TP3: sistema de mensagens

Prática de programação usando select e timers

Trabalho individual ou em dupla
Data de entrega: 03/07/2012

O objetivo

Do ponto de vista funcional, o objetivo do trabalho é implementar um sistema de envio de mensagens curtas (um mini-twitter) que funcionará em um modelo multi-servidor. Do ponto de vista didático, o objetivo é o domínio da programação orientada a eventos utilizando-se a primitiva `select` e a temporização por sinais.

O problema

Você desenvolverá um servidor para um sistema simples de troca de mensagens em C ou C++, sem bibliotecas especiais, utilizando comunicação via protocolo TCP. O programa servidor deve enviar periodicamente uma mensagem para todos os clientes com informações gerais sobre o funcionamento do sistema.

Três programas devem ser desenvolvidos: um servidor, que será responsável pelo controle da troca de mensagens, e dois programas clientes, um para exibição das mensagens recebidas e um programa para envio de mensagens para o servidor. Cada programa cliente se identifica com um valor inteiro único no sistema, que ele deve receber como parâmetro de entrada (não há, na verdade, nenhuma verificação interna ao sistema para garantir a unicidade dos identificadores). Cada progama cliente estabelece uma conexão TCP para o servidor, que deverá então gerenciar as diversas conexões paralelamente. Para fins de alocação de recursos, pode-se limitar o no. de clientes conectados a 10 de cada tipo (exibição e envio).

Apesar da referência ao twitter, não há a noção de seguidores ou retwits, para quem já conhece. Cada mensagem de texto pode ser enviada para todo mundo ou para um único usuário, não há outras opções.

O protocolo

O protocolo de aplicação deverá funcionar sobre TCP, portanto todas as mensagens serão entregues como sequências de bytes, sem divisões fixas pré-estabelecidas, com garantia.

As mensagens possuem um formato bem definido, com quatro campos binários (inteiros de dois bytes): o tipo de mensagem, um campo de identificação de usuário (origem e destino) e um campo de tamanho do texto, se houver. No caso da mensagem conter um texto, esse texto segue após o cabeçalho e pode ter no máximo 140 caracteres (como o twitter), que devem ser sempre terminadas com o caractere de fim de string de C, um caractere com valor zero (que pode ser o caractere 141). Os campos inteiros devem ser enviados na ordem de bytes da rede (*network byte order*).

Para facilitar, o seguinte `struct` em C define o formato das mensagens:

```
typedef struct {
    unsigned short int type;
    unsigned short int orig_uid;
    unsigned short int dest_uid;
    unsigned short int text_len;;
    unsigned char      text[141];
} msg_t;
/* Pode-se agora definir variáveis to tipo msg_t */
```

As seguintes mensagens são definidas (identificadas pelo valor inteiro associado):

1. **OI (0):** Primeira mensagem de um programa cliente (tanto de teclado quanto de exibição) para se identificar para o servidor. Caso a conexão seja aceita, o servidor deve enviar de volta uma mensagem idêntica; caso contrário, basta que ele feche a conexão.

2. **TCHAU (1):** Última mensagem de um cliente de envio de mensagens para registrar a sua saída. A partir dessa mensagem o servidor fecha a conexão para aquele cliente e qualquer cliente de exibição que esteja associado a ele.

3. **MSG (2):** Mensagem enviada pelo usuário para o servidor e do servidor para o exibidor. O campo de identificador da origem indica o cliente que enviou a mensagem e o campo de destino deve conter o número do programa de exibição alvo ou zero para indicar que a mensagem deve ser enviada a todos os clientes. O servidor, ao receber tal mensagem, deve confirmar que o identificador do cliente de origem corresponde ao que foi usado na mensagem de OI daquele cliente.

Detalhes de implementação

Pequenos detalhes devem ser observados no desenvolvimento de cada programa que fará parte do sistema. É importante observar que o protocolo é simples e único, de forma que programas de todos os alunos deverão ser inter-operáveis, funcionando uns com os outros.

Servidor

Um sistema simples de mensagens de texto apresenta apenas um processo servidor ou repetidor e a sua função é distribuir as mensagens de texto dos transmissores para os receptores a ele conectados. O servidor deve aceitar pedidos de conexão dos diversos clientes (com a função `accept`), controlar os programas que se identificam por mensagens OI, fechar as conexões dos que enviem mensagens TCHAU. Ao receber uma mensagem MSG, deve primeiramente confirmar que a mensagem contém o identificador de origem correto (e descartá-la, caso contrário). Em seguida, ele deve observar o identificador de destino contido na mensagem. Se o identificador indicado na MSG for zero, a mensagem deve ser enviada a todos os exibidores cadastrados. Caso contrário (identificador diferente de zero), deve procurar pelo registro de um exibidor com o valor indicado e enviar apenas para ele. O programa exibidor, ao receber uma mensagem, deve exibir a identificação da origem (algo como "Mensagem de 12345: blá blá blá."). Caso uma conexão seja interrompida inesperadamente, o servidor deve tratar o evento como se uma mensagem TCHAU daquele cliente tivesse sido recebida.

Por poderem existir diversos clientes ativos simultaneamente e outros chegando e saindo a qualquer instante, o servidor não pode adotar um comportamento sequencial simples. Cada conexão será associada a um socket independente e ele não pode optar por fazer um `recv` em um deles à espera da próxima mensagem, já que não sabe de onde virá tal mensagem. Ele deve ser capaz de receber mensagens de qualquer conexão, assim como aceitar novos pedidos de conexão. Para isso, ele deve utilizar a primitiva `select` da biblioteca padrão.

Há diversos exemplos de código na rede sobre servidores que usam `select`. Obviamente, usar esses códigos não é proibido. Entretanto, é responsabilidade de cada aluno entender o princípio de funcionamento e a estrutura do código gerado.

A cada minuto, o servidor deve enviar uma mensagem de texto a todos os clientes de exibição conectados informando sua "identidade" (um string que o desenvolvedor escolher) o no. de clientes de exibição conectados e o tempo desde que o servidor foi iniciado. O formato para essa informação é de escolha do programador, mas pelas restrições do protocolo deve caber em uma mensagem de 140 caracteres. O identificador do servidor é zero, para indicar a origem da mensagem.

Programas de envio e recepção de mensagens

Não há exigências específicas sobre a interface desses programas com o usuário além do que já foi mencionado. O importante é que eles permitam ao usuário enviar mensagens para um destinatário

específico ou para todos e exibir as mensagens recebidas com a informação da origem.

Os programas de envio de envio e recepção de mensagens devem receber como argumentos o valor do seu identificador e o endereço (e porto) do servidor a ser contactado. Em seguida, devem abrir uma conexão para o servidor, enviar uma mensagem OI e esperar pelo OI de volta. Depois disso, programas de envio de mensagens devem entrar em um *loop* lendo mensagens do teclado (com a identificação do destino desejado), montando e enviando a mensagem para o servidor. Já o programa de exibição deve executar um *loop* esperando por mensagens do servidor e exibindo-as na saída padrão (a tela). A forma de exibição não é fixa, mas deve incluir a identificação do remetente e uma observação se a mensagem foi privada (enviada apenas para aquele cliente).

Identificadores

Clientes de exibição serão identificados por inteiros positivos (entre 1 e 999, para simplificar). Esses identificadores devem ser controlados pelos servidores: um servidor não deve aceitar uma conexão de um cliente se já houver outro cliente conectado localmente do mesmo tipo, com o mesmo identificador (uma mensagem de ERRO deve ser enviada de volta nesse caso). Não há, entretanto, nenhuma autenticação: desde que um identificador ainda não esteja sendo usado, ele pode ser usado por qualquer usuário. Clientes de envio de mensagens também devem ser identificados, só que com valores entre 1001 e 1999.

Quando um programa de envio de mensagem se registrar (enviar uma mensagem OI com um certo identificador maior que 1000), o servidor deve apenas se certificar de que não há um outro programa registrado com o mesmo identificador. Quando um programa exibidor se registrar o servidor deve se certificar que nem o número de identificação usado por ele e nem aquele número acrescido de 1000 estão sendo utilizados. Isso implica que, ao se registrar um par exibidor/enviador, o programa exibidor deve ser sempre disparado primeiro.

Controle de temporização no servidor

Para que o programa servidor envie sua mensagem periódica, ele deve controlar o tempo de forma independente dos demais eventos internos. Como há diversas conexões, a temporização embutida no `recv` não pode ser usada. Ao invés disso, uma temporização com sinais deve ser adotada. Um exemplo de como isso pode ser feito já foi apresentado anteriormente no programa impaciente.c.

Qualquer chamada do sistema operacional que possa ser interrompida por um sinal retorna com um valor de erro e um valor `EAGAIN` na variável `errno`. Caso isso ocorra em uma chamada que não seja a função `select` (pouco provável, mas não impossível) basta que a chamada seja reiniciada. Se o select for interrompido por um sinal, seria o momento correto para testar se o timer expirou (uma variável de *flag*) e enviar a mensagem determinada. Em seguida, o *flag*/timer deve se reiniciado e o processo reinicia.

Esta funcionalidade, exceto pelo tratamento da interrupção de outras funções da biblioteca, como `send` e `recv`, é praticamente independente do restante do trabalho. Uma forma razoável de organizar o trabalho seria colocar em funcionamento o sistema sem a mensagem periódica primeiro e fazer essa parte por último.

O que deve ser apresentado

Cada aluno/dupla deve entregar junto com o código um relatório curto que deve conter uma descrição da arquitetura adotada para o servidor, os refinamentos das ações identificadas no mesmo, as estruturas de dados utilizadas e decisões de implementação não documentadas nesta especificação.

Submissão eletrônica

Os trabalhos deverão ser entregues juntamente com o relatório gerado em um arquivo do tipo zip ou tar.gz através do Minha.ufmg. O código fonte do servidor e dos clientes devem ser submetidos com comentários que identifiquem no código as partes do algoritmo de alto nível e expliquem os

detalhes das partes principais do programa. Junto do código fonte, um arquivo `Makefile` deve ser incluído, de forma a facilitar a compilação do programa. **Não inclua arquivos objeto (.o) nem executáveis no seu arquivo de entrega.**

Critérios de avaliação

Serão atribuídos pontos para a execução correta do programa, para a organização e clareza do relatório e para a organização do código.

Observações Gerais

1. **Dúvidas:** esclarecimentos podem ser solicitados através do minha.ufmg sempre que possível.
2. Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
3. **Planeje para que seu código seja robusto. Tente prever falhas no processo de comunicação entre os componentes.**
4. **Vão valer pontos clareza, indentação e comentários no programa, bem como a qualidade da documentação que o acompanhe.**
5. É terminantemente proibido compartilhar programas ou trechos de programas. Tal comportamento poderá ser punido severamente.
6. É de inteira responsabilidade do aluno verificar a correta submissão dos arquivos dentro do prazo.

Última alteração: 03/06/2012