

Atividade 2.2: Servidor TCP concorrente

Aluno: Guilherme Luis Domingues

RA: 155619

Instituto de Computação
Universidade Estadual de Campinas

Campinas, 22 de Outubro de 2020.

Sumário

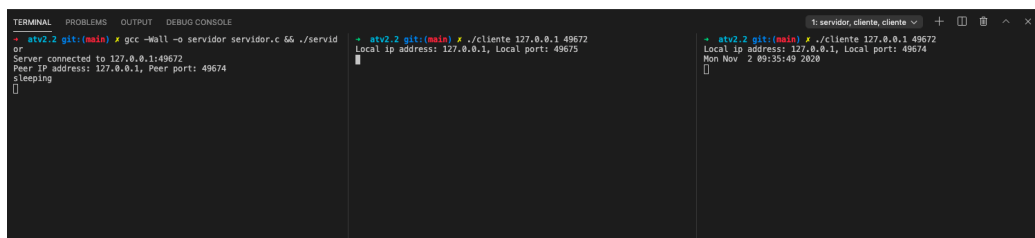
1	sleep()	2
2	Modificação cliente.c e servidor.c	3
3	Imprimindo Data e Hora	4
4	Demais modificações	5
5	Servidor sempre escutando?	5
6	Criando subprocessos	5
7	Para executar o programa	6
7.1	servidor.c	6
7.2	cliente.c	6

1 sleep()

Ao utilizar o comando `sleep(10)` logo antes de encerrar a conexão do peer pelo servidor. Além disso, para compreender o estado do servidor (sleeping ou waking up), colocamos um `print`.

Ao utilizar o comando `sleep()` o servidor não aceita mais de uma conexão por vez, isto é, não aceita conexões de forma concorrente. As conexões estabelecidas entre ele e os clientes acontece de forma sequencial, como apresentado nas figuras abaixo.

A Figura 1 ilustra o servidor executando e conectado com o terminal 3, o mais à direita. Enquanto isso, repare que o terminal do meio confirmou o endereço do servidor, porém não obteve nenhuma mensagem do mesmo.



```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
- atv2.2 git:(main) # gcc -Wall -o servidor servidor.c && ./servid
or
Server connected to 127.0.0.1:49672
Peer IP address: 127.0.0.1, Peer port: 49674
sleeping
[]

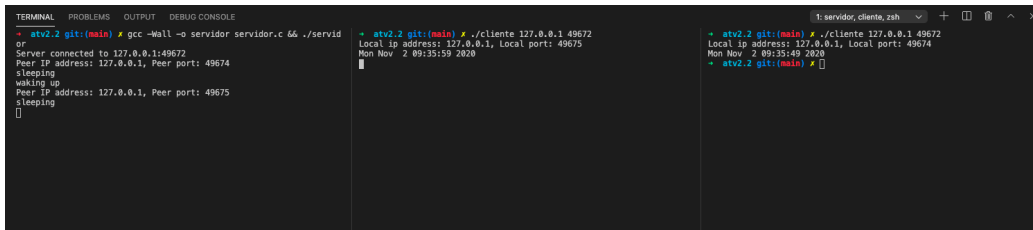
- atv2.2 git:(main) # ./cliente 127.0.0.1 49672
Local ip address: 127.0.0.1, Local port: 49675
[]

- atv2.2 git:(main) # ./cliente 127.0.0.1 49672
Local ip address: 127.0.0.1, Local port: 49674
Mon Nov 2 09:35:49 2020
[]

```

Figura 1: Conexão servidor-terminal 3

Neste momento, a conexão com o primeiro cliente foi finalizada com sucesso. Então, o servidor saiu do "sleeping" e ficou novamente escutando. Sendo assim, aceitou e respondeu a conexão do terminal 2 e chamou novamente na função `sleep`.



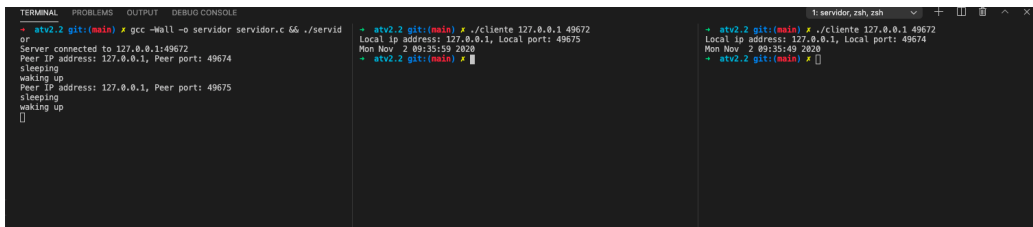
```
atv2.2 git:(main) # gcc -Wall -o servidor servidor.c && ./servid
or
Server connected to 127.0.0.1:49672
Peer IP address: 127.0.0.1, Peer port: 49674
sleeping
waking up
Peer IP address: 127.0.0.1, Peer port: 49675
sleeping
[]

+ atv2.2 git:(main) # ./cliente 127.0.0.1 49672
Local IP address: 127.0.0.1, Local port: 49675
Mon Nov 2 09:35:59 2020

+ atv2.2 git:(main) # ./cliente 127.0.0.1 49672
Local IP address: 127.0.0.1, Local port: 49674
Mon Nov 2 09:35:49 2020
+ atv2.2 git:(main) # []
```

Figura 2: Conexão servidor-terminal 2

Por fim, ao encerrar a conexão do cliente 2, o servidor saiu do sleep e estava apto a receber outras conexões novamente.



```
atv2.2 git:(main) # gcc -Wall -o servidor servidor.c && ./servid
or
Server connected to 127.0.0.1:49672
Peer IP address: 127.0.0.1, Peer port: 49674
sleeping
waking up
Peer IP address: 127.0.0.1, Peer port: 49675
sleeping
waking up
[]

+ atv2.2 git:(main) # ./cliente 127.0.0.1 49672
Local IP address: 127.0.0.1, Local port: 49675
Mon Nov 2 09:35:59 2020

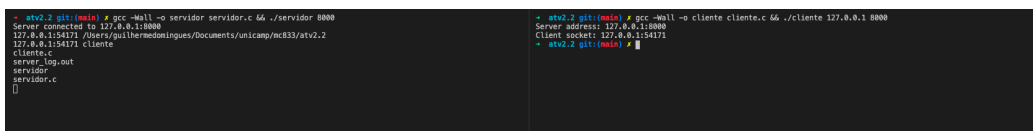
+ atv2.2 git:(main) # []

+ atv2.2 git:(main) # ./cliente 127.0.0.1 49672
Local IP address: 127.0.0.1, Local port: 49674
Mon Nov 2 09:35:49 2020
+ atv2.2 git:(main) # []
```

Figura 3: Servidor apto à novas conexões

2 Modificação cliente.c e servidor.c

Após algumas modificações no código do cliente e do servidor, utilizados no exercício anterior, as saídas obtidas após as modificações estão apresentadas na Figura 4.



```
atv2.2 git:(main) # gcc -Wall -o servidor servidor.c && ./servidor 8000
Server connected to 127.0.0.1:8000
127.0.0.1:154171 /Users/guilhermedomingues/Documents/unilcamp/mc833/atv2.2
cliente.c
server_log.out
servidor
servidor.c
[]

+ atv2.2 git:(main) # gcc -Wall -o cliente cliente.c && ./cliente 127.0.0.1 8000
Server address: 127.0.0.1:8000
Client socket: 127.0.0.1:154171
+ atv2.2 git:(main) # []
```

Figura 4: Conexão cliente-servidor após modificações

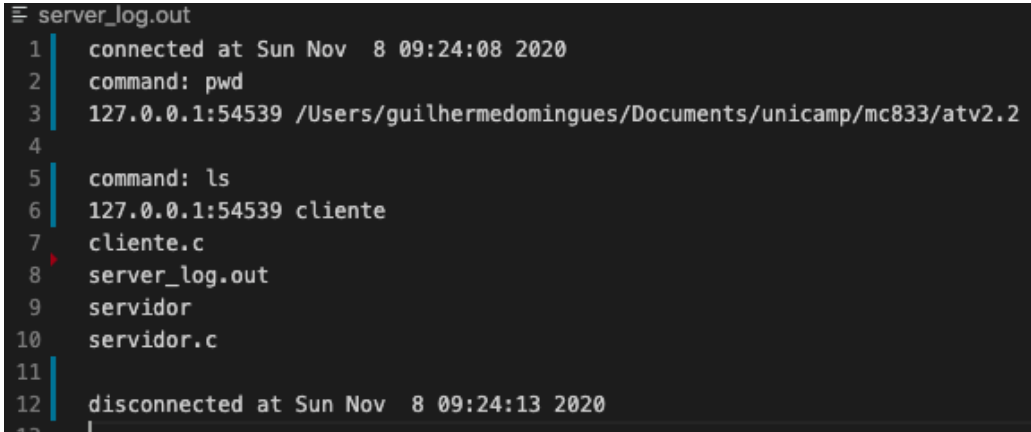
Podemos ver que a saída do servidor apresenta as respostas dos comandos enviados por ele para o cliente. Nesse caso, foram os comandos pwd e

ls.

Na saída do cliente existem apenas as informações da conexão, tanto do servidor quanto sua própria. Em ambos os casos, as informações são endereço de IP e porta.

3 Imprimindo Data e Hora

Para que a data e hora da conexão do cliente com o servidor fosse salva no arquivo, criamos uma função chamada `logDateTime(FILE *file, char* status)`, a qual recebe o arquivo e o status (connected, disconnected). O arquivo, após esta modificação ficou como mostra a Figura 4.



```
server_log.out
1  connected at Sun Nov  8 09:24:08 2020
2  command: pwd
3  127.0.0.1:54539 /Users/guilhermedomingues/Documents/unicamp/mc833/atv2.2
4
5  command: ls
6  127.0.0.1:54539 cliente
7  cliente.c
8  server_log.out
9  servidor
10 servidor.c
11
12 disconnected at Sun Nov  8 09:24:13 2020
13 |
```

Figura 5: Arquivo de log do servidor com data e hora da conexão e desconexão do cliente.

4 Demais modificações

5 Servidor sempre escutando?

O servidor estará sempre escutando para receber novos clientes pois, uma vez que o cliente manda o comando de terminar, apenas ele é desconectado, o servidor ainda continua executando e demais clientes podem se conectar nele novamente. A Figura 6 mostra o servidor esperando por novas conexões mesmo com o cliente tendo enviado "quit" para o mesmo.

```
+ netstat -p tcp -van | grep 8000
tcp4      0      0 127.0.0.1.8000      *.*          LISTEN    131072 131072  22623      0 0x0000 0x00000002
```

Figura 6: Servidor sendo executado após encerramento do cliente

6 Criando subprocessos

Ao utilizar o comando fork, uma nova instância daquele processo é criada. Com isso, um novo processo para o servidor é instanciado à medida que os clientes vão se conectando à ele. A Figura 7 mostra o caso de dois clientes conectados ao servidor. Na imagem é possível ver o processo pai PID 26952 e os processos filhos, PID 26992 e 27057.

```
[~] ~ pstree 26925
--+= 26925 guilhermedomingues ./servidor 8000
|--- 26992 guilhermedomingues (servidor)
\--- 27057 guilhermedomingues (servidor)
```

Figura 7: Subprocessos do Servidor sendo utilizado por clientes diferentes.

7 Para executar o programa

7.1 servidor.c

O servidor é compilado e executado normalmente.

```
1 gcc -Wall -o servidor servidor.c // Compilar o servidor
2 ./servidor <porta> // Executar o servidor, passando o nmero da porta como
   parmetro
```

A lista com os comandos enviados para o cliente está na linha 111. Além da lista, deve-se alterar também a constante `NUM_OF_COMMANDS`, a qual deve estar de acordo com o número de comandos existentes na lista.

```
1 char commands[NUM_OF_COMMANDS][MAXDATASIZE] = {
2     "pwd",
3     "ls",
4     "ps"
5 };
```

7.2 cliente.c

O cliente deve ser compilado normalmente. Para ser executado, deve-se passar como primeiro parâmetro o endereço do servidor e segundo parâmetro a porta que o servidor está ouvindo.

```
1 gcc -Wall -o cliente cliente.c // Compilando o cliente
2 ./cliente <ip> <porta> // Executando o cliente com o IP e porta
```

Além disso, para que o cliente feche a conexão com o servidor, o último comando que o servidor envia para o cliente deve ser o "exit". Desta forma, o cliente recebe este comando e responde para o servidor que, por sua vez, finaliza a conexão com aquele cliente.