Trabalho 2.2 - Servidor TCP Concorrente (MC833)

José Ribeiro Neto - RA 176665

Utilize cliente.c e servidor.c disponibilizados para realizar este 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.

   R: Não, o servidor não aceitará a conexão dos dois clientes de forma concorrente, dado que não existe mecanismo de multi-thread implementado no servidor. Como podemos ver na imagem abaixo, primeiro o servidor atende um cliente e só depois atende o outro. Podemos constatar este fato olhando a diferença de 5s do time recebido pelo cliente. Este valor de 5s representa exatamente o sleep utilizado no código do cliente.

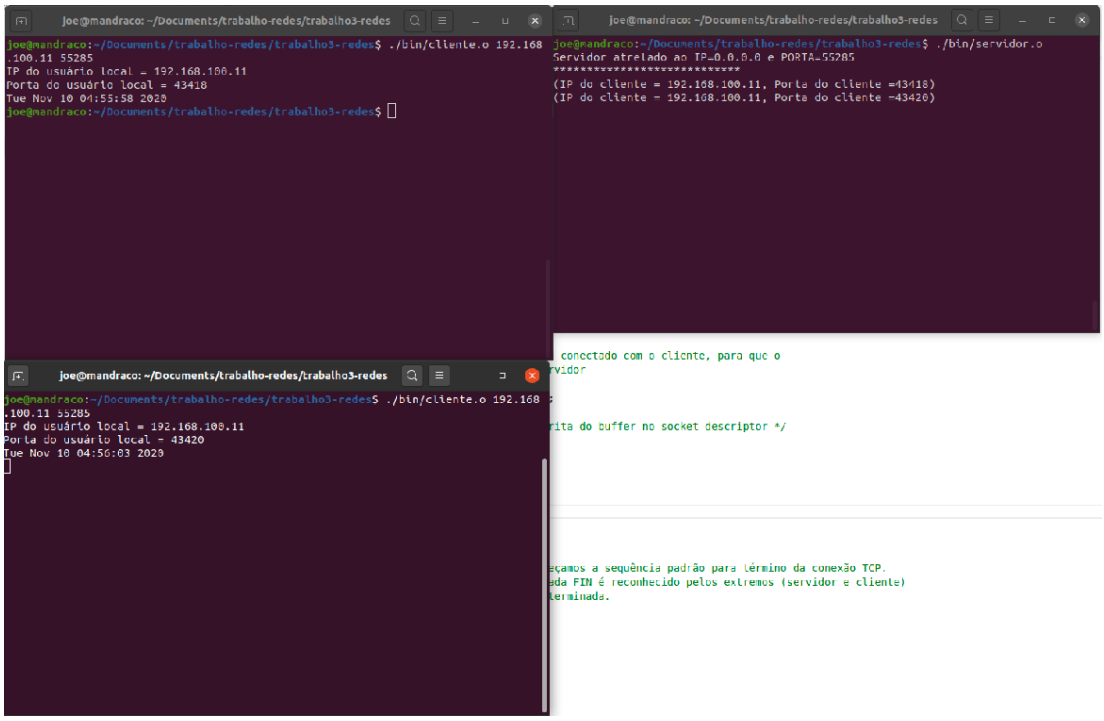

Figure 1: Resposta da questão 1

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.

R: Como podemos ver na imagem 2 abaixo, o servidor está atendendo múltiplas requisições. Em cada requisição, o servidor envia uma cadeia de caracters (no exemplo da imagem, a cadeia 'date') que por sua vez é executada no cliente que então retorna o resultado para o servidor.

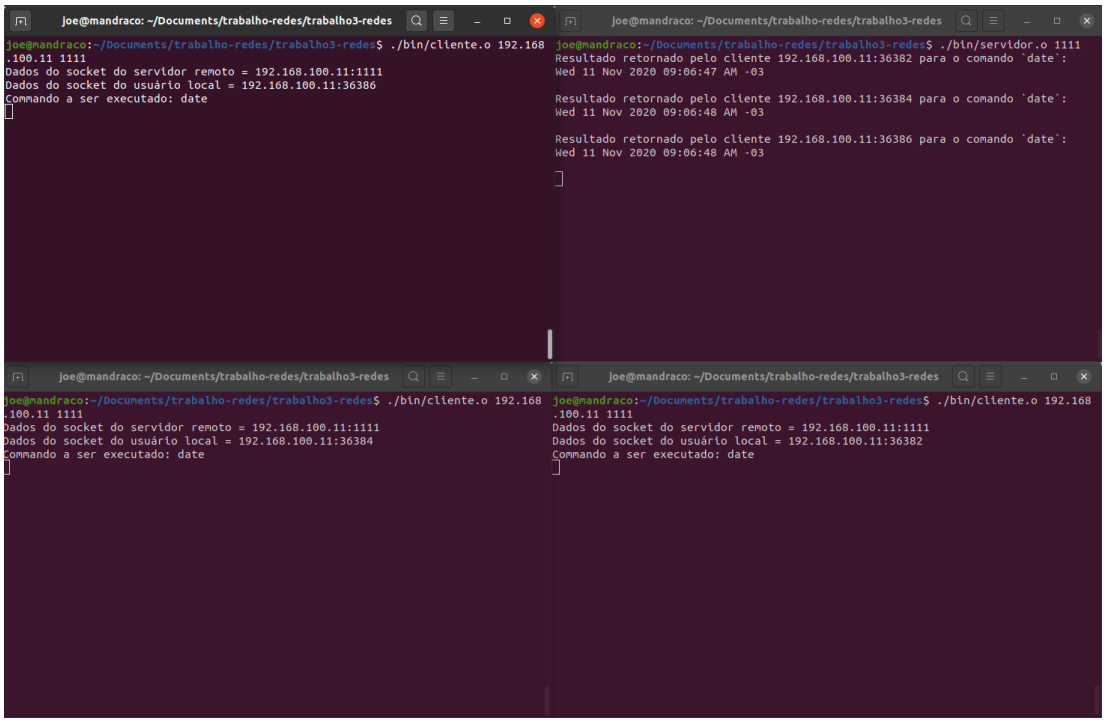

Figure 2: Resposta da questão 2

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.

   R: Isto foi feito. Porém, para evitar problemas de escrita simultâneas em arquivos, dado que o servidor atende simultaneamente clientes, optei por criar um arquivo para cada cliente atendido, utilizando para tanto o padrão 'user_file_X.txt', sendo 'X' o contador de clientes atendidos pelo servidor, começando em 0.

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.

   (d) O servidor irá escrever em um arquivo texto o endereço IP, porta, instante de conexão e de desconexão para cada cliente.

R: Como podemos ver na imagem da figura 3 abaixo, o servidor atende as execuções de forma concorrente. Os clientes executam a cadeia de caracters e enviam o resultado para o servidor. Então o servidor armazena os resultados em arquivos txt. No caso, como foram executados 3 clientes, temos três arquivos txt na imagem (user_file_0.txt, user_file_1.txt, user_file_2.txt). Mais informações sobre os detalhes de implementação encontram-se no programa, onde o código foi comentado.

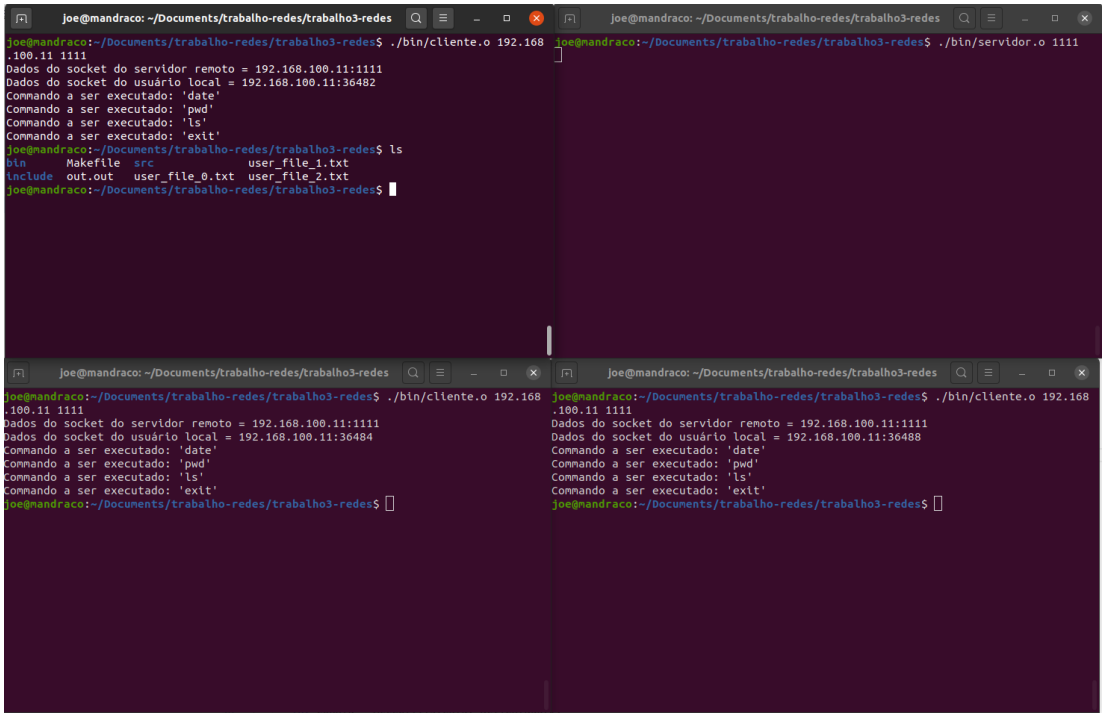

Figure 3: Resposta da questão 4

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.

   R: Como nosso programa foi implementado, o servidor envia uma cadeia de caracters igual a ‘exit‘ que quando recebida pelo cliente, faz com que o mesmo pare de escutar requisições de comandos pelo servidor. Assim, tanto o cliente como o servidor, após o envio dessa string, começam o processo de fechamento da conexão. Portanto, ambos enviam FIN um para o outro. Assim, sempre o cliente e o servidor receberão o FIN. Este fato pode ser compravo com base na figura 4 abaixo, onde usamos o comando ’nestat’ para verificar o estado dos sockets. Nela, vemos que o socket do servidor passa do estado ESTABLISHED, para o TIME_WAIT, e depois some do output do netstat (representando que o socket deixou de existir). Já o socket do cliente aparece somente em estado ESTABLISHED e depois já desaparece, representando assim que o mesmo de fato recebeu FIN e então (depois de seguir todo o processo) fechou de fato a conexão do socket.

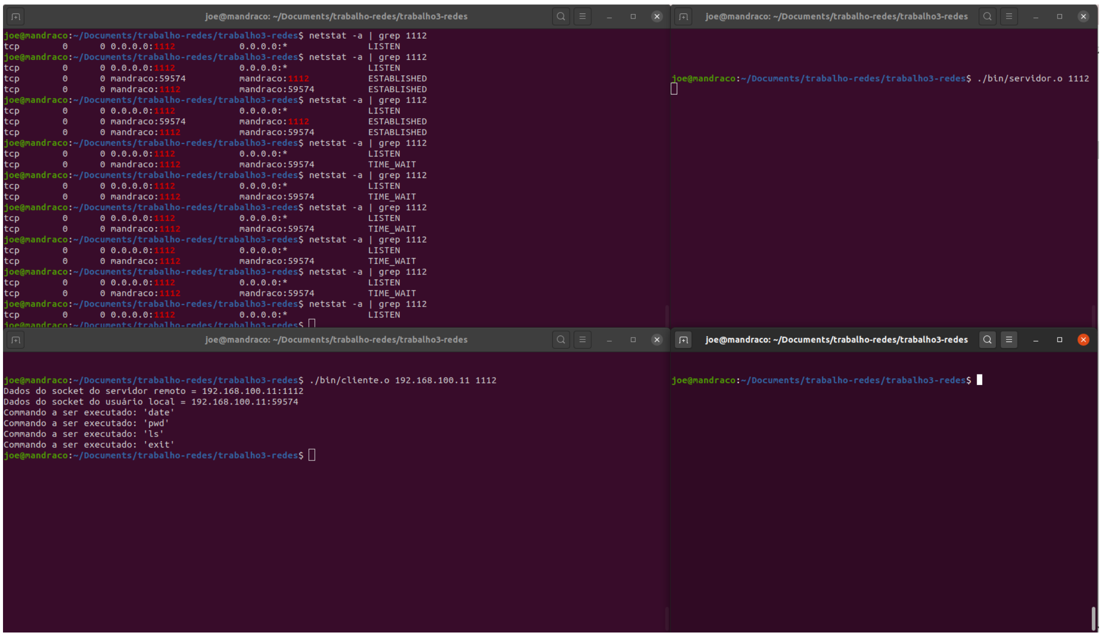

Figure 4: Resposta da questão 5

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.

   R: Para comprovar que os processos criados pelo servidor através do 'fork' são de fato filhos do processo principal, utilizamos a ferramenta 'htop' e printamos no código do servidor o 'pid' dos processos filhos criados bem como o do processo pai. Verificamos que pela figura 5 apenas temos o processo 163984 executando (o pid do processo pai, mostrado no terminal superior da esquerda), uma vez que nenhum cliente fez ainda requisição. Já quando rodamos o código do cliente mostrado na figura 6, o servidor cria um novo processo para atender a requisição do mesmo, dado pelo 'pid' de 164035. Verificamos que este pid de fato é o do filho, pois olhando para o terminal rodando o htop (terminal superior da esquerda), notamos que surgiu um novo processo de pid exatamente igual ao printado no servidor (pid = 164035). E o mesmo ocorre quando executamos o segundo cliente, apresentado na figura 7, onde o pid do novo processo filho é printado no servidor (igual a 164055), e exatamente no mesmo instante surgi um novo processo no terminal do htop, com o mesmo pid deste processo filho criado. Por fim, quando as requisições dos clientes terminam, verificamos pela imagem 8 que todos os processos filhos sumiram, restando somente o processo pai (com pid igual a 163984). Portanto, comprovamos que os processos criados para manipular requisições de clientes são de fato processos filhos. Em adendo, certamente poderíamos ter utilizado o comando 'ps' para obter tais dados também, como mostra a figura 9. Nela, vemos que dois processos filhos foram criados (pid = 171990, pid = 172026), como mostra a aba do terminal do servidor (superior à direita), que por sua vez aparecem na aba do terminal executando o comando ps (superior à esquerda). E como podemos ver da aba executando o ps, estes processos filhos de fato possuem o processo 171955 como pai (como já havia sido printado no terminal do servidor).

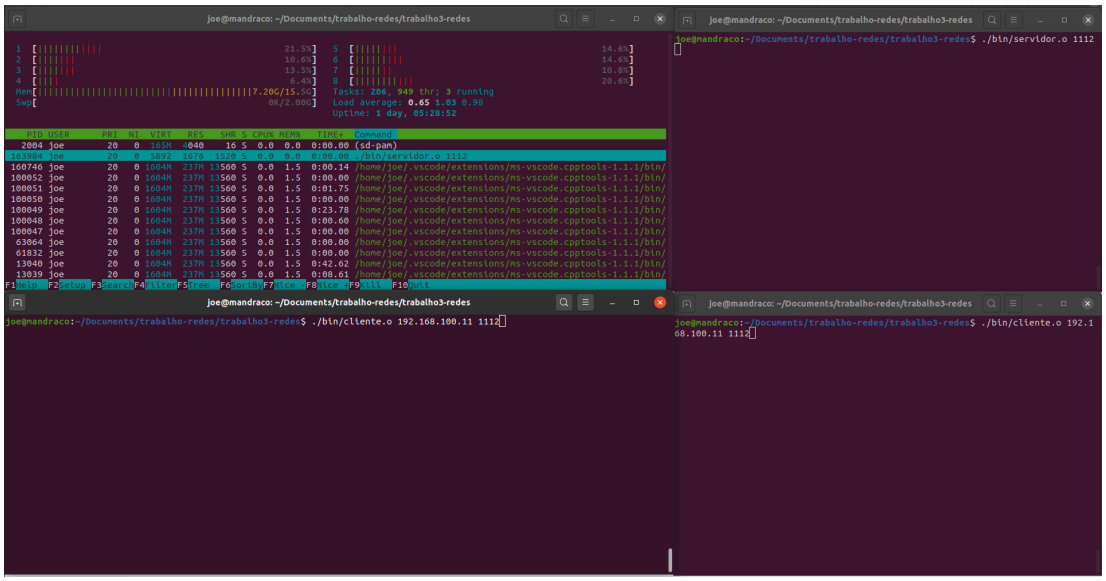

Figure 5: Resposta da questão 6a

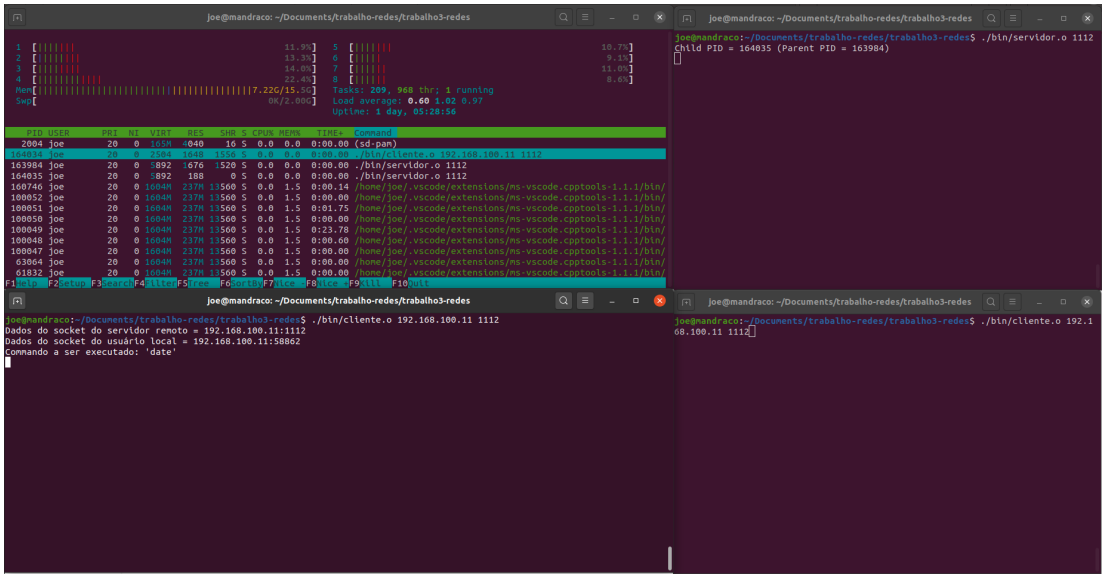

Figure 6: Resposta da questão 6b

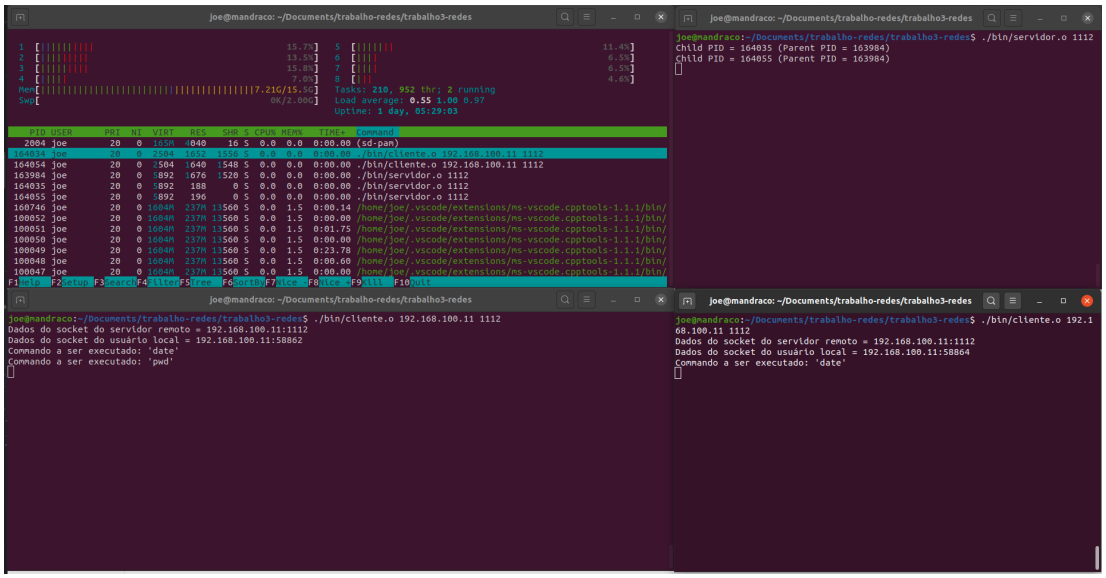

Figure 7: Resposta da questão 6c

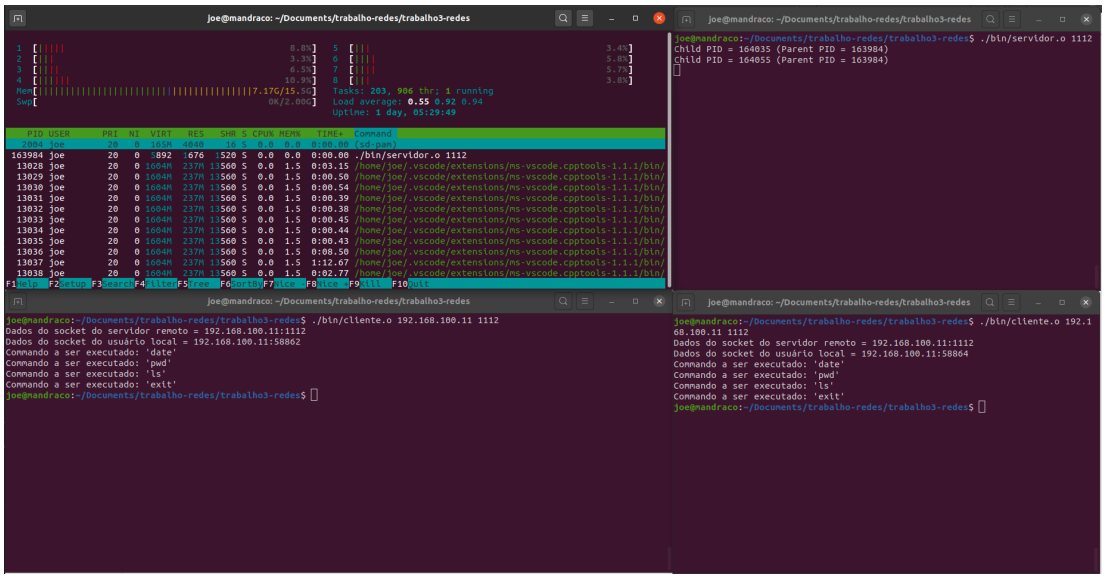

Figure 8: Resposta da questão 6d

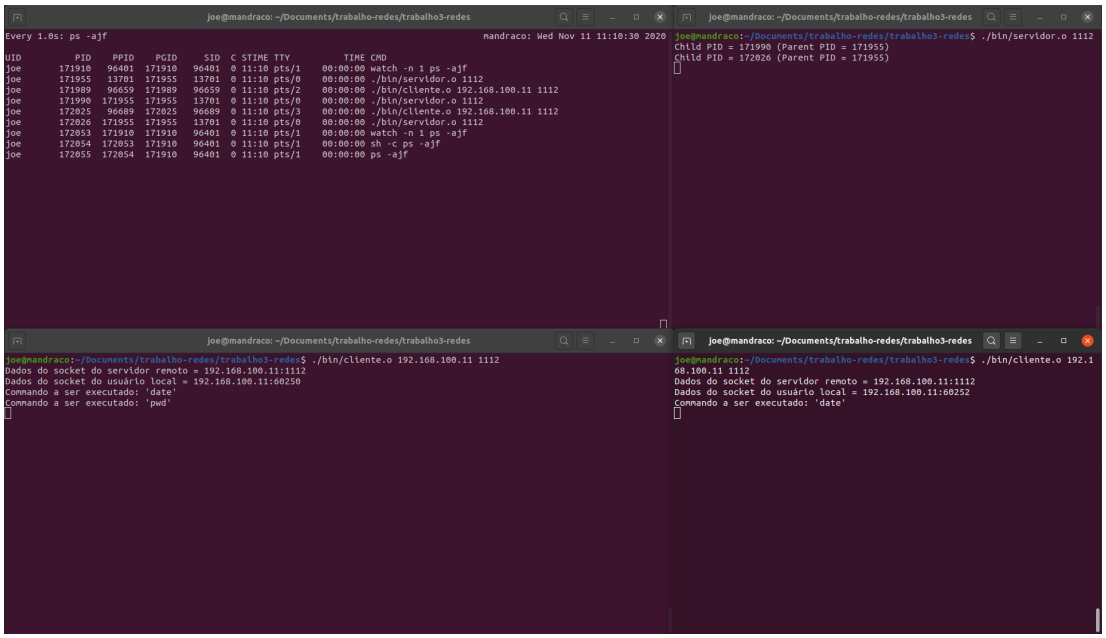

Figure 9: Resposta da questão 6e

Instruções de compilação e Execução:

1. **Instruções de Compilação: apenas rodar o comando 'make' no terminal a partir da pasta raiz. O comando 'make' utilizará o arquivo 'Makefile' para chamar o g++ e assim compilar o código.**
2. **Instruções de Execução:**
   (a) **Primeiro rodar o servidor com o comando './bin/servidor.o X', sendo X aqui a porta onde o servidor ficará escutando requisições.**
   (b) **Depois de executar o servidor, executar o comando './bin/cliente.o Y X' pra rodar o programa do cliente. Aqui, 'X' representa a porta que o servidor está escutando (passada como argumento para o servidor) e 'Y' representa o IP que o servidor estará escutando requisições.**