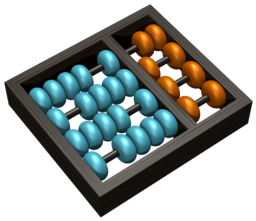

Exercício 3: Servidor TCP Concorrente

Alunos: Luiz Fernando Lima Leite e Mateus Da Costa E Silva R. A. De Andrade

RA: 248405 e 230806

Instituto de Computação

Universidade Estadual de Campinas

Campinas, 30 de Setembro de 2025.

Sumário

1 Implementação

1.1 Teste de Servidor com Comando sleep

Ao inserir o comando sleep antes de fechar a conexão TCP e conectar dois clientes de forma quase instantânea, observa-se que o servidor não aceita fazer duas conexões de forma concorrente. Em vez disso, o servidor espera o tempo do sleep até aceitar a próxima conexão, o que é esperado, pois o loop for está estruturado para aceitar as conexões de forma sequencial.

1.2 Implemementação do Servidor TCP

Foi implementado um servidor TCP que recebe requisições HTTP GET / HTTP/1.0 e GET / HTTP/1.1, respondendo com o trecho abaixo para estas requisições ou com uma mensagem 404 Not Found para demais requisições.

```
"HTTP/1.0 200 OK\r\n"
"Content-Type: text/html\r\n"
"Content-Length: 91\r\n"
"Connection: close\r\n"
"\r\n"
"<html><head><title>MC833</title</head><body><h1>MC833 TCP"
"Concorrente </h1></body></html>"
```

Em vez de lidar com conexões de forma sequencial, esta versão do servidor cria processos filhos para administrar a conexão com o cliente, enquanto que o processo pai se encarrega apenas de aceitar a requisição e obter o socket de conexão que será utilizado pelos filhos.

1.3 Implementação do Cliente Browser

Foi implementado um cliente simples que recebe um IP e uma porta de um servidor via linha de comando, e utiliza estes dados para fazer a requisição HTTP abaixo e imprimir a resposta dela na saída padrão.

```
"GET / HTTP/1.0\r\n"
"Host: teste\r\n"
"\r\n"
```

2 Questões

2.1 Questão 1

No servidor, ao inserir o comando sleep no filho antes de fechar connfd, é possível observar concorrência quando dois clientes se conectam quase ao mesmo tempo. Isso demonstra que o processo pai continua aceitando novas requisições e criando novos processos filhos mesmo que haja outros filhos com conexões ainda ativas. É por isso que a utilização do comando sleep no filho deixa evidente a capacidade de realizar conexões simultâneas do servidor.

2.2 Questão 2

No trecho abaixo, o bloco de código dentro do if só é executado em processos filhos, uma vez que eles são os únicos que possuem pid igual a zero. Logo, a função close que fecha o socket de escuta nunca é executada no processo pai, permitindo que o servidor continue em modo de escuta mesmo após aceitar uma requisição.

Assim que o processo filho do servidor termina de executar a função

doit e executa a função Close no socket de conexão, ele envia uma mensagem FIN para o cliente, que deve ser respondida com uma mensagem ACK do cliente. Como o processo filho é encerrado abruptamente após o Close, há risco do ACK do cliente não chegar ao servidor a tempo, e por isso a conexão fica viva em estado de TIME_WAIT mesmo após o Close, em vez de completar o 4-way handshake e encerrar a conexão como esperado.

Apesar disso, não há nenhum Close sobrando neste código. Todos eles são necessários e corretos para realizar a conexão TCP. O primeiro Close é correto pois o processo filho só lida com a parte de conexão, e não com a parte de escuta. Como as referências ao socket de escuta foram duplicadas no fork, é preciso utilizar o comando Close no socket de escuta dentro do processo filho. O segundo Close é necessário pois o processo filho já realizou seu processamento para aquela conexão, e ela deve ser encerrada. O terceiro Close é correto pois o processo pai não lida com conexões, apenas com a parte de escuta, e como as referências ao socket de conexão foram duplicadas no fork, a função Close é necessária.

2.3 Questão 3

Não, pois embora o servidor mantenha o processo pai escutando, o processo filho lida com a conexão e envia o FIN.

2.4 Questão 4

A screenshot mostra os processos utilizados pelo servidor para lidar com as conexões.

```
bash-5.2$ ps f | grep servidor
  14171 pts/3    S+     0:00  \_ grep servidor
   5908 pts/0    S      0:00  \_ ./servidor
   5971 pts/0    Z      0:00  |   \_ [servidor] <defunct>
   6007 pts/0    Z      0:00  |   \_ [servidor] <defunct>
  14101 pts/0    S+     0:00  \_ ./servidor 5000 10
  14107 pts/0    Z+     0:00      \_ [servidor] <defunct>
  14164 pts/0    Z+     0:00      \_ [servidor] <defunct>
```

2.5 Questão 5

O estado TIME_WAIT indica que a conexão já foi fechada, e que o servidor já enviou o FIN para o cliente. A conexão se mantém nesse estado apenas para segurar a porta local caso algum pacote atrasado chegue após a conexão ser fechada, possivelmente confundindo uma outra aplicação que poderia usar a mesma porta. Isso pode ser visto na screenshot abaixo.

```
bash-5.2$ netstat -tanp | grep :5000
(Nem todos os processos puderam ser identificados, informações sobre processos
 de outrem não serão mostrados, você deve ser root para vê-los todos.)
tcp        0      0 127.0.0.1:5000          0.0.0.0:*               OUÇA       8204/./servidor
tcp        0      0 127.0.0.1:42016         127.0.0.1:5000          TIME_WAIT   -
tcp        0      0 127.0.0.1:42010         127.0.0.1:5000          TIME_WAIT   -
```

3 Ambiente de Execução

Foi utilizada a máquina "zecolmeia"(143.106.16.43) para a execução dos testes.