Faculdade de Engenharia da Universidade do Porto

Data Link Layer Protocol

Projeto 1 – Grupo F

Redes de computadores

Maria Pimentel Pestana Cardoso Ribeiro – up201703844

Miguel Filipe Santos Silva – up201904672

Índice

Sumário

No âmbito do primeiro trabalho laboratorial de Redes de Computadores foi realizada a transferência de dados de um computador para outro através de uma porta de série assíncrona.

Foi possível realizar uma transmissão de dados e a utilizar mecanismos de deteção e controlo de erros de modo a parar a transmissão se forem detetados erros ou realizar as ações necessárias.

Introdução

Neste trabalho foi pedido para implementar um protocolo de ligação de dados através da porta de série utilizando o protocolo Stop & Wait de modo a detetar erros e anomalias. Desenvolvemos diversas funções de deteções de erros para que seja possível evitar falhas de receção de tramas.

Arquitetura e Estrutura do código

No nosso código temos uma única API para criar uma interface entre a Application Layer e a Data Link layer, que está definida numa única biblioteca, linklayer.c. Nesta biblioteca temos implementadas 4 funções, llopen(), llwrite(), llread() e llclose(), que nos permite comunicar entre dois nós através da camada de aplicação.

A função de abertura e configuração da ligação é realizada na primeira função, llopen(), onde inicialmente alteramos as configurações da porta série pretendida. Esta função pode ser chamada de duas formas: “TRANSMITTER” e “RECEIVER”.

Implementamos também as funções llread() e llwrite() que realizam a leitura e a escrita da informação. No caso da primeira inicialmente descodifica a informação presente nas tramas e envia uma confirmação dessa receção ao contrário da llwrite() que codifica a informação e posteriormente escreve e envia a trama.

Todas as funções descritas anteriormente utilizam a Struct linkLayer, que nos permite receber valores cruciais definidos pelo main.c, tais como a porta de série, número de tentativas, timeout e a role do programa.

Como auxiliar às 4 funções principais, temos uma função wait_for_answer(), que é responsável por ler os dados iniciais da ligação (HEADER) e retornar o valor do byte de controlo lido.

Casos de uso principais

Os casos de uso principais são utilizados na inserção de dados pelo utilizador e transmissão de informação que diferem no caso do emissor e do recetor.

No caso do emissor:

- Inicia a ligação entre emissor e recetor através da função llopen(), que envia trama de início de ligação SET e espera receção da confirmação ao receber a trama UA.
- Se a ligação for iniciada corretamente, o emissor passa à função llwrite() para enviar as tramas de informação com a mensagem desejada.
- Quando terminar de enviar, fecha a ligação através do envio de uma trama DISC.

O emissor terá um timeOut, que ativa quando não recebe confirmação do recetor.

No caso do recetor:

- Inicia ligação através do llopen(), ficando à espera da receção da trama de início de ligação SET, e envia a trama UA, como confirmação de receção.
- Se a ligação for iniciada corretamente, passa à função llread(), que espera receção da informação e retorna à aplicação.
- Quando terminar de receber a informação, recebe uma trama DISC, envia outra trama DISC de volta e recebe UA como confirmação, para poder fechar a ligação.

Protocolo de ligação lógica

O nosso protocolo de ligação, tem várias funções:

- Iniciar uma ligação, com o uso de tramas de Supervision e Unnumbered através de tramas SET e UA;
- Enviar e receber informação através de tramas de Informação;
- Usar bits de paridade, BCC2, para verificação de erros na receção;
- Usar stuffing/destuffing, para sabermos quando recebemos o fim da mensagem;
- Reenvio de mensagem em caso de TimeOut ou erro.

Estando estes objetivos divididos pelas seguintes funções:

wait_for_answer():

Esta função é uma função genérica de receção do HEADER. Pode ser chamada de duas formas distintas. Se for como Transmitter, iniciamos o timer e depois passamos à receção de dados. É aqui que será dado o timeOut caso não receba dados de volta. Para verificar a receção dos dados, verificamos a primeira FLAG, verificamos o BCC1 e verificamos o Address.

No fim da função retornamos o valor do Control. Em caso de erro na verificação dos valores recebidos ou timeOut, iremos retornar valores de erro.

llopen():

A função é responsável por abrir a conexão através de parâmetros definidos na camada de aplicação, main.c. Guardamos estes parâmetros, e passamos à verificação da abertura da ligação.

É importante definir que tipo de uso será utilizado, "TRANSMITTER" ou "RECEIVER", pois operam de maneira diferente. No caso do emissor, enviamos uma trama SET a indicar que queremos iniciar a ligação, e esperamos a receção de uma trama UA, com o apoio da função wait_for_answer(). Caso haja erro, retransmitimos.

No outro caso, o recetor estará sempre à espera de uma trama. Ao receber a trama SET, sabemos que a ligação foi iniciada e enviamos a trama UA a confirmar que a receção foi bem recebida. Caso haja erros, ignoramos e voltamos a ler novamente.

Na eventualidade de existirem erros, estes são detetados na função wait_for_answer() ou na leitura última FLAG.

llread()

Utilizamos esta função para realizar a leitura das tramas.

A sua funcionalidade é ler a informação recebida, onde também realiza verificações de erros na Data utilizando o BCC2. Inicialmente esta função permanece num ciclo enquanto espera pela receção da trama. Ao receber a trama, lemos o Header e enquanto a receção da Data, descodificamos a informação e verificamos o bit de paridade.

Enviamos uma trama de controlo RR em caso de sucesso da transmissão, o que permite a saída do ciclo. No caso de existirem erros na Data é enviada a trama REJ para que a informação seja enviada de novo, e se os erros foram no Header, ignoramos a informação. O bit de controlo de erro enviado, será o oposto do recebido.

llwrite()

Esta função é utilizada para escrever informação (Data).

Aqui definimos o bit de paridade da informação a enviar, e fazemos byte stuffing para uma variável stuffed[]. Depois, escrevemos os dados e esperamos a confirmação.

Nesta trama é enviado um controlo 0 ou 1 e recebemos uma trama RR ou REJ, com controlo 1 ou 0, respetivamente.

Na receção, se for detetado algum erro iremos receber a trama REJ, que indica que temos de reenviar a trama. Caso recebamos o RR, o envio e dados foi feito com sucesso, e a função é concluída.

LIclose():

Quando terminarmos de enviar o ficheiro, é necessário encerrar a conexão. Para isto temos de verificar, se se trata do "TRANSMITTER" ou "RECEIVER". Para terminar a ligação, o emissor envia uma trama DISC e espera por uma trama DISC de volta. Para ambos poderem dar a ligação como concluída, o emissor envia a trama UA, de confirmação e o recetor espera a receção desta mesma trama.

Após o encerrar da ligação, imprimimos as estatísticas do programa onde indicamos as tramas de informação enviadas, número de timeOuts, número de erros e numero de tramas REJ enviadas/recebidas.

Validação

O nosso protocolo realiza com sucesso:

- Transmissão sem erros;
- Ligar e desligar a porta séria a meio da transmissão;
- Ligação com ruído

O ficheiro de teste foi "penguin.gif", com 10,7 KBytes. Este ficheiro foi recebido com sucesso tanto em ambiente virtual, como no laboratório em ambiente físico.

Ao enviar este ficheiro tivemos sempre uma ligação de sucesso, com diferentes BaudRats, diferentes tamanhos da informação enviada, diferentes néméros de retransmissões e diferentes intervalos de timeOut.

A realização da parte de ligar e desligar a porta série, foi nos possível realizar em ambos os ambientes, e validou a eficácia do timeOut e a capacidade de ignorar uma trama recebida ou não.

Quando introduzimos o ruído, que foi maioritariamente verificado no laboratório, notamos a capacidade do nosso protocolo de ignorar um erro no Header, e em caso de deteção de erro na Data, através do bit de paridade, é capaz de enviar a trama REJ e atuar em conforme.

Elementos de valorização

Como elementos de valorização, implementamos:

- Impressão das estatísticas de erro
- Erro aleatório

De forma a testar o nosso protocolo em ambiente virtual, foi criado uma biblioteca fake_error.h, que tem uma certa probabilidade de criar erro. Esta biblioteca foi implementada na leitura do Header e na leitura da Data. Assim conseguimos testar em ambiente virtual a nossa implementação. No entanto, embora a biblioteca funcione, não é usada no programa principal em laboratório, pois interferiria com a implementação.

Conclusões

O objetivo principal deste trabalho é que a implementação do protocolo de Ligação de dados fosse independente da camada de aplicação para que aumente a facilidade ao abrir uma ligação entre dois nós. Tal objetivo foi atingido com sucesso. Foi por isso mantida a independência de camadas entre a camada de aplicações e a camada de ligações sendo que a camada de aplicação é responsável pela criação, interpretação de dados e o cabeçalho do pacote de dados. Já a camada de ligação é apensas responsável pela coesão das tramas e o envio das mesmas.

Tendo isto em conta, a realização deste trabalho permitiu que aprofundássemos e puséssemos em prática os conhecimentos adquiridos na unidade curricular.

Anexos

De seguida encontram-se as funções que criamos de modo a facilitar a implementação do trabalho.

- Função que verifica os fake errors

```c
#include "fake_error.h"

// se for 1, é erro, if zero, no erro
int data_error(int probability)
{
    time_t t1;
    int aux;

    srand((unsigned)time(&t1));

    aux = (rand() % 100);
    printf("aux->%d\n", aux);

    if (aux < probability)
    {
        return 0;
    }

    return 1;
}
```

Fig. 1 – Função que verifica os "fake error"

- Definição Struct Layer

```c
struct linkLayer ll;
sprintf(ll.serialPort, "%s", argv[1]);
ll.role = 1;
ll.baudRate = 9600;
ll.numTries = 3;
ll.timeOut = 3;
```

Fig 2 – Struct layer

- Função wait_for_answer()

```
char wait_for_answer()
{

    char input[HEADER_SIZE + 1], aux;
    int res = 0;

    (void)signal(SIGALRM, escrita);

    if (!state)
    {
        state = 1;
        alarm(0);
        alarm(ll.timeOut);
    }

    while (state < 3)
    {
        switch (state)
        {
        case -1:
            return -2;

            // if error, but we only count the errors inside the function
        case 0:
            return -1;
```

(3)

```
case 1:
    while ((!read(fd, &input[0], 1)) && state)
    {
    }
    if (input[0] == FLAG)
    {
        state = 2;
        alarm(0); // paramos o timer, porque de facto temos uma receção de dados
    }
    else
    {
        printf("Primeira Flag mal recebida\n");
        nerrors++;
        state = -1;
        alarm(0);
    }

    break;

case 2:
    // flag += data_error(PROBABILITY); // remove when using actual code
    // flag++;
    // printf("flag->%d\n", flag); // remove aswell
    res = 0;
    while (!res && state)
    {
        sleep(0.00001);
        res = read(fd, &input[1], 3);
    }
```

(4)

```
        if ((input[3] != (input[1] ^ input[2])) || flag == 1)
        {
            printf("ERRO, BCC1 diferente\n");
            read(fd, &aux, 1); // le FLAG para dar clear no buffer
            nerrors++;
            state = -1;
            break;
        }
        if (input[1] != A_T)
        {
            printf("ERRO, Address isnt whta it should be");
            state = -1;
            nerrors++;
            break;
        }
        sleep(0.00001);
        state++;
        break;
    }
}
return input[2]; // enviamos o control para verificação
```

(5)

Fig. 3,4,5 – Codigo função wait_for_answer

- Função llopen()

```
int llopen(linkLayer connectionParameters)
{
    char aux;

    sprintf(ll.serialPort, "%s", connectionParameters.serialPort);
    ll.role = connectionParameters.role;
    ll.baudRate = connectionParameters.baudRate;
    ll.numTries = connectionParameters.numTries;
    ll.timeOut = connectionParameters.timeOut;

    /*
      Open serial port device for reading and writing and not as controlling tty
      because we don't want to get killed if linenoise sends CTRL-C.
    */

    if (!fd) // verifico se fd ja foi iniciado or not
    {
        fd = open(connectionParameters.serialPort, O_RDWR | O_NOCTTY);
        if (fd < 0)
        {
            perror(connectionParameters.serialPort);
            exit(-1);
        }

        if (tcgetattr(fd, &oldtio) == -1)
        {
            perror("tcgetattr");
            exit(-1);
        }
```

(6)

```
    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = get_baud(connectionParameters.baudRate) | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;
    newtio.c_lflag = 0;

    newtio.c_cc[VTIME] = 1;
    newtio.c_cc[VMIN] = 0;

    tcflush(fd, TCIOFLUSH);

    if (tcsetattr(fd, TCSANOW, &newtio) == -1)
    {
        perror("tcsetattr");
        exit(-1);
    }

    printf("\nLigação Iniciada\n");
    // Comunicaçao aberta
}

int k = 0, res = 0;
char input[5];
```

(7)

```
switch (connectionParameters.role)
{

case TRANSMITTER:

    input[0] = FLAG;
    input[1] = A_T;
    input[2] = SET;
    input[3] = BBC1_T;
    input[4] = FLAG;

    printf("Transmissor enviou info\n");
    state = 0;
    while (state != 2)
    {

        switch (state)
        {

        // escrever informaçao e verificar
        case 0:
            if (k > connectionParameters.numTries)
            {
                printf("Dados não recebidos de volta, problemas de envio\n");
                return -1;
            }
            if (k > 0)
            {
                printf("Retransmissão\n");
            }

            k++;
            res = write(fd, input, 5);
            aux = wait_for_answer();
            if (aux == UA)
```

(8)

```
            res = write(fd, input, 5);
            aux = wait_for_answer();
            if (aux == UA)
            {
                state = 1;
                break;
            }

            state = 0;
            break;

            // receber informaçao
        case 1:
            while (!read(fd, &aux, 1) && state)
            {
            }

            if (aux != FLAG)
            {
                printf("Nao recebemos last FLAG, trying again\n");
                nerrors++;
                state = 0;
                break;
            }

            state++;
            printf("llopen Transmitter done successfully\n");
            return 1;

            break;
        }
    }

return 1;
```

(9)

```
case RECEIVER:

    input[0] = FLAG;
    input[1] = A_T;
    input[2] = UA;
    input[3] = BCC1_R;
    input[4] = FLAG;

    printf("A começar RECEIVER\n");

    // control field definition
    while (state != 2)
    {
        // Receiver
        switch (state)
        {
        case 0:
            state = 1;
            aux = wait_for_answer();
            if (aux == SET)
            {
                state = 1;
                break;
            }

            state = 0;
            break;
```

(10)

```
        case 1:
            while (!read(fd, &aux, 1))
            {
            }

            if (aux != FLAG)
            {
                nerrors++;
                printf("Nao recebemos last FLAG\n");
                state = 0;
                break;
            }

            write(fd, input, SUP_SIZE);
            state++;
            printf("llopen RECEIVER done successfully\n");
            return 1;

            break;
        }
    }
    break;
}
return -1;
```

(11)

Fig. 6,7,8,9,10,11 – Codigo função llopen()

- Função llwrite()

```c
int llwrite(char *buf, int bufSize)
{

    int i = 0, k = 0, inputSize = bufSize + 5, res;
    int stuffedSize;
    char *input = malloc(sizeof(char) * (inputSize));
    char stuffed[MAX_PAYLOAD_SIZE * 2];
    char bcc2 = 0;
    char help = 0, rr_aux = 0, rej_aux = 0;

    input[0] = FLAG;
    input[1] = A_T;
    input[2] = S; // atualizar S se receçao da resposta foi bem sucedida
    input[3] = input[1] ^ input[2];

    rr_aux = RR ^ R;
    rej_aux = REJ ^ R;

    S = S ^ 2;
    R = R ^ 32;

    // BCC2 creation
    for (i = 0; i < bufSize; i++)
    {
        bcc2 ^= buf[i];
        input[i + HEADER_SIZE] = buf[i];
    }

    input[bufSize + HEADER_SIZE] = bcc2;

    // Byte Stuffing
    // n tenho de ir verificar bit a bit, porque a leitura é feita byte a byte
    stuffed[0] = FLAG;
    k = 0;
    for (i = 1; i < inputSize; i++)
    {
```

(12)

```c
for (i = 1; i < inputSize; i++)
{
    if ((input[i] == 0x7e) || (input[i] == 0x7d))
    {
        stuffed[i + k] = 0x7d;
        k++;
        stuffed[i + k] = input[i] ^ 0x20;
    }
    else
    {
        stuffed[i + k] = input[i];
    }
}

stuffedSize = i + k;
stuffed[stuffedSize] = FLAG;

state = 0;
k = 0;
while (state < 2)
{

    switch (state)
    {

    case 0:

        if (k > ll.numTries)
        {
            printf("Dados não recebidos de volta, problemas de envio\n");
            return -1;
            break;
        }
```

(13)

```c
    if (k > 0)
    {
        printf("Retransmiting\n");
    }

    k++;
    sleep(0.00001);
    res = write(fd, stuffed, stuffedSize + 1);
    nI++;
    help = wait_for_answer();

    if (help == rr_aux)
    {
        state = 1;
        break;
    }
    if (help == rej_aux)
    {
        while (!read(fd, &help, 1)) // dar clear da last flag
        {
        }
        printf("Received REJ\n");
        state = 0;
        nREJ++;
        break;
    }
```

(14)

```c
            state = 0;
            break;

        case 1:

            // add code to verify rejection or not
            while (!read(fd, &help, 1))
            {
            }

            if (help != FLAG)
            {
                printf("Nao recebemos last FLAG\n");
                nerrors++;
                state = 0;
                break;
            }

            state++;
            return 1;

            break;
        }
    }
}
```

(15)

Fig. 12, 13,14,15 – Codigo função llwrite()

- Função llread()

```c
// Receive data in packet, which has already  MAXSIZE
int llread(char *packet)
{
    int packetSize = 0, res;
    char output[5];
    char bcc2 = 0;
    char aux;

    output[0] = FLAG;
    output[1] = A_T;
    output[4] = FLAG;

    state = 1;
    aux = wait_for_answer();

    if (aux == SET)
    {
        state = 1; // somente para ler a ultima flag
        llopen(ll);
        return 0;
    }

    if ((aux == 0) || (aux == 2))
    {
        nI++;
    }
    else
        return 0;

    R = (aux ^ 2) << 4;

    int j = 0;

    // como ja lemos anteriormente o HEADER, ja so temos DATA e BCC2 ate FLAG
    // FLAG also fica lida, need to tirar BCC2 de packet
```

(16)

```c
while (1)
{
    sleep(0.00001);
    res = read(fd, &packet[j], 1);
    if (res == 0)
    {
        printf("Badly read DATA\n");
        break;
    }
    if (packet[j] == FLAG)
    {
        j++;
        break;
    }
    if (packet[j] == 0x7d)
    {
        sleep(0.00001);
        (void)read(fd, &packet[j], 1);
        packet[j] ^= 0x20;
    }
    bcc2 ^= packet[j];
    j++;
}

packetSize = j;
// debugging code
```

(17)

```c
    packetSize = j;
    // debugging code

    // flag += data_error(PROBABILITY);
    // printf("flag->%d\n", flag);
    // data_error adiciona erro ficticio ao cenas nos Dados para verifying
    if (bcc2 || flag == 1)
    {
        printf("Error in received data (BCC2), sending REJ\n");
        nerrors++;
        nREJ++;
        output[2] = REJ ^ R;
        output[3] = output[1] ^ output[2];
        res = write(fd, output, 5);
        return 0;
    }

    output[2] = RR ^ R;
    output[3] = output[1] ^ output[2];
    res = write(fd, output, 5);

    return packetSize - 2;
}
```

(18)

Fig. 16, 17,18 – Codigo função llread()

- Função llclose()

```c
int llclose(int showStatistics)
{

    char output[SUP_SIZE] = {0}, ack[SUP_SIZE] = {0};
    int k = 0, res = 0;
    char aux = 0;

    output[0] = FLAG;
    output[1] = A_T;
    output[2] = DISC;
    output[3] = A_T ^ DISC;
    output[4] = FLAG;

    ack[0] = FLAG;
    ack[1] = A_T;
    ack[2] = UA;
    ack[3] = BCC1_R;
    ack[4] = FLAG;

    printf("Entramos em llclose\n");
    switch (ll.role)
    {

    case TRANSMITTER:
        state = 0;

        while (state != 3)
        {

            switch (state)
            {
```

(19)

```c
// escrever informação e verificar
case 0:

    if (k > ll.numTries)
    {
        printf("Dados não recebidos de volta, problemas de envio\n");
        return -1;
    }

    k++;
    res = write(fd, output, SUP_SIZE);
    aux = wait_for_answer();
    if (aux == DISC)
    {
        printf("Recebemos DISC llopen\n");
        state = 1;
        break;
    }
    nerrors++;
    printf("Wait for answer error\n");
    break;

    // receber informação
case 1:
    while (!read(fd, &aux, 1))
    {
    }

    if (aux != FLAG)
    {
        nerrors++;
        printf("Nao recebemos last FLAG, trying again\n");
        state = 0;
        break;
    }

    state++;
    break;
```

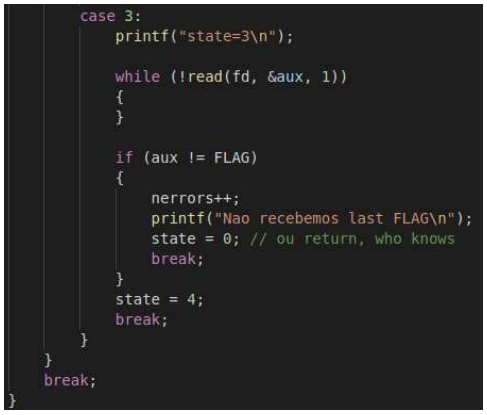
(20)

```c
        case 2:
            printf("UA written back\n");
            write(fd, ack, SUP_SIZE);
            state++;

            break;
        }
    }

    break;

case RECEIVER:

    state = 0;
    while (state != 4)
    {
        // Receiver
        switch (state)
        {
        case 0:
            state = 1;
            aux = wait_for_answer();
            if (aux == DISC)
            {
                printf("Header well received\n");
                state = 1;
                break;
            }

            nerrors++;
            printf("Header badly received\n");
            state = 0; // ou return 0, who knows honestly
            break;
```

(21)

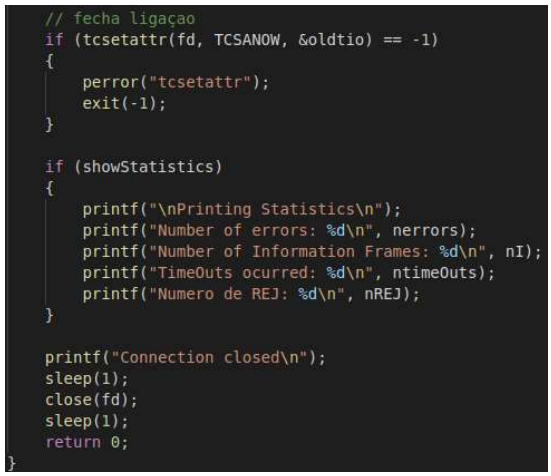
```c
case 1:
    while (!read(fd, &aux, 1))
    {
    }
    if (aux != FLAG)
    {
        nerrors++;
        printf("Nao recebemos last FLAG\n");
        state = 0; // ou return, who knows
        break;
    }

    printf("%ld bytes Written back\n", write(fd, output, SUP_SIZE));
    state++;

    break;

case 2:
    state = 1;
    sleep(0.00001);
    aux = wait_for_answer();
    if (aux == UA)
    {
        printf("Header well received\n");
        state = 3;
        break;
    }
    nerrors++;
    printf("Header badly received\n");
    state = 0; // ou return 0, who knows honestly

    break;
```

(22)



```
        case 3:
            printf("state=3\n");

            while (!read(fd, &aux, 1))
            {
            }

            if (aux != FLAG)
            {
                nerrors++;
                printf("Nao recebemos last FLAG\n");
                state = 0; // ou return, who knows
                break;
            }
            state = 4;
            break;
        }
    }
    break;
}
```

(23)



```
    // fecha ligaçao
    if (tcsetattr(fd, TCSANOW, &oldtio) == -1)
    {
        perror("tcsetattr");
        exit(-1);
    }

    if (showStatistics)
    {
        printf("\nPrinting Statistics\n");
        printf("Number of errors: %d\n", nerrors);
        printf("Number of Information Frames: %d\n", nI);
        printf("TimeOuts ocurred: %d\n", ntimeOuts);
        printf("Numero de REJ: %d\n", nREJ);
    }

    printf("Connection closed\n");
    sleep(1);
    close(fd);
    sleep(1);
    return 0;
}
```

(24)

Fig. 19,20,21,22,23,24 – Codigo função llread()