



ENGENHARIA DA COMPUTAÇÃO
REDES DE COMPUTADORES A

ATIVIDADE 1

EQUIPE:

Agostinho Sanches de Araújo -----	RA: 16507915
Evandro Douglas Capovilla Junior -----	RA: 16023905
Lucas Tenani Felix Martins -----	RA: 16105744
Pedro Andrade Caccavaro -----	RA: 16124679

12/03/2019



SUMÁRIO

INTRODUÇÃO.....	03
OBJETIVO.....	03
DESCRIÇÃO.....	04
OBSERVAÇÕES.....	06
RESULTADO.....	07
CONCLUSÃO.....	09



INTRODUÇÃO

User Datagram Protocol, conhecido como UDP, é um protocolo da camada de transporte não orientado à conexão, ou seja, não é necessário estabelecer conexão antes de enviar pacote. UDP é muito utilizado quando se trata de streaming de áudio e vídeo, pois é necessário o transporte rápido dos dados, entretanto esse desempenho possui a desvantagem de não garantir que o pacote seja recebido com sucesso, através de retransmissões ou confirmações, portanto é importante atentar ao tratamento dessa comunicação nos programas que serão utilizados. Com isso, foi desenvolvido a atividade de realizar a comunicação de cliente e servidor através do protocolo de rede UDP.

OBJETIVO

A atividade tem como principal objetivo demonstrar a interação de dois programas diferentes se comunicando através da rede. Para efetuar a comunicação via rede, foi utilizado um cliente UDP e um servidor UDP, onde o usuário cliente envia uma mensagem (comando do cmd) para o servidor, e o mesmo identifica essa mensagem e retorna uma resposta para o cliente.

DESCRIÇÃO

A atividade tem início com a execução do programa servidor, no qual é passado um parâmetro correspondente a porta que será utilizada. Assim, um *socket* é criado e o servidor é setado com as devidas informações, sendo elas:

- `server.sin_family = AF_INET` (tipo de endereço)
- `server.sin_port = port` (porta que foi passada por parâmetro)
- `server.sin_addr.s_addr = INADDR_ANY` (faz com que o servidor fique ligado em todos os endereços IP)

Após setar as devidas informações para o servidor, é feita uma conexão (*bind*) entre o *socket* criado e o próprio servidor. Essa conexão faz com que o servidor fique em uma porta mandando ou recebendo informações de usuários que consigam ter acesso a mesma porta. Efetuando a conexão, o servidor irá ficar esperando uma mensagem do programa cliente (espera ocupada).

O programa cliente será executado passando como parâmetro o endereço IP (endereço local da máquina - 127.0.0.1) e a mesma porta que foi utilizada para o programa servidor. Após a execução do programa cliente, um *socket* é criado e o servidor cliente é setado do mesmo formato que o programa servidor. O usuário cliente digita uma mensagem sendo equivalente a um comando no `cmd` e é comparado a mensagem “*exit*” antes de mandar para o servidor. Caso a comparação dê verdadeiro, o programa cliente é encerrado.

Após o envio da mensagem do programa cliente para o programa servidor, o programa cliente fica esperando uma resposta do programa servidor (espera ocupada). O servidor lê a mensagem e a função *popen()* converte a mensagem digitada em um ponteiro que indicará um arquivo que contém a resposta para o comando. Caso a função não consiga encontrar um arquivo no qual a mensagem seja correspondente, há uma mensagem de erro mostrada no programa servidor e é enviado para o programa cliente o valor vazio.

O programa servidor manda uma resposta para o programa cliente e a mensagem é mostrada. Ambos os programas continuam em execução até que o usuário cliente digite a mensagem “*exit*”. O programa servidor fica em *loop* infinito.

Abaixo, todas as funções que foram apresentadas para a realização da atividade com as eventuais explicações sobre seus funcionamentos, parâmetros que foram utilizados e seus retornos.

- **htons(uint16_t hostshort);**
Converte o inteiro não sinalizado *hostshort* de bytes em ordem de host para bytes em ordem de rede.
- **int socket(int domain, int type, int protocol);**
A função *socket* cria um terminal de comunicação.
- **in_addr_t inet_addr(const char *cp);**
Converte o endereço IP números-e-pontos em dados binários na ordem de bytes da rede.
- **void *memset(void *s, int c, size_t n);**
Preenche os *n* primeiros bytes da área de memória apontada por *s* com a constante *c*.
- **strcmp(const char *s1, const char *s2);**
Compara duas strings *s1* e *s2*, o valor de retorno é negativo para *s1* maior que *s2*, nulo para strings iguais e positivo para *s2* maior que *s1*.
- **void *memcpy(void *dest, const void *src, size_t n);**
Copia *n* bytes da área de memória *src* para a área de memória *dest*.
- **ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);**
Transmite uma mensagem *buf*, de tamanho *len*, do socket *sockfd* para o socket *dest_addr*.
- **ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen);**
Recebe uma mensagem *buf*, de tamanho *len*, do socket *src_addr* para o socket *sockfd*.
- **int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);**
Atribui o endereço especificado no socket pelo arquivos descritor.
- **int close(int fd);**



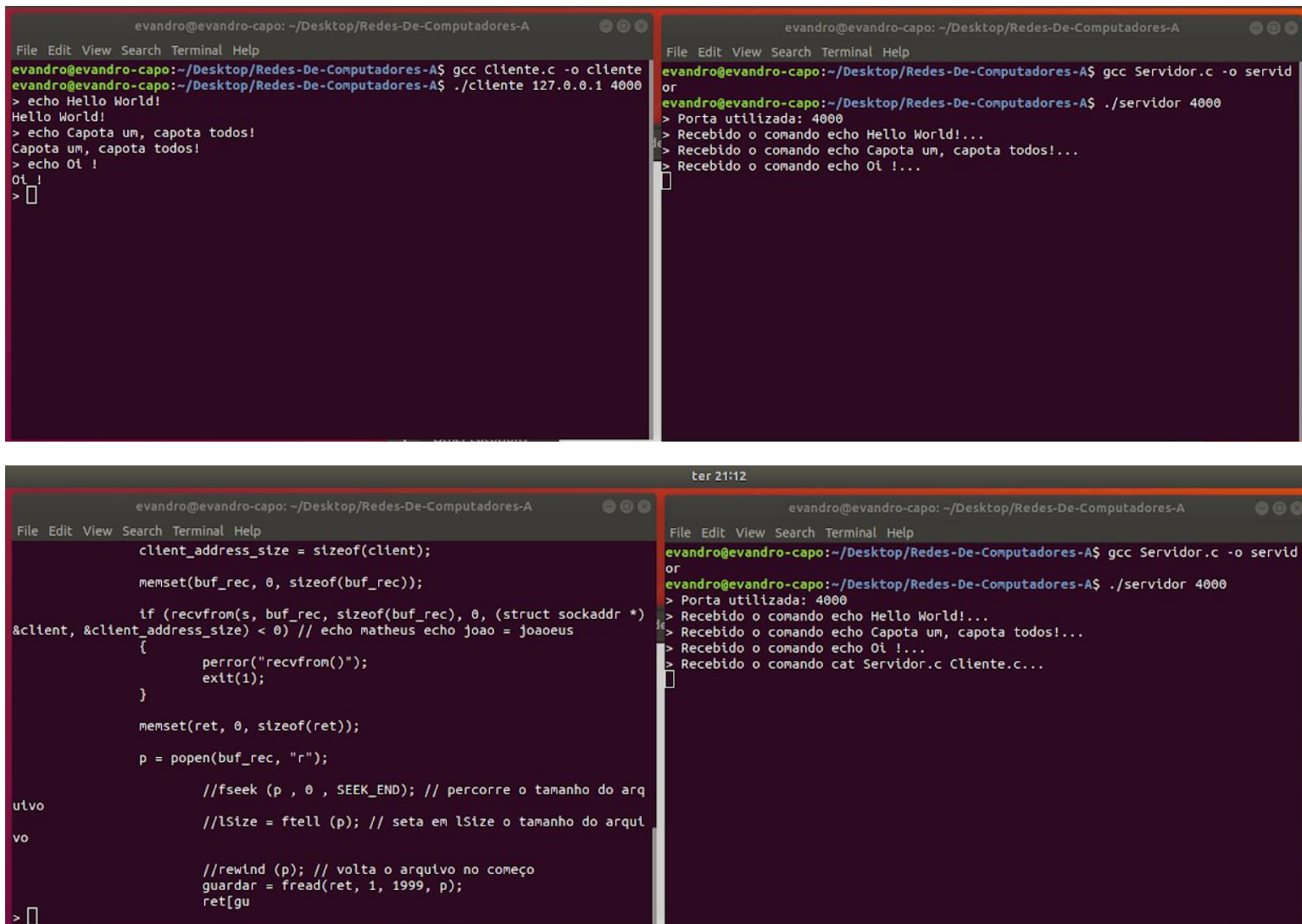
Desaloca da memória o socket *fd*.

- **FILE *popen(const char *command, const char *type);**
Executa um processo solicitado invocando o shell.

OBSERVAÇÕES

Alguns comandos não retornam mensagem, como é o caso do *MKDIR*, que cria um diretório, portanto ao executá-lo, o servidor envia um *buffer* vazio ao cliente, ocasionando a falta de um retorno de sucesso da execução, assim como ocorre quando é enviado um comando inválido pelo cliente, gerando a mensagem de falha no terminal do servidor no qual não foi possível retorná-lo.

RESULTADOS



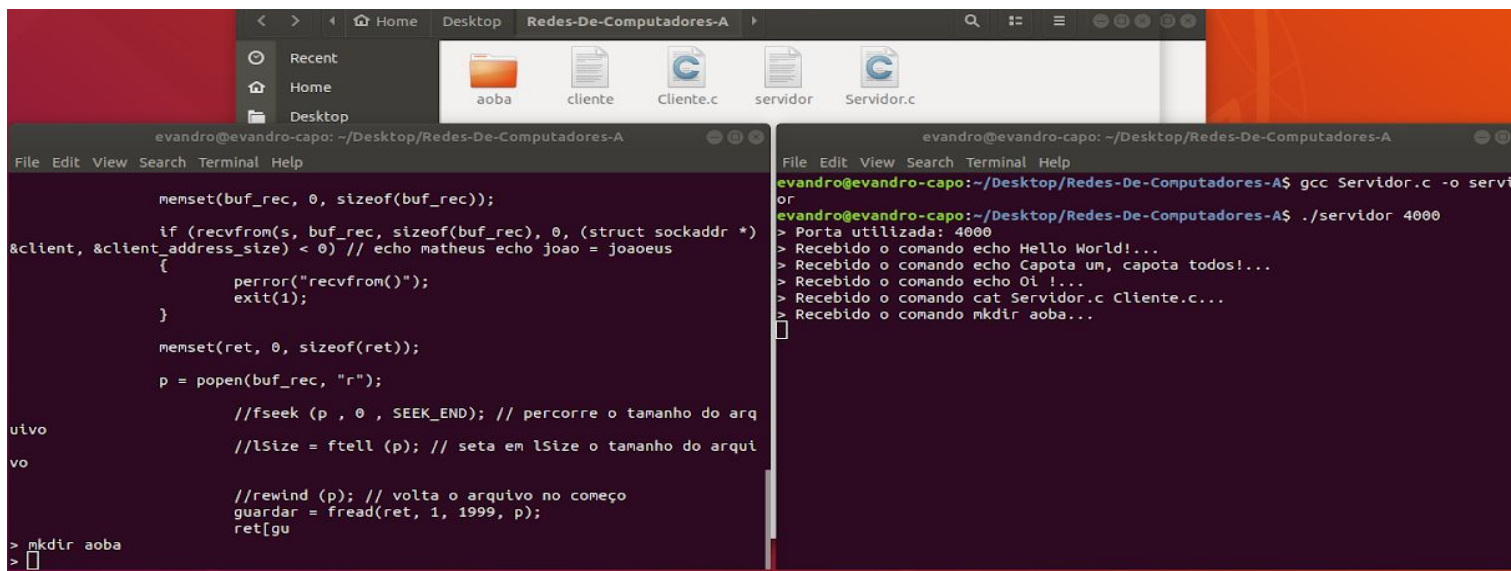
```
evandro@evandro-capo: ~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ gcc Cliente.c -o cliente
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ ./cliente 127.0.0.1 4000
> echo Hello World!
Hello World!
> echo Capota um, capota todos!
Capota um, capota todos!
> echo Oi !
Oi !
>

evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ gcc Servidor.c -o servid
or
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ ./servidor 4000
> Porta utilizada: 4000
> Recebido o comando echo Hello World!...
> Recebido o comando echo Capota um, capota todos!...
> Recebido o comando echo Oi !...

evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help
client_address_size = sizeof(client);
memset(buf_rec, 0, sizeof(buf_rec));
if (recvfrom(s, buf_rec, sizeof(buf_rec), 0, (struct sockaddr *)
&client, &client_address_size) < 0) // echo matheus echo joao = joaoeus
{
    perror("recvfrom()");
    exit(1);
}
memset(ret, 0, sizeof(ret));
p = popen(buf_rec, "r");
//fseek (p , 0 , SEEK_END); // percorre o tamanho do arq
//lSize = ftell (p); // seta em lSize o tamanho do arqui
//rewind (p); // volta o arquivo no começo
guardar = fread(ret, 1, 1999, p);
ret[gu
>

evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ gcc Servidor.c -o servid
or
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ ./servidor 4000
> Porta utilizada: 4000
> Recebido o comando echo Hello World!...
> Recebido o comando echo Capota um, capota todos!...
> Recebido o comando echo Oi !...
> Recebido o comando cat Servidor.c Cliente.c...
```

Nas imagens acima utilizamos os comandos ECHO e CAT para o servidor retornar as mensagens solicitadas não ultrapassando o limite de 2000 caracteres.



```

evandro@evandro-capo: ~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help

    memset(buf_rec, 0, sizeof(buf_rec));

    if (recvfrom(s, buf_rec, sizeof(buf_rec), 0, (struct sockaddr *)
&client, &client_address_size) < 0) // echo matheus echo joao = joaoeus
    {
        perror("recvfrom()");
        exit(1);
    }

    memset(ret, 0, sizeof(ret));

    p = popen(buf_rec, "r");

    //fseek (p , 0 , SEEK_END); // percorre o tamanho do arq
uivo
    //lSize = ftell (p); // seta em lSize o tamanho do arqui
vo

    //rewind (p); // volta o arquivo no começo
    guardar = fread(ret, 1, 1999, p);
    ret[gu

> mkdir aoba
>

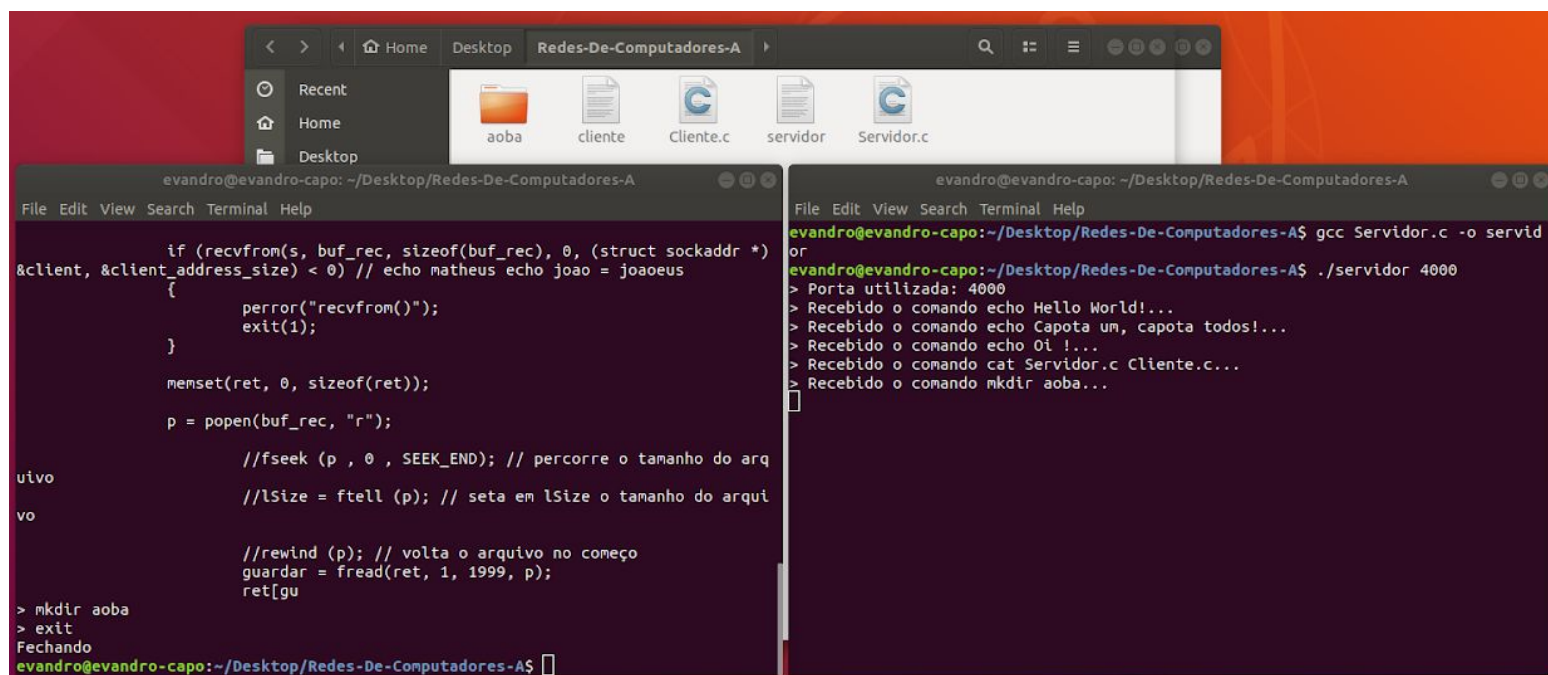
```

```

evandro@evandro-capo: ~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help

evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ gcc Servidor.c -o servid
or
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ ./servidor 4000
> Porta utilizada: 4000
> Recebido o comando echo Hello World!...
> Recebido o comando echo Capota um, capota todos!...
> Recebido o comando echo Oi !...
> Recebido o comando cat Servidor.c Cliente.c...
> Recebido o comando mkdir aoba...

```



```

evandro@evandro-capo: ~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help

    if (recvfrom(s, buf_rec, sizeof(buf_rec), 0, (struct sockaddr *)
&client, &client_address_size) < 0) // echo matheus echo joao = joaoeus
    {
        perror("recvfrom()");
        exit(1);
    }

    memset(ret, 0, sizeof(ret));

    p = popen(buf_rec, "r");

    //fseek (p , 0 , SEEK_END); // percorre o tamanho do arq
uivo
    //lSize = ftell (p); // seta em lSize o tamanho do arqui
vo

    //rewind (p); // volta o arquivo no começo
    guardar = fread(ret, 1, 1999, p);
    ret[gu

> mkdir aoba
> exit
Fechando
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$

```

```

evandro@evandro-capo: ~/Desktop/Redes-De-Computadores-A
File Edit View Search Terminal Help

evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ gcc Servidor.c -o servid
or
evandro@evandro-capo:~/Desktop/Redes-De-Computadores-A$ ./servidor 4000
> Porta utilizada: 4000
> Recebido o comando echo Hello World!...
> Recebido o comando echo Capota um, capota todos!...
> Recebido o comando echo Oi !...
> Recebido o comando cat Servidor.c Cliente.c...
> Recebido o comando mkdir aoba...

```

As duas imagens acima mostra a criação da pasta utilizando o comando MKDIR enviado do cliente para o servidor e o comando EXIT que encerra apenas o cliente.



CONCLUSÃO

O foco durante a realização deste trabalho fora a melhor compreensão acerca do protocolo UDP, estudando suas definições e aplicações para poder implementar o sistema Cliente/Servidor. Ainda não tendo implementado o protocolo TCP não foi tirada nenhuma conclusão no sentido comparativo entre os dois protocolos, apenas confirmaram-se as informações passadas na teoria sobre a relação entre o cliente e servidor.