Descrição do programa

O programa realiza a trocar de mensagem entre nós da rede. Utiliza o protocolo UDP e realiza a entrega confiável de mensagens com base no reenvio de mensagens não sinalizadas como recebidas. Simula também a perda de pacotes com base em um parâmetro passado na linha de comando que indica a probabilidade (em percentual) de o pacote ser perdido pelo roteador.

static void dijkstra(int *m, int a, int dim)

A função **dijkstra** é a responsável por encontrar o menor caminho no grafo, ou seja, ela é quem procura os nós mais próximos na rede, para que a informação se propague de maneira mais rápida e eficiente. Ela recebe 3 parâmetros: a matriz de adjacências “m” do grafo, um vértice inicial para começar a busca “a” e o tamanho da matriz “dim”, esta função percorre toda a matriz de adjacências partindo pelo vértice inicial procurando sempre o menor caminho entre os seus vértices vizinhos.

Pode-se notar que “dim” é maior do que o tamanho real da matriz, para que o vértices comecem em 1, ao invés de zero, coincidindo assim, com o número dos nós. O índice zero não é usado nesta abordagem.

static void loadgraph(void)

Monta o grafo a partir do arquivo "enlaces.config".

static int path_next(int n)

Acessa os valores calculados por “dijkstra()”. O atributo “n” é o vértice final do caminho a ser calculado, e é retornado o próximo nó do caminho.

static int luck(int per)

Realiza um sorteio com chance percentual "per" de retornar verdadeiro. Utilizada para simular a perda de pacotes durante a transmissão de dados.

static void sys_exit(int n)

Ao se optar para sair do programa, digitando qualquer string no terminal começando com a letra “q”, esta função libera os ponteiros do programa e o mutex. O parâmetro “n” serve para indicar qual o tipo de saída (1 – para algum tipo de erro ou 0 – para saída normal).

static void sys_sleep(int milli)

É usada para fazer o programa dormir por alguns milissegundos, a variável "milli" armazena o valor do tempo que o programa deve ficar inativo em milissegundos, poupando assim, processamento do programa.

static int sys_getmilli(void)

Captura o tempo em milissegundos para ser usado com a variável global “TIMEOUT” e saber se já está na hora do reenvio ocorrer ou não.

static void loadconfig(void)

Abre o arquivo "roteador.config" onde contém as informações sobre as configurações dos nós: número do nó, porta e ip. Note que o número dos nós começa em 1.

static int msg_getdst(const char *msg)

Obtém o nó de destino gravado no cabeçalho do pacote "msg".

static void msg_setdst(char *msg, int n)

Coloca na string “msg” (o pacote) quem é o nó de destino, alterando seu cabeçalho.

static int msg_getsrc(const char *msg)

Idem a “msg_getdst(const char *msg)”, porém, para obter a origem.

static void msg_setsrc(char *msg, int n)

Idem a “msg_setdst(char *msg, int n)”, porem, para gravar a origem.

static int msg_getcmd(const char *msg)

Verifica qual comando do terminal foi solicitado: “M” para uma mensagem, ou “C” para um confirmação de uma mensagem.

static void msg_setcmd(char *msg, int cmd)

Grava qual o tipo de mensagem no pacote.

static int msg_getseq(const char *msg)

Extrai de “msg” qual a sequência de pacotes esperados (se 0 ou 1).

static void msg_setseq(char *msg, int n)

Grava em “msg” qual a sequência do pacote esperado.

static const char *msg_gettext(const char *msg)

Extrai de “msg” qual a mensagem de texto que deve ser trocada.

static void msg_settext(char *msg, const char *text)

Grava em “msg” qual a mensagem de texto que deve ser trocada.

static void sendmessage(const char *msg)

Esta função é responsável por encaminhar a mensagem a ser trocada entre os nós. Primeiro, ela verifica se a mensagem é válida, após isso, o mutex é setado para 1, para que assim se possa tomar os procedimentos pertinentes.

Todos os atributos relacionados acima são ajustados: origem, destino, tipo (“cmd”) é setado para “M”, e a sequência dos pacotes. A variável “msg_pending” é setada em 1 até que ela seja entregue ao destinatário; esta flag serve para saber se a mensagem foi ou não entregue. Caso a mensagem for entregue, a variável mutex é liberada.

A mensagem tem tamanho de 128 bytes, onde os 6 primeiros bytes são para as informações sobre destinatário, origem, tipo e sequencia, os próximos 100 bytes são usados para a mensagem em si e os demais estão sobressalentes, reservados para uso futuro.

static void kbd_main(void)

Esta função é responsável por montar o cabeçalho da "msg" (pacote) e ver se ela é ou não válida. Se o comando escrito pelo usuário começar com o caractere "q", significa que o usuário deseja sair do programa. A mensagem então é montada através da função" sendmessage(msg)". Retira a mensagem da entrada, coloca "\0" (final de string) no final da mensagem no lugar de "\n" (que brade linha).

static void *send_main(void *arg)

Esta função roda em uma thread separada. Recebe como parâmetro o que foi passado pelo usuário, se "msg_pending==1", então mutex é setado para 1, o destino é calculado e a mensagem é retransmitida, enquanto "msg_pending" estiver setado em 1, "msg_timeout" é disparado para que se torne possível saber se a mensagem foi ou não entregue, isso quer dizer que enquanto a mensagem não for entregue, dentro de um certo limite (neste caso 500 milissegundos), o programa fica dormindo, esperando a mensagem de confirmação, o programa acorda quando a mensagem chega, ou retransmite novamente a mesma.

static void net_send(int dst, const char *data)

Encarregado de enviar as mensagens para o destinatário.

static void makesocket(void)

Cria e configura um socket UDP para a troca das mensagens entre os vértices do grafo da rede. Os parâmetros são: Domínio ("AF_INET" - é usada para protocolos de domínio da Internet IPV4"), Protocolo ("SOCK_DGRAM – protocolo UDP") e a porta a ser usada.

A função "bind(my_socket, (struct sockaddr *)&si_me, sizeof(si_me))" atribui um IP aleatório - criado em "htonl(INADDR_ANY)" - ao socket, ela retorna 0 se a atribuição for bem sucedida ou -1 caso contrário. Em caso de erro, o programa é fechado.

int main(int argc, char **argv)

Primeiramente são passados para a main os parâmetros argc, que contém a contagem do número de argumentos da linha de comando, e argv, que é um array com os próprios argumentos.

A primeira tarefa a ser executada é gerar uma semente para números aleatórios com a função srand(), passando o tempo como parâmetro. Após isso, o mutex é criado.

Logo após, as informações do arquivo "roteador.config" são lidas através da função "loadconfig()". A função "loadgraph()" monta o grafo e a função "makesocket()" cria o socket. O programa cria duas threads: uma para receber as mensagens e outra para envia-las, como cada thread é individual, o programa consegue tanto enviar, quanto receber mensagens de maneira independente. É mostrado na tela o número do

nó. Em seguida, a função “kbd_main()” é chamada para montar o cabeçalho, e por fim, o programa chama a função “sys_exit()”.

static void cmdline(int argc, char **argv)

Usada para pegar o que foi escrito na linha de comando e colocar nas variáveis globais.

static void net_receive(void)

Esta função é recebe um pacote destinado ao roteador (nó) do grafo, não importando qual sua posição dentro do grafo da rede.

A linha “socklen_t slen = sizeof(si_other)” adiciona em “slen” o tamanho da estrutura “si_other”. Já na linha “reclen = recvfrom(my_socket, indata, DATAGRAM_LEN, MSG_WAITALL, (struct sockaddr *)&si_other, &slen)”, “reclen” recebe um pacote de qualquer lugar (nó) direcionado para o socket e preenche “si_other” com o endereço do nó que está mandando a mensagem.

static void *rcv_main(void *arg)

Com a função “net_receive()”, consegue-se capturar o pacote de dados que contém tanto as informações de cabeçalho, quanto a mensagem em si, com isso, define-se a origem e o destinatário da mensagem, e os demais atributos.

Para confirmar que a mensagem foi recebida com sucesso, a linha “msg_setcmd(indata, 'C')” seta a flag tipo (“cmd”) com o valor “C”, assim ela pode ser enviada de volta a quem mandou confirmando sua entrega. A variável “msg_pending” é setada para 0, significando o recebimento da mensagem (lembre-se que aqui estamos falando da mensagem em si e não da confirmação da entrega).

Os atributos origem, destinatário e tipo são ajustados e então a mensagem de confirmação é encaminhada até o nó de destino.