PUC CAMPINAS
PONTIFÍCIA UNIVERSIDADE CATÓLICA

Redes de Computadores A

Campinas, 12 de Marco de 2019

1ª Atividade

NOME:	**RA:**
Ettore Biazon Baccan	16000465
Mateus Henrique Zorzi	16100661
Matheus Martins Pupo	16145559
Murilo Martos Mendonça	16063497
Victor Hugo do Nascimento	16100588

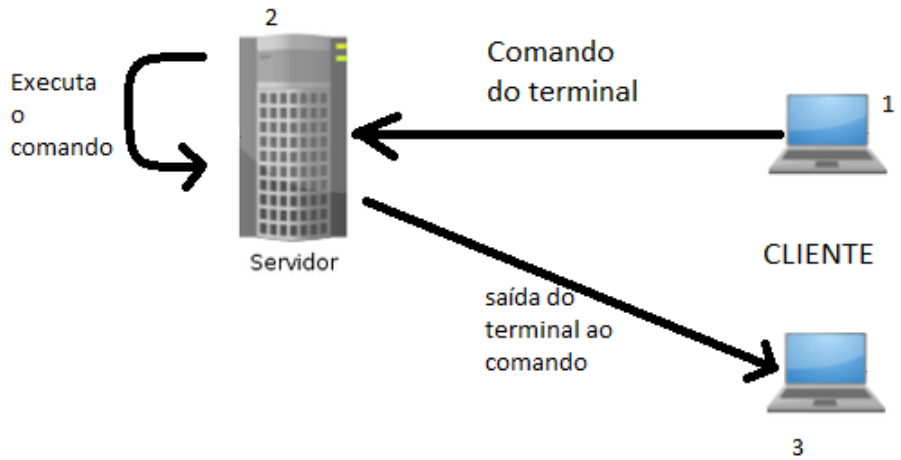
Introdução

Nessa atividade foi implementado um modelo de cliente-servidor UDP.

O UDP, diferente do TCP, não é orientado a conexão, ou seja, não é necessário que seja estabelecida uma conexão prévia entre o cliente e o servidor, por outro lado, em cada pacote enviado deve haver o endereço que se deseja enviar e a porta que será utilizada.

A principal vantagem do UDP em relação ao TCP é a facilidade de implementação, já que vários clientes podem enviar pacotes facilmente ao mesmo servidor, apenas conhecendo o endereço IP e a porta usada. Por outro lado, a principal desvantagem encontrada é a falta de confiabilidade, já que no UDP não há nenhum tipo de tratamento que confirme o envio ou recebimento do(s) pacote(s), caso uma das “pontas” (cliente ou servidor) se desconecte e pacotes ainda estejam sendo enviados, eles serão simplesmente perdidos.

Implementação



Cria-se o socket no cliente, para que a conexão possa ser estabelecida

```
if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
    perror("socket()");
    exit(1);
}
```

Em seguida, associamos o servidor ao socket criado

```
if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
{
    perror("bind()");
    exit(1);
}
```

Ao ser iniciado, o servidor aguarda um comando vindo do cliente.

```
if (recvfrom(s, buf, sizeof(buf), 0, (struct sockaddr *)&client, &client_address_size) < 0)
{
    perror("recvfrom()");
    exit(1);
}
```

Buf é o vetor onde a mensagem será armazenada.

```
fp = popen(buf, "r");
if (fp == NULL)
```

Quando o comando é recebido, o servidor o executa através do comando “popen”, que permite capturar a saída do terminal.

```
fread(resposta, (sizeof(resposta) + 1), 1, fp);
```

Com o “fread” pegamos o conteúdo apontado por fp e o armazenamos em resposta.

```
if ((size_sendto = sendto(s, resposta, strlen(resposta) + 1, 0, (struct sockaddr *)&client, sizeof(client))) < 0)
{
    perror("sendto()");
    exit(1);
}
```

Então enviamos o resultado da execução de volta ao cliente. (Armazenamos o retorno do sendto para verificarmos se a função está funcionando de acordo com o esperado).

```
if ( retorno_recv = recvfrom(s, res, sizeof(res), 0, (struct sockaddr *)&server, &server_address_size) < 0)
{
    perror("recvfrom()");
    exit(2);
}
```

O cliente recebe a mensagem do servidor e a exibe na tela.

O programa cliente todo roda dentro de um “do-while”, para que seja encerrado quando o usuário digitar “exit”. Já o servidor, quando isso acontece, continua esperando por uma mensagem.

Resultados

```
murilo@murilo:~/Downloads/Redes-de-Computadores-A$ ./servidor 1
**SERVIDOR INICIADO**
1
bind(): Permission denied
murilo@murilo:~/Downloads/Redes-de-Computadores-A$ ./servidor 1024
**SERVIDOR INICIADO**
1024
bind(): Permission denied
murilo@murilo:~/Downloads/Redes-de-Computadores-A$
```

As portas 1 a 1024 são reservadas,por isso, pedem acesso root para utilizá-las.

```
murilo@murilo:~/Downloads/Redes-de-Computadores-A$ ./cliente 127.0.0.1 6000
**CLIENTE INICIADO**
6000
> ls
RETORNO => 63
resposta do servidor ao comando ls:

cliente
cliente.c
README.md
servidor
servidor.c
teste
teste.c


O endereco e: 0
Porta utilizada eh: 35756
>
```

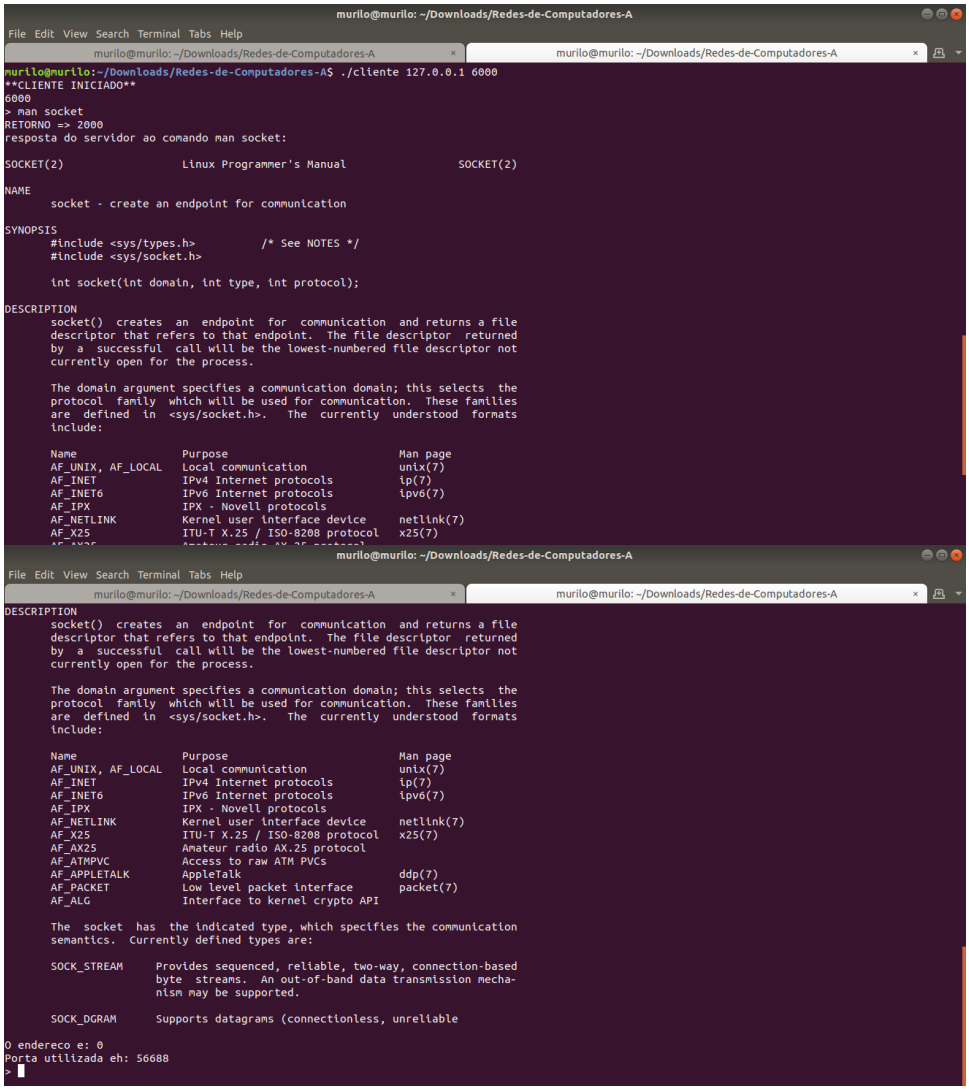
O comando "ls”, que retorna as pastas e arquivos no diretório atual. Exibimos também a quantidade de caracteres enviados/recebidos a fim de conferência.

```
murilo@murilo:~/Downloads/Redes-de-Computadores-A$ ./servidor 6000
**SERVIDOR INICIADO**
6000
o endereco e: 0
Porta utilizada eh: 6000

Recebida a mensagem 'ls' do endereco IP 127.0.0.1 da porta 35756

 SIZE SENDTO: 63

 STRLEN RESPOSTA + 1: 63
```

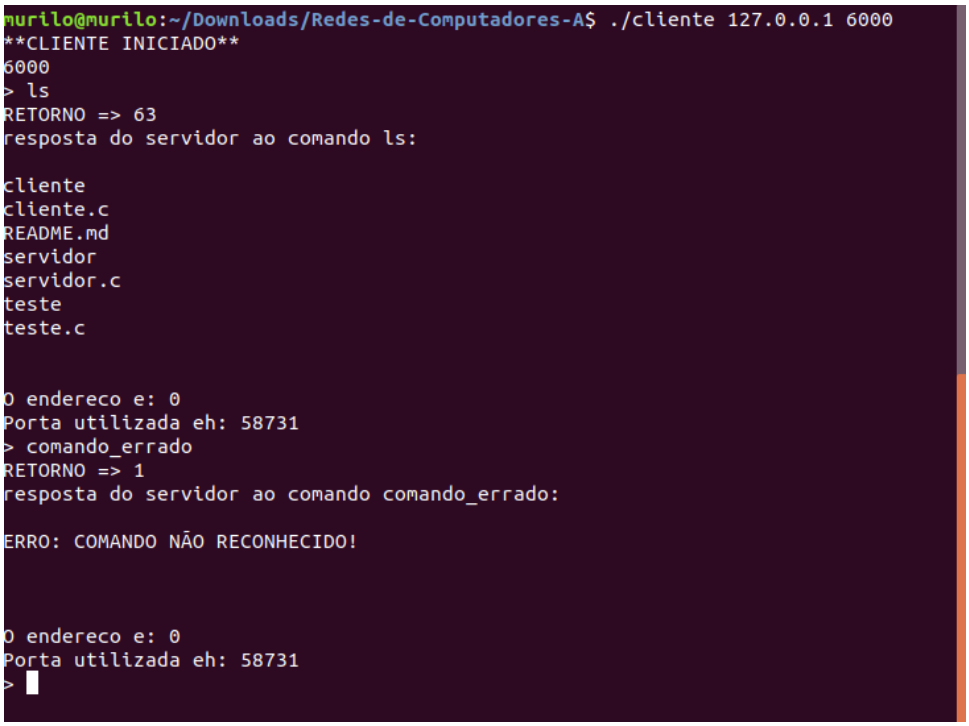



```
murilo@murilo:~/Downloads/Redes-de-Computadores-A$ ./servidor 6000
**SERVIDOR INICIADO**
6000
o endereco e: 0
Porta utilizada eh: 6000
Recebida a mensagem 'man socket' do endereco IP 127.0.0.1 da porta 56688

 SIZE SENDTO: 2000

 STRLEN RESPOSTA + 1: 2000
```

Quando o cliente requisita um comando não reconhecido pelo servidor, mostramos uma mensagem avisando ao mesmo sobre o erro.



Conclusão

Com a realização do experimento pudemos concluir que a implementação de programas cliente-servidor UDP é bem simplificada, ainda que não haja certeza de recebimento dos pacotes, esse tipo de aplicação é útil em muitos casos. Um bom exemplo disso são as vídeo chamadas, caso algum pacote seja perdido no caminho, a experiência da chamada não é gravemente afetada, ocorrerão pequenos cortes, serrilhados ou perda de resolução momentâneos. Seria muito pior, por exemplo, se para evitar perda, os pacotes que não puderam ser entregues fossem armazenados e enviados novamente, o que não seria natural em uma conversa.

Portanto, percebemos que há diferenças entre o TCP e o UDP e que os dois são úteis, dependendo da necessidade de cada aplicação. Como nosso programa é limitado, nossas execuções são sempre semelhantes, pois o cliente envia a mensagem e o servidor apenas à executa e retorna a resposta ao cliente, com isso, podemos concluir que o UDP é o melhor método para nossa aplicação, principalmente pela não necessidade em se guardar todos os pacotes recebidos.