

Relatório - Servidor TCP Concorrente

Alunos: Rian Radeck e Igor Brito

RAs: 187793 e 171929

Instituto de Computação
Universidade Estadual de Campinas

Campinas, 05 de Setembro de 2023.

Sumário

1	Modificações no Cliente	2
2	Modificações no Servidor	4
3	Execução	6

1 Modificações no Cliente

- Modificamos todas as funções de manipulação dos sockets para utilizar os encapsuladores, que foram colocados em `netutils.h`
- Agora o cliente recebe dados do servidor uma linha por vez, dessa forma, ele sabe quando parar o `read()` para poder fazer suas simulações, uma vez que estamos utilizando chamadas blocking e apenas uma thread de execução no cliente.

```
1      while ( (n = read(sockfd, recvline + recvline_offset, MAXLINE -  
2      recvline_offset)) > 0) {  
3          recvline_offset += n;  
4          if (recvline[recvline_offset - 1] == '\n')  
5              break;  
6      }  
7  
8      if (n < 0) {  
9          perror("read error");  
10         exit(1);  
11     }  
12  
13     recvline[recvline_offset] = 0;  
14  
15     puts("----- RECIEVED FROM SERVER -----");  
16     if (fputs(recvline, stdout) == EOF) {  
17         perror("fputs error");  
18         exit(1);  
19     }  
20     puts("-----");
```

- Após o cliente receber a instrução do servidor, ele simula sua execu-

ção e retorna para o servidor logo em seguida.

```
1      if (strcmp(recvline, "SIMULE: CPU_INTENSIVA\n") == 0)
2      {
3          puts("Starting CPU stress");
4          sleep(5);
5          char response[] = "SIMULACAO: CPU_INTENSIVA CONCLUIDA\n";
6          write(sockfd, response, strlen(response));
7      }
8      else if (strcmp(recvline, "SIMULE: MEMORIA_INTENSIVA\n") == 0)
9      {
10         puts("Starting MEM stress");
11         sleep(5);
12         char response[] = "SIMULACAO: MEMORIA_INTENSIVA
13         CONCLUIDA\n";
14         write(sockfd, response, strlen(response));
15     } else if (strcmp(recvline, "DC\n") == 0){
16         char response[] = "DISCONNECTED\n";
17         write(sockfd, response, strlen(response));
18         break;
19     } else {
20         char response[] = "OPERACAO INVALIDA\n";
21         write(sockfd, response, strlen(response));
22     }
23     recvline_offset = 0;
```

- Finalmente o cliente repete os últimos dois itens para processar múltiplas requisições do servidor.

2 Modificações no Servidor

- No servidor, assim como no cliente, alteramos todas as chamadas de funções manipuladoras de socket para utilizar a versão encapsulada definida em `netutils.h`
- Modificamos também a saída do programa, chamando a função `Log()`, fazendo que nossas informações sejam guardadas em um arquivo texto.
- Mais uma mudança requerida foi o parâmetro porta na execução do comando.
- Agora, para cada cliente instanciamos um novo processo através da função `fork()`, e continuamos a processar pedidos de outros clientes no processo pai. Podemos diferenciar o processo pai do filho através do parâmetro de retorno do `fork()`.

```
1  for ( ; ; ) {  
2      connfd = Accept(listenfd, (struct sockaddr *) NULL, NULL);  
3  
4      if ((connpid = fork()) == 0)  
5      {  
6          close(listenfd);
```

- No processo filho mandamos uma mensagem de bem vindo para o cliente e emitimos um LOG que uma nova conexão foi iniciada

```
1      ticks = time(NULL);  
2      snprintf(buf, sizeof(buf), "Hello from server!\nTime: %.24s\r\nPID: %d\n", ctime(&ticks), getpid());  
3      write(connfd, buf, strlen(buf));  
4      int k;
```

```

5         while((k = read(connfd, buf, MAXDATASIZE)) > 0)
6             if (buf[k - 1] == '\n')
7                 break;
8         Log("SENT A HELLO AND RECIEVED ANY RESPONSE,
    STARTING COMMUNICATION");

```

- Agora simulamos 4 queries para esse cliente, escolhendo aleatoriamente entre `SIMULE: MEMORIA_INTENSIVA` e `SIMULE: CPU_INTENSIVA` e mandamos esse pedido para o cliente. Após a finalização das 4 queries o processo filho envia uma requisição de desconexão, fazendo o cliente se desconectar e terminando a execução do `fork()`.

```

1         for(int cnt = 5;cnt--;)
2         {
3             char cbuf[MAXDATASIZE + 1];
4             srand(time(NULL));
5             char cmd[30];
6             if (rand() % 2)
7                 strcpy(cmd, "SIMULE: MEMORIA_INTENSIVA\n");
8             else
9                 strcpy(cmd, "SIMULE: CPU_INTENSIVA\n");
10            if (cnt == 0)
11                strcpy(cmd, "DC\n");
12
13            snprintf(cbuf, sizeof(cbuf), "%s", cmd);
14            write(connfd, cbuf, strlen(cbuf));
15
16            snprintf(logbuf, sizeof(logbuf), "Sending -> %s", cmd);
17            Log(logbuf);

```

- Para cada uma das queries recebemos uma resposta do cliente. Da mesma maneira que no cliente, paramos assim que recebermos um

\n para ler somente uma linha do cliente e saber quando devemos parar.

```
1      int n;  
2      while((n = read(connfd, cbuf, MAXDATASIZE)) > 0)  
3      {  
4          cbuf[n] = 0;  
  
5  
6          snprintf(logbuf, sizeof(logbuf), "Recieved -> %s", cbuf);  
7          Log(logbuf);  
8  
9          if (cbuf[n - 1] == '\n')  
10             break;  
11     }
```

3 Execução

Abordados nossas principais mudanças em relação ao servidor não concorrente vamos executar nosso programa.

Comandos de compilação:

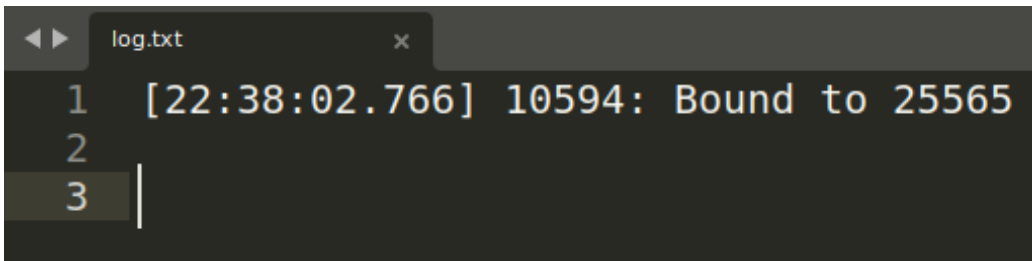
- # gcc servidor.c netutils.c -Wall -o servidor
- # gcc cliente.c netutils.c -Wall -o cliente

Vamos executar uma situação onde apenas um cliente se comunica com o servidor para entendermos o funcionamento básico de nosso programa. Para executar o servidor utilizamos o comando # ./servidor 25565. Sendo assim abrimos o servidor na porta 25565.

```
rianc@rianc: gcc servidor.c netutils.c -Wall -o servidor
rianc@rianc: ./servidor 25565
```

Figura 1: Execução inicial do servidor

Observe que o terminal de execução não vai mudar, já que estamos escrevendo todas as comunicações do servidor com o usuário no arquivo log.txt, sendo assim vamos dar uma olhada nele



```
log.txt
1 [22:38:02.766] 10594: Bound to 25565
2
3
```

Figura 2: Log inicial do servidor

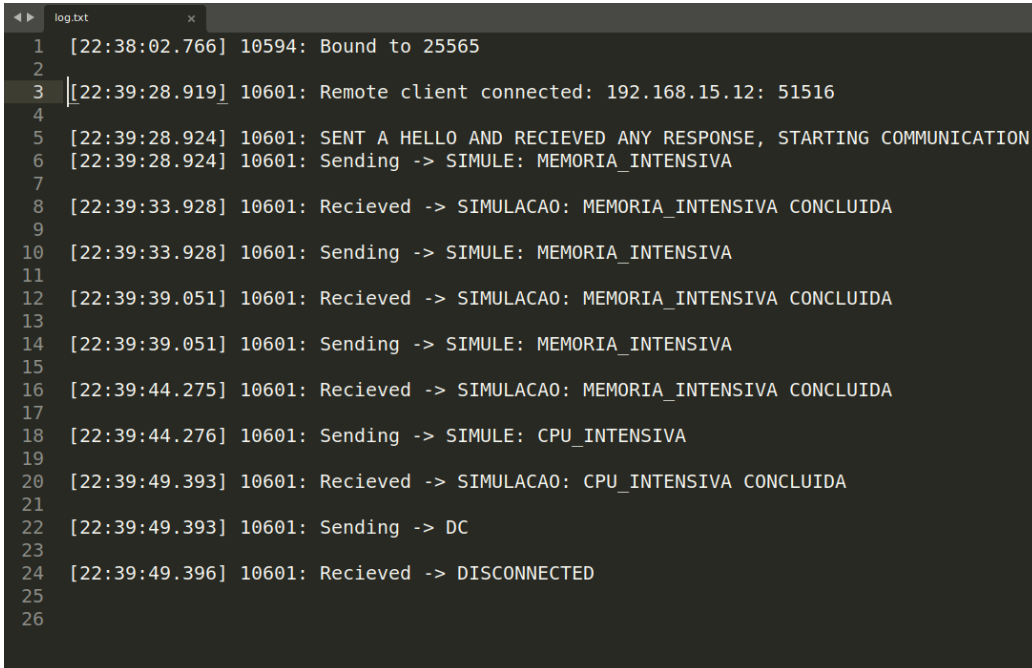
Agora é um bom momento para explicarmos a estrutura de nosso logger, ele é composto por duas informações primordiais em cada entrada, o horário e o ID do processo que executou aquele log, e a mensagem logada em seguida. Sendo assim, nosso log mostra que o processo pai de nosso servidor tem ID igual a 10594.

Decidimos fazer a comunicação entre dois computadores na rede local, portanto precisamos encontrar o IP local da máquina onde o servidor está rodando. Isso se torna fácil executando o comando # `ifconfig`.

```
wlp0s20f3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.15.7 netmask 255.255.255.0 broadcast 192.168.15.255
```

Figura 3: Informações da máquina do servidor na rede local

Quando executado o comando # `./cliente 192.168.15.7 25565` em outra máquina observamos o seguinte resultado no log do nosso servidor



```
log.txt
1 [22:38:02.766] 10594: Bound to 25565
2
3 [22:39:28.919] 10601: Remote client connected: 192.168.15.12: 51516
4
5 [22:39:28.924] 10601: SENT A HELLO AND RECIEVED ANY RESPONSE, STARTING COMMUNICATION
6 [22:39:28.924] 10601: Sending -> SIMULE: MEMORIA_INTENSIVA
7
8 [22:39:33.928] 10601: Recieved -> SIMULACAO: MEMORIA_INTENSIVA CONCLUIDA
9
10 [22:39:33.928] 10601: Sending -> SIMULE: MEMORIA_INTENSIVA
11
12 [22:39:39.051] 10601: Recieved -> SIMULACAO: MEMORIA_INTENSIVA CONCLUIDA
13
14 [22:39:39.051] 10601: Sending -> SIMULE: MEMORIA_INTENSIVA
15
16 [22:39:44.275] 10601: Recieved -> SIMULACAO: MEMORIA_INTENSIVA CONCLUIDA
17
18 [22:39:44.276] 10601: Sending -> SIMULE: CPU_INTENSIVA
19
20 [22:39:49.393] 10601: Recieved -> SIMULACAO: CPU_INTENSIVA CONCLUIDA
21
22 [22:39:49.393] 10601: Sending -> DC
23
24 [22:39:49.396] 10601: Recieved -> DISCONNECTED
25
26
```

Figura 4: Log do servidor após a conexão do cliente

Observe que estamos logando tudo que enviamos para o cliente e tudo que recebemos do cliente.

Outro grande ponto aqui é a ação do `fork()` após o processo pai aceitar a conexão. Veja que toda a comunicação durante todo o processo é lidado por um processo filho de ID 10601, enquanto o processo pai volta novamente para o estado passivo, escutando novas conexões.

Podemos garantir que o processo que está fazendo a comunicação é o de ID 10601 também olhando para a saída padrão de nosso cliente, pois em nossa mensagem de boas vindas enviamos também qual processo está lidando com conexão. Observe a saída padrão do cliente

```

igorbrito@: ./cliente 192.168.15.7 25565
Local: 172.25.139.129 44814
Remote: 192.168.15.7 25565
----- RECIEVED FROM SERVER -----
Hello from server!
Time: Tue Sep 19 22:39:28 2023
PID: 10601
-----
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
SIMULE: CPU_INTENSIVA
-----
Starting CPU stress
----- RECIEVED FROM SERVER -----
DC
-----
igorbrito@: |

```

Figura 5: Saída padrão do cliente

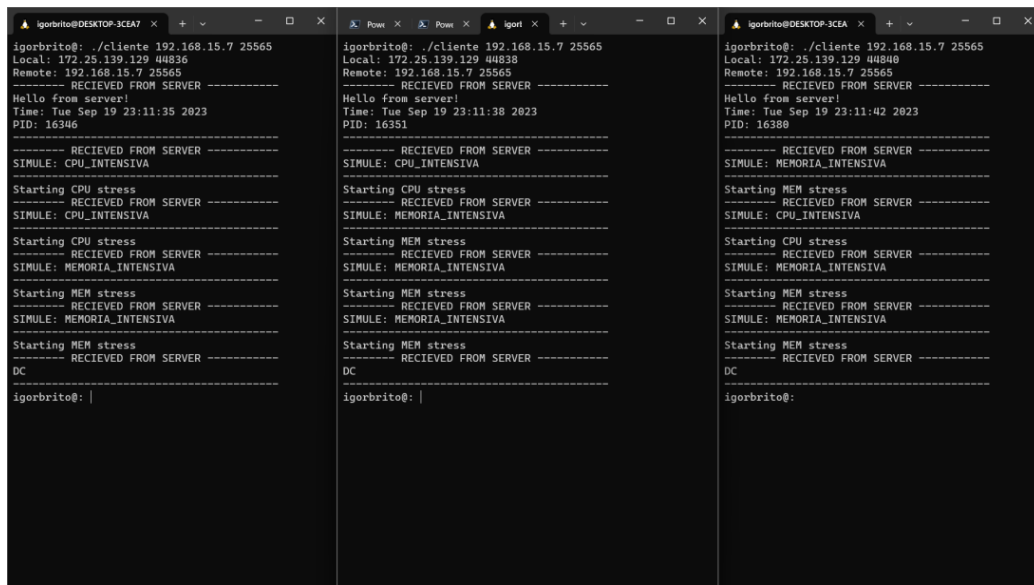
Veja que nosso cliente recebe a mensagem de boas vindas, as 4 requisições e um pedido de desconexão, quando ele recebe esse último pedido ele fecha a conexão e termina sua execução.

Agora vamos fazer a execução concorrente de três processos, para isso só precisamos executar o código do cliente em três terminais diferentes. Observe o que acontece com o log do servidor quando isso acontece

```
log.txt x
1 [23:09:06.930] 16299: Bound to 25565
2
3 [23:11:35.081] 16346: Remote client connected: 192.168.15.12: 51438
4
5 [23:11:35.084] 16346: SENT A HELLO AND RECIEVED ANY RESPONSE, STARTING COMMUNICATION
6 [23:11:35.084] 16346: Sending -> SIMULE: CPU_INTENSIVA
7
8 [23:11:38.356] 16351: Remote client connected: 192.168.15.12: 51440
9
10 [23:11:38.359] 16351: SENT A HELLO AND RECIEVED ANY RESPONSE, STARTING COMMUNICATION
11 [23:11:38.359] 16351: Sending -> SIMULE: CPU_INTENSIVA
12
13 [23:11:40.200] 16346: Recieved -> SIMULACAO: CPU_INTENSIVA CONCLUIDA
14
15 [23:11:40.200] 16346: Sending -> SIMULE: CPU_INTENSIVA
16
17 [23:11:42.454] 16380: Remote client connected: 192.168.15.12: 51442
18
19 [23:11:42.457] 16380: SENT A HELLO AND RECIEVED ANY RESPONSE, STARTING COMMUNICATION
20 [23:11:42.457] 16380: Sending -> SIMULE: MEMORIA_INTENSIVA
21
22 [23:11:43.474] 16351: Recieved -> SIMULACAO: CPU_INTENSIVA CONCLUIDA
23
24 [23:11:43.475] 16351: Sending -> SIMULE: MEMORIA_INTENSIVA
25
26 [23:11:45.319] 16346: Recieved -> SIMULACAO: CPU_INTENSIVA CONCLUIDA
27
28 [23:11:45.320] 16346: Sending -> SIMULE: MEMORIA_INTENSIVA
29
30 [23:11:47.570] 16380: Recieved -> SIMULACAO: MEMORIA_INTENSIVA CONCLUIDA
31
32 [23:11:47.570] 16380: Sending -> SIMULE: CPU_INTENSIVA
33
34 [23:11:48.478] 16351: Recieved -> SIMULACAO: MEMORIA_INTENSIVA CONCLUIDA
35
36 [23:11:48.478] 16351: Sending -> SIMULE: MEMORIA_INTENSIVA
37
38 [23:11:50.437] 16346: Recieved -> SIMULACAO: MEMORIA_INTENSIVA CONCLUIDA
39
40 [23:11:50.437] 16346: Sending -> SIMULE: MEMORIA_INTENSIVA
41
42 [23:11:52.694] 16380: Recieved -> SIMULACAO: CPU_INTENSIVA CONCLUIDA
2 lines, 1 character selected
```

Figura 6: Log com três clientes ao mesmo tempo

Olhe também as saídas dos programas cliente



```
igorbrito@DESKTOP-3CEA7 x + v - □ X
igorbrito@: ./cliente 192.168.15.7 25565
Local: 172.25.139.129 44836
Remote: 192.168.15.7 25565
----- RECIEVED FROM SERVER -----
Hello from server!
Time: Tue Sep 19 23:11:35 2023
PID: 16346
-----
----- RECIEVED FROM SERVER -----
SIMULE: CPU_INTENSIVA
-----
Starting CPU stress
----- RECIEVED FROM SERVER -----
SIMULE: CPU_INTENSIVA
-----
Starting CPU stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
DC
igorbrito@: |

igorbrito@: ./cliente 192.168.15.7 25565
Local: 172.25.139.129 44838
Remote: 192.168.15.7 25565
----- RECIEVED FROM SERVER -----
Hello from server!
Time: Tue Sep 19 23:11:38 2023
PID: 16351
-----
----- RECIEVED FROM SERVER -----
SIMULE: CPU_INTENSIVA
-----
Starting CPU stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting CPU stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
DC
igorbrito@: |

igorbrito@: ./cliente 192.168.15.7 25565
Local: 172.25.139.129 44840
Remote: 192.168.15.7 25565
----- RECIEVED FROM SERVER -----
Hello from server!
Time: Tue Sep 19 23:11:42 2023
PID: 16380
-----
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
SIMULE: CPU_INTENSIVA
-----
Starting CPU stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting CPU stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
SIMULE: MEMORIA_INTENSIVA
-----
Starting MEM stress
----- RECIEVED FROM SERVER -----
DC
igorbrito@:
```

Figura 7: Saída padrão dos três clientes

Gravamos as execuções e elas podem ser vistas nesses links: [Server](#) e [Clientes](#). Coloque uma do lado da outra e execute ao mesmo tempo.

Nesse exemplo nosso processo pai tinha ID 16299 e os processos filhos que estavam lidando com a comunicação com os clientes tinham IDs 16346, 16351 e 16380. Sendo assim conseguimos distinguir quais logs pertencem a quais clientes.