

Exercício 2.2 – Servidor TCP Concorrente

# **Aluna:** Naomi Takemoto

# **RA:** 184849

# Instituto de Computação

# Universidade Estadual de Campinas

# Novembro de 2020

# Instruções para execução

Para compilar os arquivos:

* make

Para limpar os executáveis:

* make clean

Para executar o servidor (porta é o número de porta de sua escolha):

* ./server <PORTA>

Para executar o cliente:

* ./cliente 127.0.0.1 <PORTA>

Para inserir/editar comandos de teste que serão enviados do servidor para os clientes

* colocar/retirar comandos do arquivo arq01.in

Para visualizar o ouput da saída padrão:

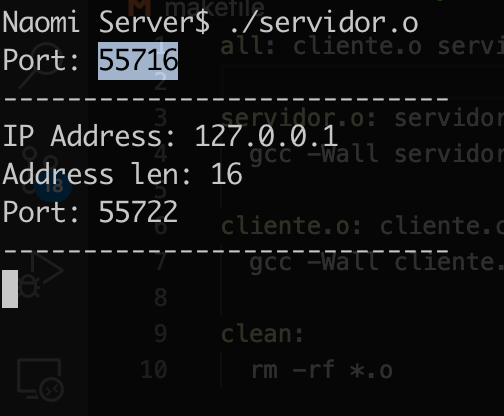
* setar a variável verbose para qualquer valor diferente de 0 e 0 caso queira suprimir os outpus.

# Exercício 1

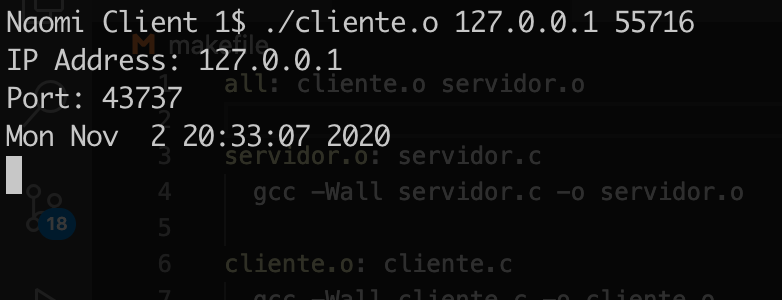
Adicione a função ​sleep no ​servidor.c da atividade prática anterior antes do socket ser fechado close(connfd) de modo que o servidor ''segure'' a conexão do primeiro cliente que se conectar. Com essa modificação, o servidor aceita a conexão de dois clientes de forma concorrente ? Comprove sua resposta através de testes.

O código do servidor foi modificado para que o e “segurasse” a conexão por 1h, através do uso da função sleep antes de de fechar a conexão.

Não, ao iniciar o servidor na port 55716 como mostrado na figura a seguir



E realizar a conexão a partir de um cliente



Não foi possível que outro cliente se conectasse.



# Exercício 2

Escreva, utilizando sockets TCP, um programa cliente e um programa servidor de echo que possibilitem a execução remota de comandos enviados pelo cliente. ​**Lembre-se que o servidor deve atender a vários clientes de forma concorrente.** O servidor deve receber como argumento na linha

de comando a porta na qual irá escutar. O cliente deve receber como argumento na linha de comando o endereço IP do servidor e a porta na qual irá conectar.

**Detalhes do funcionamento:**

* ○  O ​**cliente**​ faz continuamente o seguinte:
  + Estabelece conexão com o servidor
  + Recebe uma cadeia de caracteres do servidor
  + Executa uma cadeia de caracteres
  + Envia o resultado para o servidor
* ○  O ​**servidor**​ faz continuamente o seguinte:
  + Recebe o resultado do cliente
  + Escreve em um arquivo o resultado IP e porta dos clientes

O ​**cliente**​ deverá exibir na saída padrão:

* + dados do host servidor ao qual está se conectando (IP e PORTA)
  + dados de IP e PORTA locais utilizados na conexão

O ​**servidor**​ deverá exibir na saída padrão:

■ Cadeia de caracteres enviadas pelo cliente juntamente com dados de IP e porta do cliente.

**\*\*Devem\*\*** ser escritas e usadas "funções envelopadoras" (wrapper functions) para as chamadas da API de sockets, a fim de tornar o seu código mais limpo e poderem ser reutilizadas nos próximos trabalhos. Utilize a convenção do livro texto, dando o mesmo nome da função, com a 1a letra maiúscula. Veja abaixo um exemplo:

int Socket(int family, int type, int flags) {

int sockfd;

if ((sockfd = socket(family, type, flags)) < 0) {

perror("socket");

exit(1);

} else

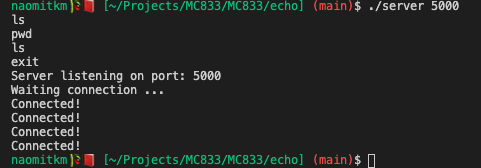
return sockfd;

}

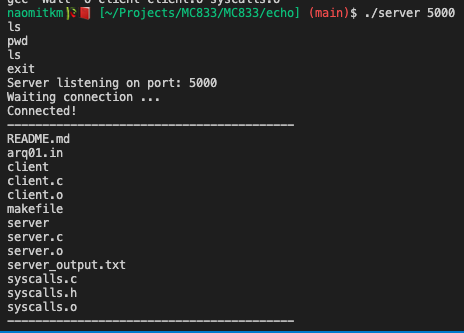
○ Dica: para desenvolver esta atividade poderá ser encontrada nos exemplos do livro-texto da disciplina e nos programas utilizados no exercício anterior.

**Resposta**

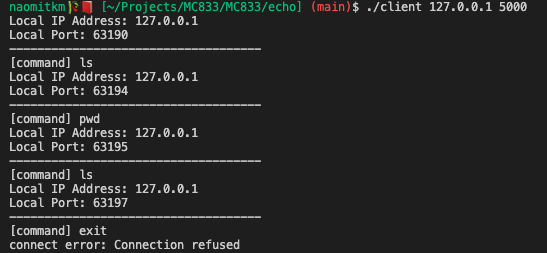
A seguir é mostrada a execução do servidor, que roda em localhost na porta 5000. Na primeira parte é mostrada lista de comandos que o servidor irá mandar para os clientes. Essa lista foi previamente carregada a partir do arquivo arq01.in para facilitar a execução dos testes.



Na figura a seguir, mostra-se um trecho da saída do servidor quando é habilitada a opção verbose, assim mostra-se o resultado do comando ls executado no cliente.



Na figura a seguir, o cliente conecta-se com o servidor da figura acima e recebe os comandos, para facilitar a visualização os comandos recebidos foram mostrados na saída padrão. O cliente executa apenas um comando por conexão, isto é depois de mandar o resultado para o servidor ele fecha a conexão atual e abre uma nova para o próximo comando.



O resultado da execução dos comandos, depois de enviados de volta ao servidor são salvos no arquivo server\_output.txt. O resultado é mostrado a seguir. A porta e o IP do cliente foram mandados pelo próprio cliente. (Neste exemplo foi utilizado apenas um cliente).

Client IP Address: 127.0.0.1

Client Port: 63190

-----------------------------------------

README.md

arq01.in

client

client.c

client.o

makefile

server

server.c

server.o

server\_output.txt

syscalls.c

syscalls.h

syscalls.o

-----------------------------------------

Client IP Address: 127.0.0.1

Client Port: 63194

-----------------------------------------

/Users/naomitkm/Projects/MC833/MC833/echo

-----------------------------------------

Client IP Address: 127.0.0.1

Client Port: 63195

-----------------------------------------

README.md

arq01.in

client

client.c

client.o

makefile

server

server.c

server.o

server\_output.txt

syscalls.c

syscalls.h

syscalls.o

-----------------------------------------

Client IP Address: 127.0.0.1

Client Port: 63197

-----------------------------------------

-----------------------------------------

# Exercício 3

Modifique o servidor para este gravar em um arquivo as informações referentes ao instante em que cada cliente conecta e desconecta, IP, e porta. O servidor não deverá mostrar nenhum mensagem na saída padrão. OBS: Comente o código onde era exibibido mensagens pois fará parte da avaliação.

Para mostrar as mensagens na saída padrão, basta setar a variável “verbose” para um valor diferente de 0 e para desabilitar colocar em 0.

A seguir é reproduzido um trecho da saída do programa (que ocorre no arquivo **connections.txt**)

# Client with 127.0.0.1 port 60216 CONNECTED at: Fri Nov 13 22:51:34 2020

# Client with 127.0.0.1 port 60216 DISCONNECTED at: Fri Nov 13 22:51:34 2020

# Client with 127.0.0.1 port 60207 CONNECTED at: Fri Nov 13 22:51:34 2020

# Client with 127.0.0.1 port 60207 DISCONNECTED at: Fri Nov 13 22:51:36 2020

# Client with 127.0.0.1 port 60239 CONNECTED at: Fri Nov 13 22:51:36 2020

# Client with 127.0.0.1 port 60239 DISCONNECTED at: Fri Nov 13 22:51:38 2020

# Exercício 4

**Detalhes das modificações:**

○  O cliente deve ser modificado de modo que, quando uma certa string for digitada na entrada padrão (por exemplo: ​exit,​ ​quit​, ​bye​, ​sair​, ...), a sua execução seja finalizada (todas as conexões abertas devem ser corretamente fechadas antes).

○  O cliente exibirá, no lugar do "echo" do servidor:

■ cadeias de caracteres enviadas pelo servidor invertidas

○ O servidor exibirá, no lugar da cadeia de caracteres:

■ os dados de IP e PORTA seguidos da string que foi enviada por aquele cliente, de modo a identificar qual comando foi enviado por cada cliente.

■ O IP e PORTA dos clientes que se desconectem, no momento da desconexão.

○ O servidor irá escrever em um arquivo texto o endereço IP, porta, instante de conexão

e de desconexão para cada cliente.

# Exercício 5

Com base ainda no seu código, é correto afirmar que os clientes nunca receberão FIN neste caso já que o servidor sempre ficará escutando (LISTEN)? Justifique.

De maneira geral, o cliente implementado recebe continuamente comandos do servidor, porém ao final de cada comando ele finaliza a conexão e para o próximo comando, uma nova é criada.

Quando o cliente recebe um “exit” do servidor, ele fecha a conexão com o comando close e pois encerra (exit(0)). De acordo com o protocolo de encerramento de conexões TCP, um FIN é enviado ao servidor, escreve-se

write(sockfd, NULL, 0);

Quando o servidor não consegue mais ler do socket relacionado à conexão com este cliente (o número de bytes lidos é 0) o servidor fecha a conexão por seu lado, assim o processo filho que estava relacionado ao cliente também fecha. O fato de o servidor continuar escutando não significa que a conexão que existia não foi fechada, pois ela foi e o cliente recebe o FIN, o que acontece é que ele ainda está apto para novas conexões. Cada uma destas por sua vez é cuidada por um processo filho diferente.

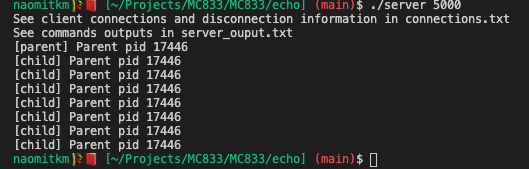
# Exercício 6

Comprove, utilizando ferramentas do sistema operacional, que os processos criados para manipular cada conexão individual do servidor aos clientes são filhos do processo original que foi executado.

Utilizou se o comando getpid() antes do fork para obter o process id do processo original. Com o comando getppid() depois do fork obteve o pid do pai de cada processo gerado após o fork e, como pode-se ver.a seguir, todos os processos filhos possuem a mesma origem.

printf("[parent] Parent pid %u\n", getpid());

printf("[child] Parent pid %u\n", getppid());



**OBS: getpid e getppid** são chamadas de sistema, logo ferramentas do sistema operacional.