



UNIVERSIDADE FEDERAL DE MINAS

GERAIS

DEPARTAMENTO DE CIÊNCIA DA
COMPUTAÇÃO

Julio Guimarães

Redes de Computadores

Belo Horizonte

Contents

1	Introdução	1
1.1	Múltiplas conexões	2
2	Implementação	5
2.1	Programa receptor	5
2.2	Programa transmissor	6
3	Metodologia	8
4	Conclusão	12

1 Introdução

A transferência de arquivos através de redes de computadores está, quase sempre, sujeita a falhas. Para alguns serviços, tais como o streaming de vídeo em tempo real, a perda de alguns pacotes de dados pode ter menos importância do que para outros, como o download de um arquivo compactado na Internet. No primeiro caso, a perda de alguns pacotes provavelmente só provocaria uma falha de menos de um segundo no vídeo, comprometendo pouco a qualidade da transmissão. No segundo, entretanto, uma falha no recebimento de pequenas porções do arquivo poderia corrompê-lo por completo. Existem, portanto, diversos métodos para permitir uma transferência confiável de dados entre computadores. Alguns deles são serviços orientados à conexões, enquanto outros são estabelecidos sobre protocolos de transferência de dados não orientados a conexões.

Neste Trabalho Prático, foi utilizado o UDP (User Datagram Protocol, definido no RFC 768) para implementar duas aplicações (um transmissor e um receptor de dados) que podem ser utilizados para transferir um arquivo entre dois computadores através da rede. O UDP é um dos principais protocolos usados na Internet. Utilizando o UDP, um computador pode mandar mensagens (chamadas, nesse caso, de datagramas) para outro computador, sem a necessidade de comunicações prévias para estabelecimento de conexões.

Neste trabalho, o servidor terá que, além de se comunicar utilizando o protocolo UDP, terá que ser capaz de atender a vários clientes, ou seja, aceitar múltiplas conexões. Dessa maneira, o servidor poderá por exemplo enviar mensagens diferentes para computadores diferentes de acordo com as solicitações, tudo sendo realizado de forma que o servidor conseguirá gerenciar toda a comunicação.

1.1 Múltiplas conexões

Para ser possível que o servidor tenha suporte a múltiplas conexões, foi utilizado o conceito de programação paralela na linguagem C, as PThreads. Um processo represente a execução pelo sistema operativo dum programa. Uma thread represente uma linha de execução das instruções deste programa. Um processo poderá conter várias threads, pelo menos uma.

Quando um programa começa a executar terá um processo com uma thread a executar. Mais threads são depois criadas com a função `pthread_create()` e destruídas com a função `pthread_exit()`.

```
while (1){
    newsockfd = accept (sockfd,
        (struct sockaddr *) &from_addr,&from_addr_len);
    if (newsockfd < 0)
        error ("Não aceito.");

    n = pthread_create( &thread_id , NULL , comunicar , (void*) &newsockfd);
    if( n < 0)
        error("Nao foi possivel criar a thread");
    pthread_join( thread_id , NULL);
}
```

Notas Explicativas:

A chamada à função `pthread_create()` tem quatro argumentos. O primeiro é usado para guardar informação sobre a thread criada. O segundo especifica algumas propriedades da thread a criar, utilizaremos o valor `NULL` para significar os valores por defeito. O Terceiro é a função que a nova thread vai executar e o ultimo é usado para representar argumentos a esta função.

A chamada à função `pthread_exit()` provoca a terminação da thread e a libertação dos recursos que esta está a consumir. Aqui não há realmente necessidade para usar esta função porque quando a função da thread termine a thread seria destruída. A função é apenas útil se for necessário destruir a thread no meio da sua execução.

A função a ser executada por uma nova thread tem sempre o formato:

$$\text{void} * \text{funcao} (\text{void} * \text{argumentos});$$

Então, eu adotei o método de criar uma thread toda vez que um cliente

fizer uma solicitação ao servidor, dessa maneira é possível isolar cada comunicação em uma thread diferente, facilitando a gerencia. Além de ter uma implementação simples e de fácil entendimento, uma vez que já havia utilizado as pthreads em C.

2 Implementação

Como já apresentado, dois programas foram implementados neste trabalho: um transmissor e um receptor de arquivos. O transmissor pode ser usado para enviar um determinado arquivo, através da rede, para um computador que esteja executando o programa receptor. O receptor, portanto, pode ser usado para receber um arquivo enviado por um transmissor. Cada programa foi implementado individualmente, mas ambos compartilham todos os outros módulos construídos para as aplicações. A implementação dos programas foi feita na linguagem C++ e em sistema operacional Linux. Para implementação da comunicação, foram utilizados sockets UDP, associados a portas do sistema. Onde a ordem e os valores dos parâmetros foram seguidos de acordo com a especificação do trabalho.

2.1 Programa receptor

O programa receptor é responsável por receber um arquivo, enviado por um programa transmissor em execução no computador de origem. Um socket é criado e adequadamente inicializado, e o receptor então aguarda pelo recebimento dos datagramas que contém os dados do arquivo. A primeira mensagem (não necessariamente a primeira a ser recebida) guarda o nome do arquivo, utilizado para criá-lo, e os próximos quadros trazem os bytes o arquivo. O receptor verifica se cada quadro de dados recebido possui um dos identificadores esperados em sua janela e, em caso afirmativo, armazena

o quadro. Quando possível, os quadros são escritos no arquivo e a janela avança, conforme explicado na Seção 2. Caso o identificador tenha um valor menor do que os esperados, o datagrama contém dados que já foram escritos no arquivo, e a única atitude do receptor é enviar o ACK correspondente. Um valor de ID maior do que os esperados indica que o quadro deve ser temporariamente ignorado pelo receptor. Quando o último datagrama contendo dados do arquivo é recebido, eles são finalmente escritos e a aplicação termina.

2.2 Programa transmissor

O programa transmissor é responsável por enviar um determinado arquivo para o programa receptor em execução no computador destino. Além disso, como já dito anteriormente, o programa transmissor ele será capaz de receber vários clientes receptores, com isso será utilizado o conceito de threads para "encapsular" e paralelizar a execução e comunicação entre o servidor e o cliente, dessa maneira em cada thread de execução um socket é criado e adequadamente inicializado e o arquivo é então enviado. Os bytes do arquivo são enviados em quadros armazenados na janela do transmissor. A primeira mensagem enviada guarda o nome do arquivo. Após preencher a janela, o transmissor aguarda pela confirmação do receptor, dada pelo ACK correspondente a um dos quadros, para que possa avançar a janela, conforme o algoritmo explicado na Seção 2. Finalizada a transmissão do arquivo, o

programa transmissor termina.

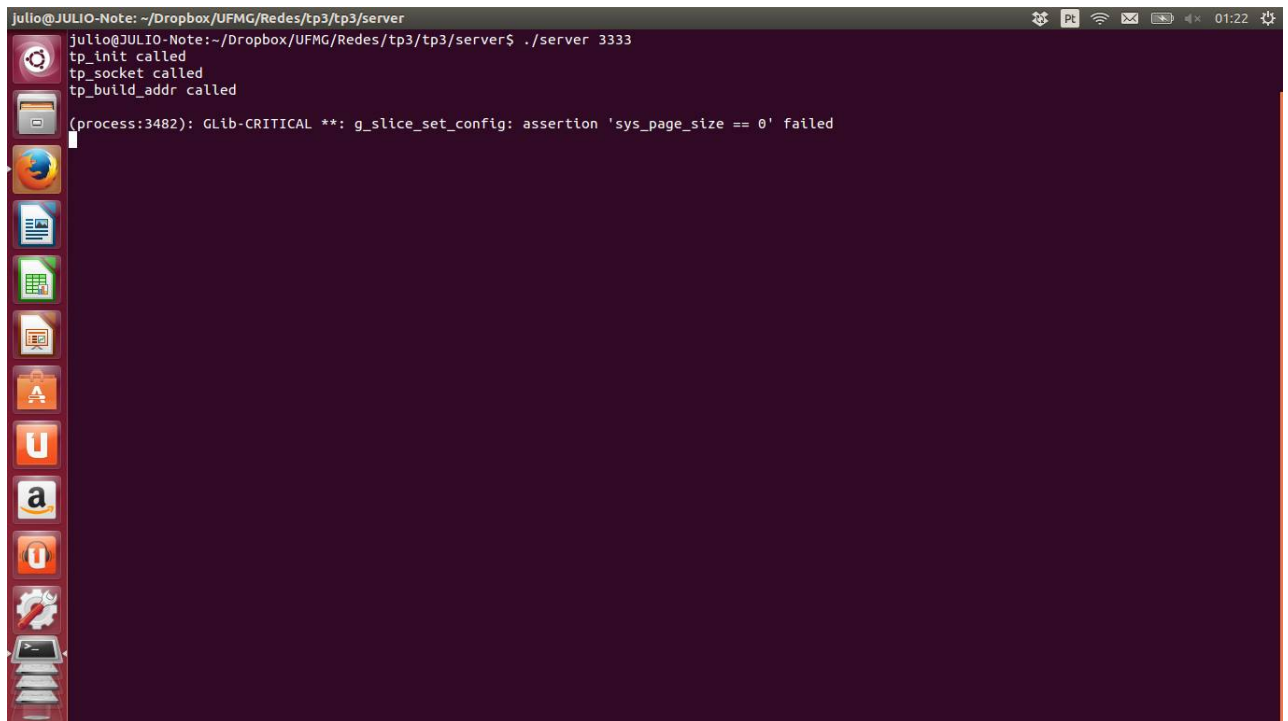
3 Metodologia

Para realizar os testes de desempenho, foi utilizado uma máquina Linux FEDORA 64 bits, 8 GB de memória RAM, Intel Core i5, @2.3 GHz e outra máquina Linux FEDORA 64 bits, 4 GB de memória RAM, Intel core 2 QUAD @2.33 GHz.

Estas duas máquinas se encontravam ligadas no mesmo roteador, ou seja, ambas estavam na mesma rede local. Uma utilizava conexão wireless e obteve IPv4: 192.168.0.11 e outra estava com conexão utilizando cabo Ethernet e recebeu IPv4: 192.168.0.7. A porta utilizada para os testes foi 3333.

Os testes foram feito de última hora, apenas para verificar que o trabalho estava funcionando como especificado. Porém nenhuma análise foi gerada em cima dos resultados por falta de tempo. Porém algumas imagens dos testes e dos resultados obtidos seguem abaixo.

Servidor sendo executado na porta (3333), passada por parâmetro.



```
julio@JULIO-Note: ~/Dropbox/UFG/Redes/tp3/tp3/server
julio@JULIO-Note:~/Dropbox/UFG/Redes/tp3/tp3/server$ ./server 3333
tp_init called
tp_socket called
tp_build_addr called
(process:3482): GLib-CRITICAL **: g_slice_set_config: assertion 'sys_page_size == 0' failed
```

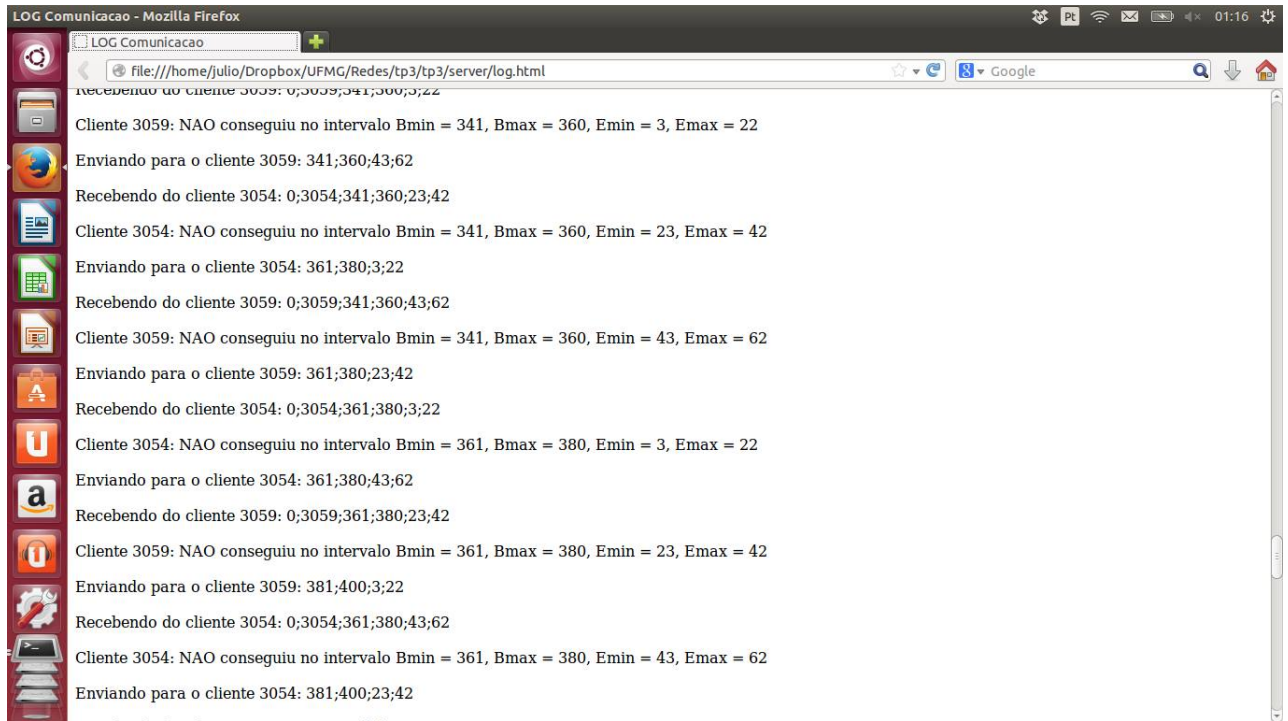
The image shows a terminal window on a Linux desktop. The terminal title bar reads 'julio@JULIO-Note: ~/Dropbox/UFG/Redes/tp3/tp3/server'. The command prompt shows the user 'julio' at the host 'JULIO-Note' in the directory '~/Dropbox/UFG/Redes/tp3/tp3/server'. The command executed is './server 3333'. The output shows several initialization messages: 'tp_init called', 'tp_socket called', and 'tp_build_addr called'. Following these, a GLib-CRITICAL error message is displayed: '(process:3482): GLib-CRITICAL **: g_slice_set_config: assertion 'sys_page_size == 0' failed'. The desktop background is dark purple, and a vertical dock on the left contains icons for various applications including a file manager, web browser, and office suite.

Cliente sendo executados de forma simultânea.

```
jullo@JULIO-Note: ~/Dropbox/UFGM/Redes/tp3/tp3/client
jullo@JULIO-Note: ~/Dropbox/UFGM/Redes/tp3/tp3/server x jullo@JULIO-Note: ~/Dropbox/UFGM/Redes/tp3/tp3/client x
jullo@JULIO-Note: ~/Dropbox/UFGM/Redes/tp3/tp3/client$ ./client 127.0.0.1 3333
tp_init called
tp_build_addr called
Conectado...
Envio: 0;2834;0;0;0
Valores recebidos: 1;20;3;22
Calculando...2834
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;2834;1;20;3;22
Valores recebidos: 1;20;23;42
Calculando...2834
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;2834;1;20;23;42
Valores recebidos: 1;20;43;62
Calculando...2834
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;2834;1;20;43;62
Valores recebidos: 21;40;3;22
Calculando...2834
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;2834;21;40;3;22
Valores recebidos: 21;40;23;42
Calculando...2834
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;2834;21;40;23;42
Valores recebidos: 21;40;43;62
Calculando...2834
Conseguir: 0
tp_build_addr called
```

```
Terminal
jullo@JULIO-Note: ~/Dropbox/UFGM/Redes/tp3/tp3/client
jullo@JULIO-Note: ~/Dropbox/UFGM/Redes/tp3/tp3/client x jullo@JULIO-Note: ~/Dropbox/UFGM/Redes/tp3/tp3/client x
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3054;281;300;3;22
Valores recebidos: 281;300;43;62
Calculando...3054
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3054;281;300;43;62
Valores recebidos: 301;320;23;42
Calculando...3054
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3054;301;320;23;42
Valores recebidos: 321;340;3;22
Calculando...3054
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3054;321;340;3;22
Valores recebidos: 321;340;43;62
Calculando...3054
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3054;321;340;43;62
Valores recebidos: 341;360;23;42
Calculando...3054
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3054;341;360;23;42
Valores recebidos: 361;380;3;22
Calculando...3054
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3054;341;360;23;42
Valores recebidos: 361;380;23;42
Calculando...3059
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3059;281;300;23;42
Valores recebidos: 301;320;3;22
Calculando...3059
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3059;301;320;3;22
Valores recebidos: 301;320;43;62
Calculando...3059
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3059;301;320;43;62
Valores recebidos: 321;340;23;42
Calculando...3059
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3059;321;340;3;22
Valores recebidos: 341;360;3;22
Calculando...3059
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3059;341;360;3;22
Valores recebidos: 341;360;43;62
Calculando...3059
Conseguir: 0
tp_build_addr called
Conectado...
Envio: 0;3059;341;360;43;62
Valores recebidos: 361;380;23;42
Calculando...3059
Conseguir: 0
tp_build_addr called
```

Página HTML gerada para que possa ser gerenciado a comunicação com todos os clientes:



4 Conclusão

No presente trabalho prático, foram implementados dois programas que permitem enviar arquivos através da Internet, utilizando um serviço capaz de suportar múltiplas conexões. Foram exercitados conceitos como a comunicação entre processos através de soquetes UDP e a implementação de protocolos, além da utilização de threads. O trabalho foi bastante interessante, e exigiu estudos sobre o funcionamento da comunicação através de soquetes UDP e a capacidade de utilizar algum método que permitisse que o servidor fosse capaz de aceitar múltiplos clientes, este método escolhido como explicado anteriormente foi as threads que possibilita uma paralelização da execução, essa ferramenta foi escolhida por atender a especificação do trabalho e por ser de fácil e conhecida implementação.