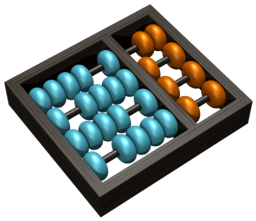

MC833 Relatório 3.2

Servidor TCP Concorrente

Aluno: Fábio Camargo Ricci

RA: 170781

Instituto de Computação

Universidade Estadual de Campinas

Campinas, 03 de Outubro de 2021.

Sumário

1 Questões

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.

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:

   (a) O **cliente** faz continuamente o seguinte:

      i. Estabelece conexão com o servidor
      ii. Recebe uma cadeia de caracteres do servidor
      iii. Executa uma cadeia de caracteres
      iv. Envia o resultado para o servidor

   (b) O **servidor** faz continuamente o seguinte:

      i. Recebe o resultado do cliente
      ii. Escreve em um arquivo o resultado IP e porta dos clientes

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

   (a) Dados do host servidor ao qual está se conectando (IP e PORTA)

(b) Dados de IP e PORTA locais utilizados na conexão

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

(a) 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.

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.

4. **Detalhes das modificações:**

   (a) 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).

   (b) O cliente exibirá, no lugar do "echo"do servidor:

      i. Cadeias de caracteres enviadas pelo servidor invertidas

   (c) O servidor exibirá, no lugar da cadeia de caracteres:

      i. 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.

ii. 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.

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.

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.

2 Respostas

1. Não, o servidor não aceita conexões de clientes concorrentes, e o sleep adicionado ajuda a evidenciar isso. Ao se conectar a um cliente, o fluxo de execução do servidor fica "parado"por 5 segundos, de forma que qualquer outro cliente que tentar se conectar deverá esperar a finalização do fluxo anterior (comando **connect()** apenas é executado a cada 5 segundos).
   Código servidor:

```c
show_client_info(connfd); // log informações cliente

ticks = time(NULL);
snprintf(buf, sizeof(buf), "%.24s\r\n", ctime(&ticks));
write(connfd, buf, strlen(buf));

sleep(5);

close(connfd); // fecha conexão
```

Execução servidor:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o servidor servidor.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./servidor
Listening on 0.0.0.0 port 52828

Client connected on 127.0.0.1 port 52840
Client connected on 127.0.0.1 port 52841
```

Execução cliente 1:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o cliente cliente.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./cliente 0.0.0.0 52828
Connected on 127.0.0.1 port 52840

Mon Oct  4 14:15:38 2021
fabio@Fabios-MacBook-Pro Relatório 3.2 %
```

Execução cliente 2:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./cliente 0.0.0.0 52828
Connected on 127.0.0.1 port 52841

Mon Oct  4 14:15:43 2021
fabio@Fabios-MacBook-Pro Relatório 3.2 %
```

Nota-se que apesar de ambos os clientes terem sido executados ao mesmo momento, há um atraso de 5 segundos entre as saídas (respostas do servidor) por conta do **sleep(5)** adicionado.

2. Primeiramente foram implementadas funções auxiliares e funções

wrapper para todos os métodos principais da comunicação entre sockets para facilitar o desenvolvimento e deixar o código mais organizado.

Servidor:

```
int Socket(sa_family_t family, int type, int flags)
void Bind(int listenfd, sa_family_t family, in_addr_t s_addr, in_port_t port)
void Listen(int listenfd, int backlog)
int Accept(int listenfd, struct sockaddr_in *addr)
void Read(int sockfd, char *buf, size_t size)
void Write(int sockfd, char *buf)
void Getsockname(int sockfd, struct sockaddr_in *addr)
int Fork()
void show_server_info(int sockfd)
void show_client_info_sent(struct sockaddr_in addr, char *content)
void show_client_info_received(struct sockaddr_in addr, char *content)
void save_client_info_received(struct sockaddr_in addr, char *content)
void show_client_connected_info(int sockfd)
void show_client_disconnected_info(int sockfd)
void handle_client(int connfd, struct sockaddr_in addr)
```

Cliente:

```
> int Socket(int family, int type, int flags)…

> void Connect(int sockfd, const char *host, struct sockaddr_in *addr)…

> void Read(int sockfd, char *buf, size_t size)…

> void Write(int sockfd, char *buf)…

> void Getsockname(int sockfd, struct sockaddr_in *addr)…

> void show_server_info(int sockfd, struct sockaddr_in addr)…

> void show_local_info(int sockfd)…

> void execute(char *cmd, char *res)…
```

Além disso, adicionou-se a funcionalidade do servidor aceitar conexões concorrentes de múltiplos clientes. Isso foi possível utilizando-se a função **fork()**, de modo que toda vez que um cliente conecta-se, é lançado um processo separado que irá tratar a comunicação com o mesmo.

Quando ocorre um **fork()**, a memória do processo pai é duplicada para o filho, então é necessário tratar o fechamento dos sockets em ambos os casos. Caso a variável **child_pid == 0**, trata-se do fluxo de execução do processo filho, o qual não se interessa pelo socket **listenfd**, então o mesmo pode ser fechado. O mesmo vale no fluxo de execução do processo pai, que não se interessa por **connfd** pois é apenas um processo passivo que escuta conexões no socket **listenfd**.

Código:

```
for (;;) // loop aceitando conexões que cheguem
{
   connfd = Accept(listenfd, &client_addr); // a

   child_pid = Fork(); // fork process to handle
   if (child_pid == 0)
   {
      close(listenfd); // fecha o listen socket
      handle_client(connfd, client_addr);
      break;
   }
   else
   {
      close(connfd); // fecha o connection socke
   }
}
```

Dessa forma cada cliente que se conecta ao servidor recebe continuamente o comando "pwd", o executa localmente e retorna o resultado ao servidor, o qual salva em um arquivo e imprime na saída padrãoa string recebida.

Servidor:

```
fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o servidor servidor.c -Wall
fabio@Fabios-MacBook-Pro Relatório 3.2 % ./servidor
Listening on 0.0.0.0 port 61387

Sent: 'pwd' to client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Sent: 'pwd' to client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Sent: 'pwd' to client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Sent: 'pwd' to client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Sent: 'pwd' to client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Sent: 'pwd' to client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
```

Arquivo servidor:

```
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61204
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61271
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61296
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61379
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
Received: '/Users/fabio/workspace/unicamp/10 semestre/mc833/Relatório 3.2
' from client connected on IP: 127.0.0.1 Port: 61388
```

Cliente:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o cliente cliente.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./cliente 0.0.0.0 61387
Connected on server IP: 0.0.0.0 Port: 61387
Connected on local IP: 127.0.0.1 Port 61388
```

3. Foram adicionadas funções para tratar o início e fim das conexões com os clientes. Nelas, são escritos o endereço IP, porta e horário em um arquivo texto.

   Além disso, a função Read() (wrapper para read()) foi modificada para retornar o valor de read(), pois um retorno 0 significa um fim de conexão do cliente com sucesso. Assim, colocou-se um break no loop da função handle_client() para essa condição, finalizando o lado do servidor também.

   Servidor:

```c
void handle_client(int connfd, struct sockaddr_in addr)
{
    char buffer[MAXDATASIZE];
    int read_res;
    show_client_connected_info(connfd, addr);

    for (;;)
    {

        snprintf(buffer, sizeof(buffer), "pwd"); // print em um b

        Write(connfd, buffer); // write() wrapper
        // show_client_info_sent(addr, buffer); // show client in

        read_res = Read(connfd, buffer, MAXDATASIZE); // read() w
        if (read_res == 0)                            // 0 signif
        {
            break;
        }

        // show_client_info_received(addr, buffer); // show clien
        save_client_info_received(addr, buffer); // save client i

        sleep(5);
    }

    show_client_disconnected_info(connfd, addr);

    shutdown(connfd, SHUT_RDWR);

    close(connfd); // fecha conexão
}
```

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o servidor servidor.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./servidor
Listening on 0.0.0.0 port 62990
```

Cliente:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o cliente cliente.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./cliente 0.0.0.0 62990
Connected on server IP: 0.0.0.0 Port: 62990
Connected on local IP: 127.0.0.1 Port 63100
^C
fabio@Fabios-MacBook-Pro Relatório 3.2 %
```

4. Modificou-se a função **handle_client()** para enviar a string "exit"após receber o retorno do comando executado pelo cliente para indicar o fim da conexão ao mesmo.

   No cliente, adicionou-se uma verificação pela string "exit"recebida do servidor e pelo **stdin**. Para tratar tanto a comunicação com o servidor como escutar inputs do usuário, foi feito um **fork()** no cliente para um processo que que apenas escuta o stdin e outro que trata a comunicação com o socket, sendo possível receber o comando de término de conexão de ambos.

   Dessa forma, quando o cliente recebe "exit"do servidor ou pelo stdin, o fluxo de execução é interrompido e o programa termina.

   Servidor:

```
{
    char buffer[MAXDATASIZE];

    show_client_connected_info(connfd, addr);

    for (;;)
    {
        snprintf(buffer, sizeof(buffer), "pwd"); // prin
        Write(connfd, buffer);                   // writ

        // show_client_info_sent(addr, buffer); // show

        if (Read(connfd, buffer, MAXDATASIZE) == 0) // 0
        {
            break;
        }

        // show_client_info_received(addr, buffer); // s
        save_client_info_received(addr, buffer); // save

        sleep(5);

        snprintf(buffer, sizeof(buffer), "exit"); // pri
        Write(connfd, buffer);                    // wri
    }

    show_client_disconnected_info(connfd, addr);

    shutdown(connfd, SHUT_RDWR);

    close(connfd); // fecha conexão
}
```

Cliente:

```c
pid_t child_pid = Fork();

if (child_pid == 0) // child
{
    for (;;)
    {
        Read(STDIN_FILENO, buffer, BUFFER_SIZE); //

        if (strcmp(buffer, "exit\n") == 0) // chec
        {
            exit(0);
        }
    }
}
else // parent
{
    for (;;)
    {
        Read(sockfd, buffer, BUFFER_SIZE); // read

        if (strcmp(buffer, "exit") == 0) // check
        {
            exit(0);
        }

        execute(buffer, buffer); // executa o coma

        Write(sockfd, buffer); // write() wrapper
    }
    waitpid(child_pid, NULL, 0); // wait for chil
}
```

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o cliente cliente.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./cliente 0.0.0.0 56309
Connected on server IP: 0.0.0.0 Port: 56309
Connected on local IP: 127.0.0.1 Port 56316
exit
fabio@Fabios-MacBook-Pro Relatório 3.2 %
```

5. Sim, é correto. Como o cliente recebe "exit"do servidor ou do stdin, é ele que finaliza a conexão, de forma que, mesmo que o servidor envie "FIN", o cliente não receberá pois já haverá terminado sua execução.

6. Servidor:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % gcc -o servidor servidor.c -Wall
[fabio@Fabios-MacBook-Pro Relatório 3.2 % ./servidor
Listening on 0.0.0.0 port 62509
```

Nota-se que o servidor está ouvindo na porta 62509, logo precisamos descobrir seu process ID:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 %  sudo lsof -i -P -n | grep LISTEN | grep 62509
[Password:
servidor  90415          fabio    3u  IPv4 0x31c41137beb366b7      0t0    TCP *:62509 (LISTEN)
```

Observa-se que o process ID obtido foi 90415, com isso, executa-se o comando **pstree** para listar todos os processos filhos:

```
[fabio@Fabios-MacBook-Pro Relatório 3.2 % pstree 90415
--= 90415 fabio ./servidor
[fabio@Fabios-MacBook-Pro Relatório 3.2 % pstree 90415
-+= 90415 fabio ./servidor
  \--- 09454 fabio ./servidor
[fabio@Fabios-MacBook-Pro Relatório 3.2 % pstree 90415
-+= 90415 fabio ./servidor
  |--- 09454 fabio ./servidor
  \--- 11198 fabio ./servidor
fabio@Fabios-MacBook-Pro Relatório 3.2 %
```

Na primeira vez, o servidor está executando sem nenhum cliente, logo é listado apenas o processo pai. Quando é executado um cliente, é feito um **fork()** e é listado um processo filho com ID 09454.

Finalmente, quando temos dois clientes concorrentes rodando, ambos são listados como filhos do processo pai.