Exercício 5 de MC833 — Programação em Redes de Computadores

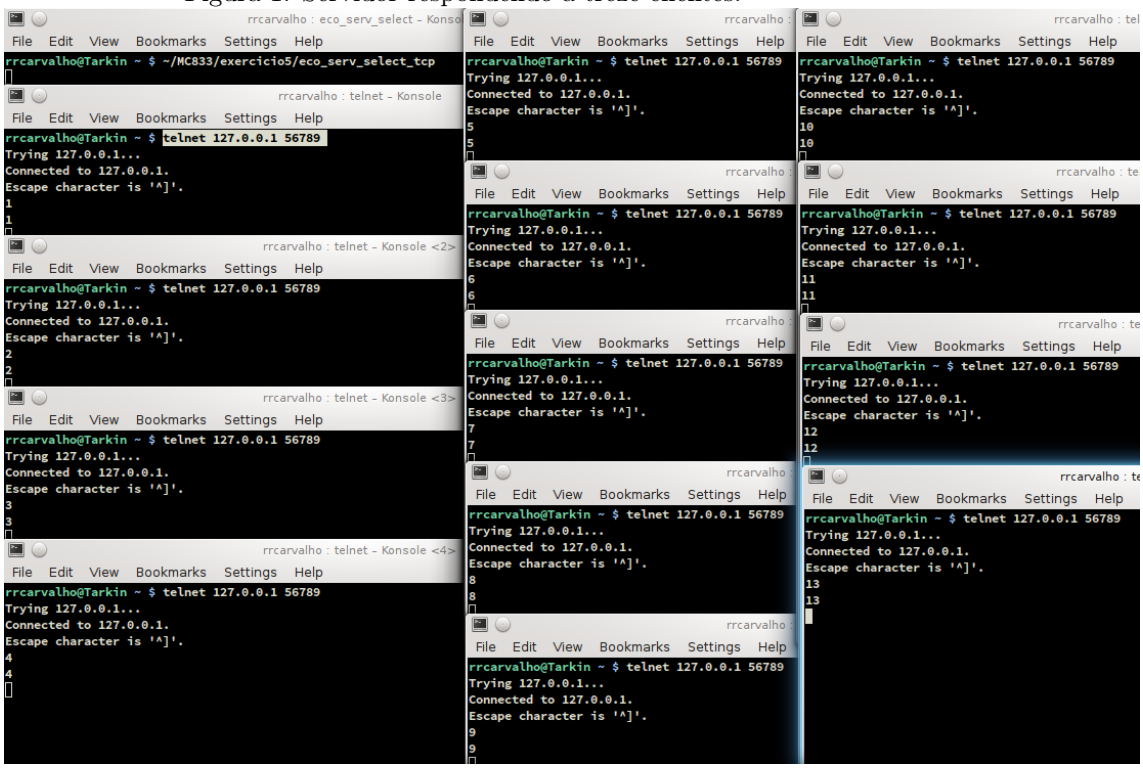
Raul Rabelo Carvalho, 105607, turma A

8 de Abril de 2014

1

O servidor pôde responder com um eco dos treze clientes como mostrado na figura 1, pois, na verdade, ele o faz em sequência. A função `select`[1] empregada no servidor marca em um vetor de bits (do tipo `fd_set`) as conexões que o *kernel* recebeu (já que, no caso do servidor deste exercício, faz-se somente leituras). As funções `read` e `write` usam a posição no vetor como descritor, já que este inteiro equivale aos descritores que o *kernel* criou. Assim, fazendo uma iteração rápida sobre as posições do vetor marcadas com conexões ativas, o servidor, para efeitos práticos, trata as requisições dos clientes simultaneamente.

Figura 1: Servidor respondendo a treze clientes.



2

Como visto no exercício 4 desta disciplina, um servidor concorrente pode empregar a chamada de sistema `fork` dos sistemas Unix para criar uma cópia própria para tratar de conexões simultâneas. Já um servidor por multiplexação de entrada e/ou saída não faz cópias de si próprio, mas emprega a capacidade de um *kernel* Unix em gerenciar diversos descritores de entrada e saída por processo.

[1]Stevens, W. Richard. Unix Network Programming, volume 1. Second Edition, pp. 150-155.

O *kernel* Unix pode gerenciar virtualmente um número ilimitado de descritores de E/S, dependente somente da memória alocada para cada descritor. No entanto, existe um limite *hardwired* de 1024, que é o valor da macro `FD_SETSIZE`[2]. Este valor pode ser alterado, mas para que o sistema utilize do novo valor, o *kernel* precisa ser recompilado. Há outro problema no fato da iteração sobre o total de descritores: se este valor aumentar muito, a interação pode se tornar demorada e o tempo de resposta de cada conexão pode aumentar, mitigando a impressão de simultaneidade das conexões.

No entanto, a solução com o `fork` pode sofrer perdas de desempenho ainda maiores, pois o efeito de simultaneidade também é conseguido por um tipo de iteração, neste caso o escalonamento de processos pelo sistema. Quando o número de processos filhos do servidor, cada um atentendo a uma conexão aumenta muito, o atraso devido a troca de contexto pode se tornar um fator que piora o desempenho. Outro problema é que cada processo filho tem uma cópia de todo contexto de execução do servidor, ocupando mais memória no *host* que o servidor multiplexado com a pilha e o *heap* de cada processo.

Deste modo, a solução para múltiplas conexões ao servidor por multiplexação tem uma melhor capacidade de escalabilidade, pelo menos até o limite de 1024 conexões por servidor, já que incorre de um gasto menor de memória e ciclos (troca de contexto) da máquina hospedeira.

[2]Stevens, W. Richard. Unix Network Programming, volume 1. Second edition, pp. 154-155.