Redes TP1 - Publish/Subscribe

Nome: Bruno Vieira
Matrícula: 2016058018

1. Introdução

Nesse trabalho prático, o objetivo é desenvolver uma interface de comunicação entre servidor e cliente conhecida como publish/subscribe, na qual clientes demonstram interesse/desinteresse em um determinado assunto utilizando tags precedidas dos sinais “+” e “-”, respectivamente. Clientes inscritos em uma tag específica recebem mensagens, sob demanda, postadas por outros clientes que possuam a mesma tag precedida do caractere “#”. Para a elaboração do trabalho foi utilizado a linguagem C++ e o código-fonte base, fornecido pelo professor, por meio do Moodle.

2. Resolução do Problema

2.1. Cliente

No código do cliente, a conexão em rede é feita pela utilização de um socket da biblioteca POSIX, para comunicação com o servidor. Para garantir que o terminal do cliente pudesse receber mensagens do teclado, simultaneamente com respostas do servidor, foram criadas duas threads, com o auxílio da biblioteca pthreads do C. A primeira thread, criada com o método send_thread() é responsável pelo envio de pacotes na rede utilizando a função send(), assim como a segunda thread, criada com o método recv_thread(), é responsável pelo recebimento dos pacotes vindos do servidor, por meio da função recv(). Ambas as threads são coordenadas por uma variável booleana, chamada aberto, que define a abertura do código da função send() após o recebimento ou não de dados, por meio da função recv(), resolvendo conflitos de impressão no terminal.

```
if (aberto) {
  printf("\b\b> ");
  fgets(buf, BUFSZ-1, stdin);
  count = send(cdata->csock, buf, strlen(buf)+1, 0);
  if (count != strlen(buf)+1) {
    logexit("send");
  }
  aberto = false;
}
```

```
count = recv(cdata->csock, buf + total, BUFSZ - total, 0);
if (count > 1) {
  printf("\b\b");
  printf("< %s", buf+total);
  printf("> ");
  fflush(stdout);
}
total += count;
aberto = true;
```

2.2. Servidor

No código do servidor, a conexão em rede é feita, também, utilizando sockets e a função bind() da biblioteca POSIX. O servidor aguarda a conexão de um cliente, por meio da função listen(), e, após essa conexão, uma nova thread é criada no programa, para que vários clientes possam ser gerenciados simultaneamente.

Quando um cliente envia uma mensagem para o servidor, esta mensagem é verificada, para garantir que não possua caracteres inválidos (ocasionando a desconexão do cliente) e, caso não haja problemas, ela é encaminhada para o método tratar_mensagem(), que faz uma iteração sobre a string da mensagem, buscando tags de inscrição, desinscrição ou hashtags. Para cada um desses três casos, é utilizado um método respectivo, que define o fluxo do código e facilita o entendimento.

Para a gerência das tags de interesse de cada cliente, foi utilizada a estrutura de dados Map, do C++, definindo como **chave** o csock do cliente (que possui a informação necessária para envio de mensagens) e como **valor** uma estrutura de dados Set, que armazena a lista de todas as tags daquele usuário em forma de string. Conforme necessário, são feitas inserções, exclusões e buscas dentro dessas estruturas.

3. Métodos e Estruturas de Dados

`map<int, set<string> > tags_clientes`

Estrutura de dados utilizada para armazenar as tags de interesse de cada cliente. O valor csock único de cada cliente é armazenado como chave do Map, sendo vinculado a um Set com todas as suas tags cadastradas, semelhante ao esquema de exemplo abaixo:

map_de_clientes {
  csock_cliente1: { ‘tag1’, ‘tag2’, ‘tag3’ }
  csock_cliente2: { ‘tag2’ }
  csock_cliente3: { }
}

`bool checar_map_tags(int csock, string tag, int acao)`

Função responsável pela busca, inserção e exclusão de novas tags de interesse do cliente, que retorna um booleano. É passado como parâmetro o csock do cliente como int, a tag a ser buscada/inserida/excluída como string e a ação a ser executada como int.

Caso seja passado o valor **0** para o parâmetro “acao”, o algoritmo busca no Map de clientes (`map<int, set<string> > tags_clientes`) alguma tag, no Set que possui como chave o csock passado como parâmetro da função, que seja igual àquela buscada. Caso ele encontre, o algoritmo retorna **false**, caso contrário, a nova tag é inserida na estrutura de dados e o algoritmo retorna **true**.

Caso seja passado qualquer valor diferente de **0** para o parâmetro “acao”, o algoritmo também busca a tag na estrutura de Map. Caso ele encontre, a tag passada é removida da estrutura e o algoritmo retorna **true**, caso contrário, é retornado **false**.

`bool checar_ascii(char c)`

Função que verifica se um caractere é uma letra válida para o sistema (tanto maiúscula, como minúscula), seguindo a tabela ASCII. Caso o caractere passado como parâmetro para a função esteja dentro do intervalo de caracteres ASCII permitidos, o programa retorna **true**, caso contrário, o algoritmo retorna **false**.

`void enviar_mensagens(set<string> hashtags, string mensagem, int csock_autor)`

Método responsável pelo envio de mensagens, que contém hashtags, aos clientes interessados nas tags. Recebe como parâmetro um Set das tags de uma mensagem, a própria mensagem em forma de string e o csock do autor da mensagem, como int. O algoritmo busca em todo o Map de tags de clientes (`map<int, set<string> > tags_clientes`) se existe algum cliente inscrito nas tags passadas como parâmetro. Caso exista algum cliente interessado em alguma das tags, a função de envio de pacotes send() é chamada, enviando a mensagem para esse cliente (desde que não seja o mesmo cliente autor da mensagem). Em seguida, é executado um comando break, para que o mesmo cliente não receba a mesma mensagem mais de uma vez, caso esteja inscrito em mais de uma tag existente na mensagem.

`void buscar_hashtags(char *buf, int csock)`

Método para buscar mensagens com hashtag, como no exemplo: “Eu amo #dota”, com o auxílio da função `bool checar_ascii(char c)`. Recebe a mensagem a ser varrida e o csock do cliente autor. Caso sejam encontradas tags válidas, é chamado o método `void enviar_mensagens(set<string> hashtags, string mensagem, int csock_autor)` para lidar com os envios necessários.

`string tratar_tag(char *buf, char sinal)`

Função responsável por varrer uma mensagem enviada pelo usuário, buscando por inscrição “+” ou desinscrição “-” em uma tag. Recebe como parâmetro uma mensagem e o tipo de sinal (+ ou -) a ser buscado. Para cada caractere da mensagem a ser varrida é feita uma chamada da função `bool checar_ascii(char c)`, para garantir a validade dos dados. Caso seja encontrada alguma tag, o algoritmo retorna uma string contendo essa tag, caso contrário, o algoritmo retorna uma string vazia.

`string tratar_mensagem(char *buf, int csock)`

Função responsável pelo tratamento geral das mensagens recebidas por um cliente. Recebe como parâmetro uma mensagem e o csock de um usuário. Caso a mensagem recebida seja “##kill\n”, o servidor é encerrado. Caso a execução continue, são feitas duas chamadas da função `string tratar_tag(char *buf, char sinal)` para verificar se a mensagem recebida é de inscrição “+” ou desinscrição “-”. Também é feita uma chamada da função `void buscar_hashtags(char *buf, int csock)` para verificar se a mensagem recebida possui hashtags que possam ser de interesse para outros usuários.

Caso a mensagem recebida seja uma inscrição de tag, é feita uma chamada da função `bool checar_map_tags(int csock, string tag, int acao)`. Se o retorno de `checar_map_tags` for **true**, é retornada a string: **subscribed +tag\n**. Caso contrário, é retornada a string **already subscribed +tag\n**.

Caso a mensagem recebida seja uma desinscrição de tag, é, também, feita uma chamada da função `bool checar_map_tags(int csock, string tag, int acao)`. Se o retorno de `checar_map_tags` for **true**, é retornada a string: **subscribed -tag\n**. Caso contrário, é retornada a string **not subscribed -tag\n**.

Em qualquer outro caso, é retornada uma string vazia.