandos. Além disso, note que mensagens de chat são compostas de duas informações: o que foi dito e quem disse. Dessa forma, você deverá definir um protocolo simples de mensagens de tamanho fixo, ou alguma convenção através do uso de caracteres delimitadores especiais de modo que o servidor saiba quando mensagens enviadas por um cliente devem ser interpretadas como comandos ou chat, e também para que o servidor/cliente saiba quando as mensagens enviadas por cada cliente foram completamente transmitidas. DICA 1: ler discussão na seção 7.5 em Beej's Guide to Network Programming, Using Internet Sockets (leitura adicional passada nas notas de aula). DICA 2: lembre-se que em um socket orientado à conexão os limites das mensagens podem não ser preservados. Portanto, se um cliente escrever 20 bytes em um socket, não é garantido que o servidor irá ler 20 bytes em uma operação de leitura. O servidor poderá ler 20 bytes; mas o servidor também poderá ler menos, se houver alguma fragmentação do pacote. Nesse caso, você precisará fazer um laço e continuar lendo. Da mesma forma, se um cliente escrever 20 bytes em um socket, e em seguida escrever mais 20 bytes, o servidor poderá ler todos os 40 bytes de uma vez ... ou não. Portanto, é sugerido que você utilize alguma convenção (troca de mensagens de tamanho fixo, utilização de algum caracter especial) para identificar os limites de cada mensagem trocadas entre clientes e servidor.

Funcionalidades adicionais irão fazer parte da nota final, e serão contabilizadas no quesito “Interface e Usabilidade”.

3. DESCRIÇÃO DO RELATÓRIO A SER ENTREGUE

Deverá ser produzido um relatório fornecendo os seguintes dados:

- Descrição do ambiente de teste: versão do sistema operacional e distribuição, configuração da máquina (processador(es) e memória) e compiladores utilizados (versões).
- Entregar dois programas C diferentes (cliente e servidor).
- Um texto explicando suas decisões e justificativas a respeito de como implementar concorrência, de como especificar os dados mantidos pelo servidor a respeito dos clientes, de como garantir sincronização no acesso a dados, além das funcionalidades adicionais que você implementou.
- Também inclua no relatório problemas que você encontrou durante a implementação e como estes foram resolvidos (ou não).

4. DATAS E MÉTODO DE AVALIAÇÃO

O trabalho pode ser feito em grupos de no MÁXIMO 3 INTEGRANTES.

A implementação do trabalho deverá ser demonstrada em aula e o trabalho (incluindo o relatório) deverá ser submetido pelo moodle até às 23:55 horas do dia da apresentação. As apresentações serão nos dias 10/06/2013 (Turma B) e 11/06/2013 (Turma A).

Faz parte do "pacote" de entrega os fontes e o relatório em um arquivo ZIP. Não esquecer de identificar claramente os componentes do grupo no relatório.

Após a data de entrega será descontado 02 (dois) pontos por semana de atraso. O atraso máximo permitido é de duas semanas, isto é, nenhum trabalho será aceito após 24/06/2013 (Turma B), e 25/06/2013 (Turma A).