Protocolo de Ligação de Dados

Relatório

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Jorge Filipe Monteiro Lima - 201000649
Nuno Filipe Dinis Cruz - 201004232
Vasco Fernandes Gonçalves- 201006652

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

11 de Novembro de 2013

1 Protocolo de ligação lógica

O protocolo de ligação lógica tem os seguintes aspectos funcionais:

- Envio de tramas de supervisão
- Envio de tramas com *timeout*
- Estabelecimento de ligação
- Construção e envio de tramas de informação
- *Byte stuffing*
- Leitura de tramas
- Extração de pacotes de dados de tramas de informação
- Término de ligação

1.1 Envio de tramas de supervisão

As tramas de supervisão são enviadas com a seguinte função:

```
void sendControlFrame(int fd, int c)
{
    writeBuf[0]=FLAG;
    writeBuf[1]=A;
    writeBuf[2]=c;
    writeBuf[3]=writeBuf[1]^writeBuf[2];
    writeBuf[4]=FLAG;
    writeBufLen=5;
    printf("SENDING COMMAND 0x%x\n", c);
    write(fd, writeBuf, writeBufLen);
}
```

Esta função constrói uma trama de supervisão, sendo que o byte C é passado como parâmetro.

1.2 Envio de tramas com *timeout*

Para enviar tramas com protecção de *timeout* criamos a seguinte função:

```
void sendFrame()
{
    if(conta>0) //incrementa numero de timeouts
        numTimeouts++;
        if(conta++ <= MAX_RETRIES)
        {
            write(globalFD, writeBuf, writeBufLen);
```

```
            printf("\tSENDING PACKET\n");
            alarm(3);
        }
        else
        {
            printf("CONNECTION LOST... GIVING UP\n");
            exit(1);
        }
}
```

Esta função é registada no programa como *handler* do *SIGALRM* e de cada vez que é chamada, no máximo o número de vezes definido em *MAX_RETRIES*, é chamada novamente. Quando houver a leitura de uma trama válida, os alarmes pendentes são cancelados.

1.3 Estabelecimento de ligação

Para estabelecer a ligação, a seguinte função é chamada:

```
int setConnection(char * porta, int user)
```

O parâmetro user toma os valores de 0 para o emissor e 1 para o receptor. Esta função configura a porta série e, do lado do emissor, envia a trama de supervisão **SET** e aguarda pela recepção da trama **UA**. Do lado do receptor aguarda a recepção da trama **SET**, envia a trama **UA** e de seguida os dois computadores estão prontos para a transmissão de dados.

1.4 Construção e envio de tramas de informação

Criamos a seguinte função (chamada pelo emissor, na camada da aplicação) para a construção de uma trama de informação:

```
int sendData(int fd, char *packet, int packetChars)
```

O conteúdo do pacote e o seu tamanho são passados da aplicação pelos parâmetros **packet* e *packetChars*. De seguida, é construído o cabeçalho da trama e os dados são inseridos:

```
for (i = 4; i<packetChars + 4; ++i)
{
    writeBuf[i] = packet[i - 4];
    //printf("writeBuf[%i] -> %x\n", i, writeBuf[i]);
    bcc2 ^= writeBuf[i];
}
```

O bcc2 é assim calculado, e os bytes são *stuffed*:

```
int packetCharsAfterStuffing = stuffBytes(packetChars+1);
```

De seguida, a trama é enviada e aguarda-se a resposta (fazendo novas tentativas enquanto não houver uma resposta). Se a resposta for afirmativa (**RR**), a função alterna o C (entre

0 e 2) e retorna, em caso contrário (**RR**) a função volta a ser chamada para efectuar o reenvio.

1.5 *Byte stuffing*

O tratamento dos dados a enviar (*stuffing* é feito pela seguinte função:

```
int stuffBytes(unsigned int numChars)
```

Esta função percorre todos os caracteres dos dados a enviar e, caso encontre os *bytes* **0x7e** ou **0x7d**, passa todos os bytes seguintes uma posição para a direita e insere os caracteres de escape apropriados (**0x5e** ou **0x5d**, respectivamente):

```
memmove(writeBuf+i+1,writeBuf+i,numChars-(i-4));
writeBuf[i]=0x7d;
writeBuf[++i]=<0x5e ou 0x5d>;
```

1.6 Leitura de tramas

Utilizamos uma única função para ler tramas de informação e supervisão:

```
int readFrame(int fd, char * buffer, int *res)
```

Esta função, que representa uma máquina de estados, lê, caracter a caracter, uma trama da porta série e retorna o byte C lido. Sendo uma máquina de estados, esta função só retorna quando tiver sido lida uma trama bem formada, na sua totalidade.

Se se tratar de uma trama de informação, a função escreve os bytes da secção de dados da trama para o array de caracteres *buffer* (passado como parâmetro) e no final escreve também o número de caracteres lidos para o apontador de inteiro *res*. É também alternado o C, entre 0 e 2.

1.7 Extração de pacotes de dados de tramas de informação

De seguida, temos a função que é chamada pelo leitor, na camada de aplicação, para receber pacotes:

```
int readData(int fd, char * buffer)
```

Esta função chama a função anteriormente descrita, faz o tratamento correcto das repostas a dar (conforme a validação do BCC2) e escreve em **buffer* o pacote lido.

É ainda nesta função que recebemos a trama **DISC**, vinda do emissor, e passamos essa informação ao receptor.

1.8 Término de ligação

Por fim, temos a função que termina a ligação:

```
int disconnect(int fd, int user)
```

De acordo com o parâmetro user (0 para emissor, 1 para receptor), envia as tramas **DISC** e **UA** apropriadas e fecha a ligação à porta série.

É também nesta função que mostramos ao utilizador as estatísticas da transmissão.

2 Protocolo de aplicação

O protocolo de aplicação tem os seguintes aspectos funcionais:

- Abertura e fecho de ligação
- Emissor - processamento do ficheiro a enviar
- Emissor - envio de pacotes
- Receptor - recepção de pacotes

2.1 Abertura e fecho de ligação

A aplicação, quando é aberta, recebe por argumento a porta série a utilizar. Com esta informação, é chamada a função:

```
int llopen(char *porta)
```

que simplesmente chama a função **setConnection** do protocolo de ligação lógica. Por sua vez, a função **llclose** chama a função **disconnect**.

2.2 Emissor - processamento do ficheiro a enviar

O emissor recebe por argumento o nome do ficheiro a enviar. Com este nome, chama a seguinte função:

```
int openFile(char *fileName, char **fileContent)
```

que guarda em ***fileContent* o conteúdo do ficheiro e retorna o seu tamanho. De seguida, a função

```
void sendFile(int fd, unsigned int fileSize, char *fileName,
    char *fileContent)
```

divide o ficheiro em *chunks* e envia-os.

2.3 Emissor - envio de pacotes

A função que trata de construir e enviar os pacotes de controlo é a seguinte:

```
void sendControlPacket(int fd, int packetC, unsigned int
    fileSize, char *fileName)
```

Esta função recebe o tipo de pacote a enviar (*Start* ou *End*), constrói-o e envia-o. Quanto aos pacotes de dados, a função é esta:

```
void sendDataPacket(int fd, char *packet, int dataChars,
    unsigned char n)
```

Recebe o *chunk* a enviar no parâmetro **packet*, o seu tamanho e o N (número de referência do pacote), envia o pacote.

2.4 Receptor - recepção de pacotes

O receptor recebe pacotes até o protocolo lhe transmitir um pedido de *disconnect*. No início, cria um ficheiro temporário para onde vai guardando os *chunks* e no final renomeia esse ficheiro para o nome recebido nos pacotes de controlo.

A função que lida com os pacotes é esta:

```
int packetStateMachine(char *buffer)
```

De acordo com o tipo de pacote recebido, armazena as informações sobre o ficheiro em variáveis globais (pacotes de controlo) ou processa os dados recebidos (pacotes de dados).

Os pacotes de dados são verificados, através do **N** (número de referência) e, se não se tratar de um pacote duplicado, armazena o *chunk* para o ficheiro.

3 Validação

O protocolo faz verificações a dois níveis: a nível das tramas com o **BCC1**, que é um xor dos campos do cabeçalho e o **C**, que alterna entre 0 e 2. Estes valores são passados no cabeçalho das tramas.
Verifica ainda, a nível dos pacotes, o **N**, um número de referência que assume valores entre 0 e 255.

Todos os valores estão sincronizados entre emissor e receptor, de forma a poderem ser verificados.

4 Elementos de valorização

Os elementos de valorização implementados foram os seguintes:

Implementação de REJ

As tramas rejeitadas implicam são reenviadas, sem ser necessário esperar pelo próximo timeout:

Do lado do receptor:

```
else if(readC == -1) //invalid bcc2
{
    printf("\nINVALID DATA, SENDING REJECT\n");
    numRejects++;
    sendControlFrame(fd, C_REJ ^ (c<<4)); //passa os 4 bits
       do c para a esquerda
    return -1;
}
```

Do lado do emissor:

```
else //REJECT OR INVALID RESPONSE
{
    numRejects++;
    printf("\tREJECT! RESENDING!!!\n");
```

```
        sendData(fd, packet, packetChars); //reenvia
    }
```

Verificação da integridade dos dados pela Aplicação

Tamanho do ficheiro recebido

É validado no leitor através dos pacotes de controlo:

```
if(fileSize != ftell(pFile))
    printf("\nERROR TRANSFERING: WRONG FILE SIZE\n");
```

Pacotes de dados duplicados

Validação feita na leitura de pacotes, do lado do leitor, com o número de referência **N**:

```
if(buffer[1]==n) //se nao for um pacote duplicado, grava
    para ficheiro
```

Registo de ocorrências

Ocorrências são gravadas em variáveis globais e escritas ao sair do programa:

```
//IMPRIME ESTATISTICAS
printf("Numero de tramas I: %i\n", numTramas);
printf("Numero de timeouts: %i\n", numTimeouts);
printf("Numero de rejects I: %i\n", numRejects);
```

5 Conclusões

No final deste trabalho concluímos que a definição *à priori* de todas as especificações para o mesmo é extremamente importante e merecedora do tempo nela investido. Só assim podemos evitar constantes obstáculos inerentes da incerteza de como proceder em determinadas situações críticas.

Assim, ficou reforçada a ideia de que para ser engenheiro não basta saber programar, mas também saber planear com exactidão os passos a percorrer no desenvolvimento de qualquer sistema ou aplicação.

Quanto ao tópico do trabalho, pareceu-nos uma boa maneira de abordar, de forma introdutória, a temática das redes de computadores, os problemas mais comuns que podem surgir e como os solucionar.

ANEXO 1 - CÓDIGO FONTE