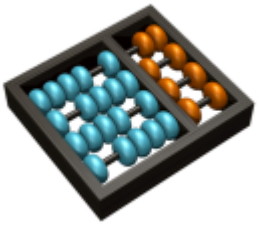


Instituto de Computação
Universidade Estadual de Campinas - Unicamp

Exercício 3

Alunos: Carlos Eduardo da Silva Santos e Felipe Correia Labbate
RA: 195396 e 196699

Campinas, 11 de Outubro de 2022

Sumário

1. Inclua no diagrama de sequência das chamadas de funções identificadas entre os sockets cliente e servidor (pode ser o diagrama do exercício anterior), a parte relacionada à concorrência do servidor.

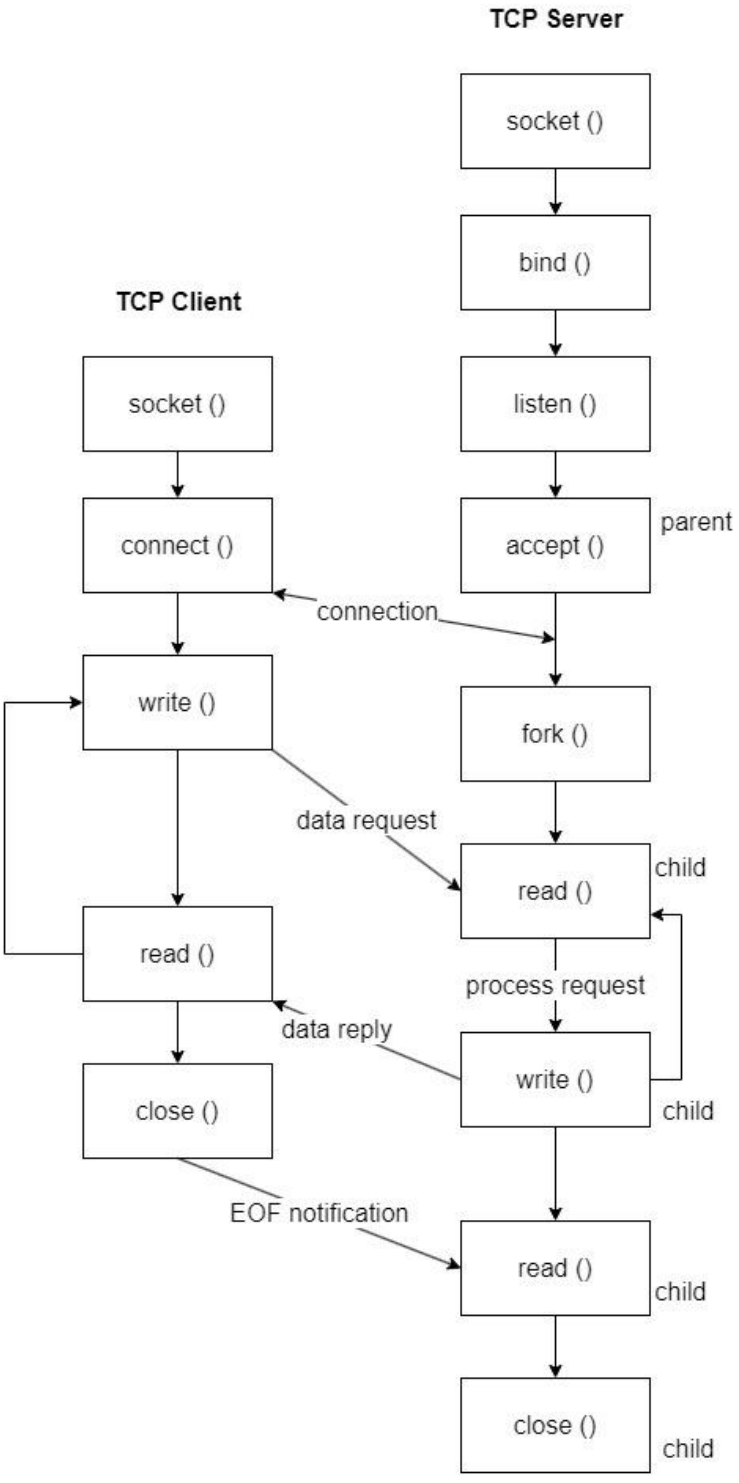


Diagrama exemplo de comportamento entre um cliente e um servidor concorrente

2. 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.

Não, o servidor não aceita a conexão do segundo cliente. O servidor só aceita a conexão do segundo cliente após a conexão do primeiro cliente ser encerrada.

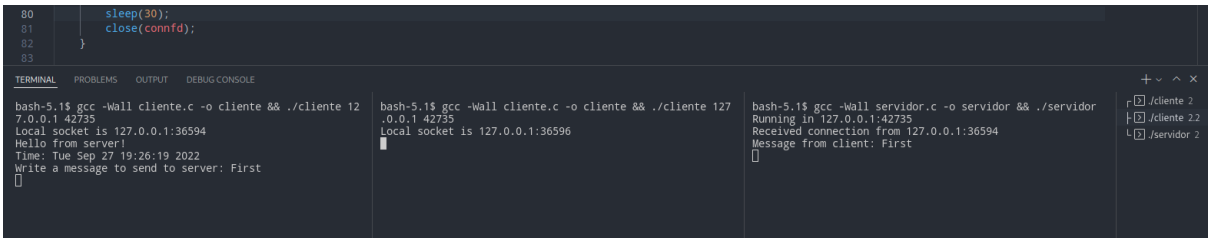


Imagem 1: primeiro cliente consegue enviar a mensagem, porém a opção não aparece para o segundo cliente

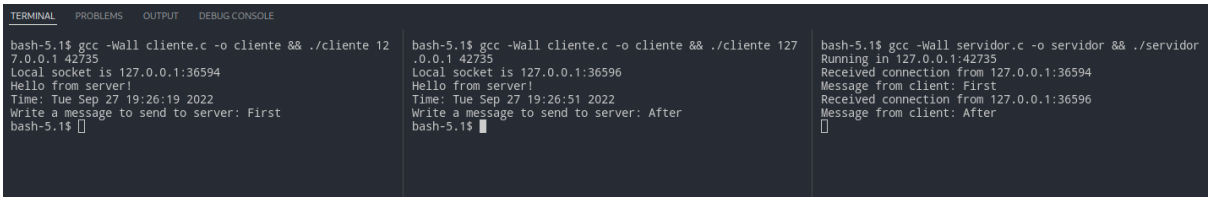


Imagem 2: logo após o primeiro cliente encerrar a conexão, o segundo cliente consegue enviar a mensagem ao servidor

Coluna da esquerda: primeiro cliente
Coluna do meio: segundo cliente
Coluna da direita: servidor

3. Fazendo uso de sockets TCP, escreva um programa cliente-servidor onde o servidor seja concorrente (pode atender a vários clientes de forma concorrente).

O servidor recebe como argumento a porta de escuta. A IP do servidor pode ser setada como a IP local da máquina.

```
30      if (argc != 2)
31      {
32          strcpy(error, "uso: ");
33          strcat(error, argv[0]);
34          strcat(error, " <Port>");
35          perror(error);
36          exit(1);
37      }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
$ gcc -Wall servidor.c -o servidor && ./servidor 1024
```

O cliente deve ser informado sobre a IP e porta nas quais escuta o servidor através de argumentos na entrada do programa.

```
36
37      if (argc != 3) {
38          strcpy(error,"uso: ");
39          strcat(error,argv[0]);
40          strcat(error," <IPaddress> <Port>");
41          perror(error);
42          exit(1);
43      }
44
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
$ gcc -Wall cliente.c -o cliente && ./cliente 127.0.0.1 1024
```

O servidor printa na saída padrão informações da conexão de um cliente. Ou seja, printa IP, porta do cliente e hora de conexão ou desconexão.

```
    servaddr_len = sizeof(servaddr);
    if (getsockname(listenfd, (struct sockaddr *)&servaddr, &servaddr_len) < 0)
    {
        perror("getsockname error");
        exit(1);
    }

    // printf("Running in %s:%d\n", inet_ntoa(servaddr.sin_addr), ntohs(servaddr.sin_port));
```

```
// ticks = time(NULL);
// printf("Closed connection from %s:%d at %.24s\r\n", inet_ntoa(servaddr.sin_addr), ntohs(servaddr.sin_port), ctime(&ticks));
close(connfd);
exit(0);
```

O servidor envia uma cadeia de comandos

```
char commands[COMMANDS_SIZE][MAXDATASIZE] = {"ls -l", "ifconfig", "pwd", "EXIT"};
```

```
for (int i = 0; i < COMMANDS_SIZE; i++)
{
    snprintf(buf, sizeof(commands[i]), "%s", commands[i]);
    write(connfd, buf, strlen(buf));
    bzero(buf, MAXDATASIZE);
    read(connfd, buf, MAXLINE);

    fprintf(received_file, "%s\n%s\n", commands[i], buf);
}
```

O servidor recebe o resultado do cliente correspondente a execução dos comandos e escreve em um arquivo junto com as informações do cliente

output.txt

```
src > exercise2-2 > output.txt
1   Received connection from 127.0.0.1:51924 at Tue Oct 11 22:31:04 2022
2
3   ls -l
4   total 48
5   -rwxrwxr-x 1 flabbate flabbate 18104 out 11 22:29 cliente
6   -rw-rw-r-- 1 flabbate flabbate  3552 out 11 22:30 cliente.c
7   -rw-rw-r-- 1 flabbate flabbate     0 out 11 22:31 output.txt
8   -rwxrwxr-x 1 flabbate flabbate 17720 out 11 22:29 servidor
9   -rw-rw-r-- 1 flabbate flabbate  3664 out 11 21:03 servidor.c
10
11  ifconfig
12  br-4012902152d3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
13          inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
14          ether 02:42:8b:70:ab:87  txqueuelen 0  (Ethernet)
15          RX packets 0  bytes 0 (0.0 B)
16          RX errors 0  dropped 0  overruns 0  frame 0
17          TX packets 0  bytes 0 (0.0 B)
18          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS  JUPYTER

flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programming/src/exercise2-2$ ./servidor 1024

flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programming/src/exercise2-2$ ./cliente 127.0.0.1 1024
Local socket is 127.0.0.1:51924
L- SL
GIFNOCFI
DWP
TIXE
^C
flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programming/src/exercise2-2$
```

O cliente estabelece conexão com servidor e printa informações de conexão

```
Connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

GetSockName(sockfd, (struct sockaddr *)&servaddr);

printf("Remote socket is %s:%s\n", argv[1], argv[2]);

PrintLocalInfo(&servaddr.sin_addr, &servaddr.sin_port);
```

```
void Connect(int sockfd, struct sockaddr *servaddr, int servaddr_len)
{
    if (connect(sockfd, servaddr, servaddr_len) < 0)
    {
        perror("connect error");
        exit(1);
    }
}

void PrintLocalInfo(struct in_addr *ip, in_port_t *port)
{
    printf("Local socket is %s:%d\n", inet_ntoa(*ip), ntohs(*port));
}
```

Recebe, executa e envia comandos para o servidor

```
while (strncmp(recvline, "EXIT", 4) != 0)
{
    FILE *fp;
    char path[1035];
    char file_content[10000] = "";

    fp = popen(recvline, "r");
    if (fp == NULL)
    {
        printf("Failed to run command\n");
        exit(1);
    }

    file_size = 0;
    while (fgets(path, sizeof(path), fp) != NULL)
    {
        strcat(file_content, path);
    }

    file_size = strlen(file_content) + 1;

    send(sockfd, file_content, file_size, 0);

    pclose(fp);

    write(sockfd, message, strlen(message));

    bzero(recvline, MAXLINE + 1);
    n = read(sockfd, recvline, MAXLINE);
    strcpy(aux, recvline);
    printf("%s\n", strupr(strrev(aux)));
}
```

4. 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 nenhuma mensagem na saída padrão. OBS: Comente o código onde era exibido mensagens pois fará parte da avaliação.

Os códigos de saída padrão foram comentados

```
// get hello
// if (fputs(recvline, stdout) == EOF)
// {
//     perror("fputs error");
//     exit(1);
// }
// bzero(recvline, sizeof(recvline));
```

cliente

```
// printf("Running in %s:%d\n", inet_ntoa(servaddr.sin_addr), ntohs(servaddr.sin_port));
```

```
// ticks = time(NULL);
// printf("Received connection from %s:%d at %.24s\r\n", inet_ntoa(servaddr.sin_addr), ntohs(servaddr.sin_port), ctime(&ticks));

// send
// ticks = time(NULL);
// snprintf(buf, sizeof(buf), "Hello from server!\nTime: %.24s\r\n", ctime(&ticks));
// write(connfd, buf, strlen(buf));

// if(recv(connfd, message, sizeof(message), 0) < 0)
// {
//     perror("recv error");
//     exit(1);
// }
// printf("Message from client: %s\n", message);
```

```
// ticks = time(NULL);
// printf("Closed connection from %s:%d at %.24s\r\n", inet_ntoa(servaddr.sin_addr), ntohs(servaddr.sin_port), ctime(&ticks));
```

servidor

Arquivo gerado pelo servidor

output.txt

src > exercise2-2 > output.txt

```
Received connection from 127.0.0.1:48686 at Tue Oct 11 21:58:53 2022

ls -l
total 48
-rwxrwxr-x 1 flabbate flabbate 18104 out 11 21:58 cliente
-rw-rw-r-- 1 flabbate flabbate  3569 out 11 21:58 cliente.c
-rw-rw-r-- 1 flabbate flabbate     0 out 11 21:58 output.txt
-rwxrwxr-x 1 flabbate flabbate 17720 out 11 21:58 servidor
-rw-rw-r-- 1 flabbate flabbate  3664 out 11 21:03 servidor.c

ifconfig
br-4012902152d3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:8b:70:ab:87  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:94:ba:3f:b8  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eno2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 2c:ea:7f:e6:14:e2  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

5. Modifique cliente e servidor para que…

O cliente imprime na saída padrão a cadeia de comandos invertido e em maiúsculo

```
flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programmin
g/src/exercise2-2$ ./cliente 127.0.0.1 1024
Local socket is 127.0.0.1:51924
L- SL
GIFNOCFI
DWP
TIXE
```

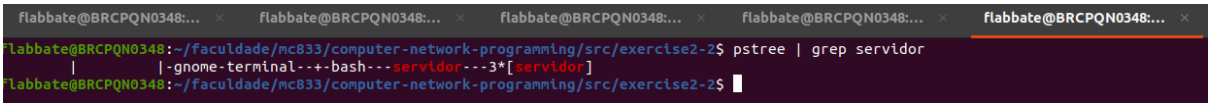
Saída do servidor contendo informações necessárias

```
Received connection from 127.0.0.1:48686 at Tue Oct 11 21:58:53 2022

ls -l
total 48
-rwxrwxr-x 1 flabbate flabbate 18104 out 11 21:58 cliente
-rw-rw-r-- 1 flabbate flabbate  3569 out 11 21:58 cliente.c
-rw-rw-r-- 1 flabbate flabbate     0 out 11 21:58 output.txt
-rwxrwxr-x 1 flabbate flabbate 17720 out 11 21:58 servidor
-rw-rw-r-- 1 flabbate flabbate  3664 out 11 21:03 servidor.c

ifconfig
br-4012902152d3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
```

...

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. Faça testes e inclua sleeps se achar necessário para provar a concorrência do servidor.

Resposta: Como é possível ver na imagem abaixo, a saída do comando *pstree*, com três clientes conectados ao mesmo tempo no mesmo servidor, indica a existência de três processos filhos do servidor em execução.



Saída do comando “pstree | grep servidor”

Abaixo temos as saídas do comando “ps ax”, indicando a existência de três processos do cliente em execução simultânea, e quatro do servidor.

```
flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programming/src/exercise2-2$ ps ax | grep cliente
 144354 pts/1    S+     0:00 ./cliente 127.0.0.1 1024
 144362 pts/2    S+     0:00 ./cliente 127.0.0.1 1024
 144369 pts/4    S+     0:00 ./cliente 127.0.0.1 1024
```

```
flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programming/src/exercise2-2$ ps ax | grep servidor
 144344 pts/0    S+     0:00 ./servidor 1024
 144355 pts/0    S+     0:00 ./servidor 1024
 144363 pts/0    S+     0:00 ./servidor 1024
 144370 pts/0    S+     0:00 ./servidor 1024
```

Saídas do comando “ps ax”, com filtros para mostrar os clientes e o servidor

Ao executar o comando “ps lax”, é possível ver na coluna PPID o valor PID do processo pai.

```
flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programming/src/exercise2-2$ ps lax | { head -1; grep servidor; }
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
0  1001  145099   53460  20   0   2500   648 inet_c S+   pts/0      0:00 ./servidor 1025
1  1001  145103  145099  20   0   2500   104 wait_w S+   pts/0      0:00 ./servidor 1025
1  1001  145110  145099  20   0   2500   104 wait_w S+   pts/0      0:00 ./servidor 1025
1  1001  145125  145099  20   0   2500   104 wait_w S+   pts/0      0:00 ./servidor 1025
flabbate@BRCPQN0348:~/faculdade/mc833/computer-network-programming/src/exercise2-2$
```

Quatro processos do servidor em execução, sendo que três deles têm como o processo pai o processo PPID = 145099