MC833 - Tarefa 5

Erik de Godoy Perillo - RA: 135582

6 de maio de 2016

1. (a) `select(int nfds, fd_set* readfds, fd_set writefds, fd_set* exceptfds, struct timeval* timeout)`

   A função `select` monitora grupos de *file descriptors* e indica quais deles estão prontos para uma certa operação de I/O. As operações podem ser de leitura (com os *file descriptors* indicados em `readfds`) ou de escrita (indicados por `writefds`). Pode-se também indicar um grupo de exceções (indicado por `exceptfds`). Passa-se o limite superior máximo (não-inclusivo) do *range* dos *file descriptors* o qual se quer inspecionar pelo argumento `nfds`. A operação pode ter um timeout para um dos *file descriptors* estar disponível especificado em `timeout`. Se esse argumento é NULL, então a função bloqueia até um *file descriptor* estar pronto. O gerenciamento dos grupos é feito através de macros:

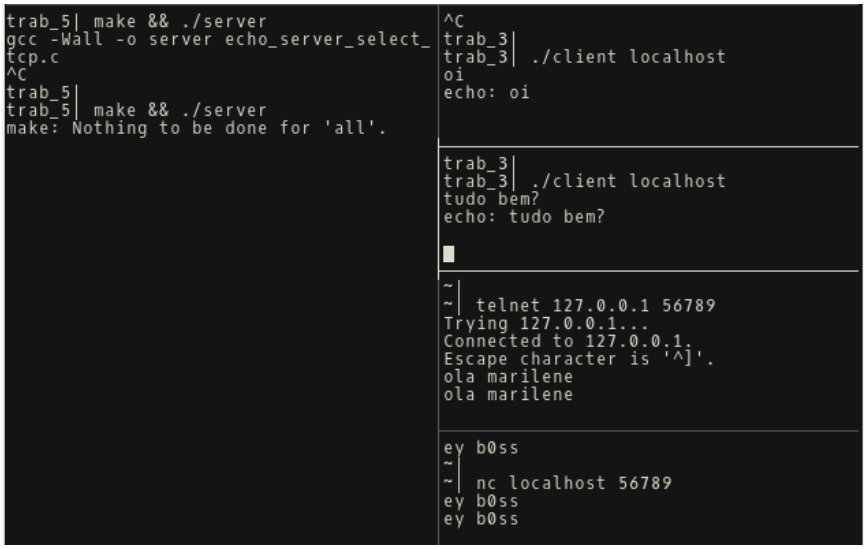
   (b) `FD_ZERO(set)` - Limpa um grupo.

   (c) `FD_SET(fd, set` - Adiciona um file descriptor a um grupo.

   (d) `FD_CLR(fd, set)` - Remove um file descriptor de um grupo.

   (e) `FD_ISSET(fd, set)` - Verifica se um file descriptor está em um certo grupo. Essa macro é usada após a chamada de `select` para se averiguar se um certo *file descriptor* está disponível para uma certa operação.

2. Um console fala mais que mil palavras. Na imagem abaixo, o servidor foi invocado à esquerda. À direita, invoca-se duas instâncias do echo_client do trabalho 3 (compilados para conectar na porta 56789), o telnet e o nc, todos conectando ao endereço do servidor. Pode-se ver que todos obtêm a resposta esperada (echo) do servidor.



3. O truque é obtido por meio de um loop sobre todos os sockets das conexões e o uso da função `select`. No começo de cada loop, checa-se se há mais conexões esperando para serem feitas e, se houver, uma nova conexão é feita e adicionada à lista de conexões (o vetor `client`). Após isso, itera-se sobre todos os sockets de conexões que podem estar ativas. Com o resultado da chamada a `select` no começo do loop e a macro `FD_ISSET`, checa-se se o socket analisado está disponível para leitura. Se sim, recebe-se uma mensagem e manda-se ela de volta. Resumindo: o processo é serial mas, por iterar-se o tempo todo sobre os clientes, é dada a impressão de paralelismo.

4. Foi adicionada checagem de erro para as chamadas de `select`, `accept`, `read` e `send`, sempre fechando-se os sockets necessários.

5. A principal diferença é que, para a atividade anterior, era criado um processo para cada nova conexão. Na atividade atual, é usado apenas um processo para todas as conexões juntamente a um loop infinito que itera sobre todas elas. Com relação a recursos, a abordagem atual é mais eficiente pois criar um processo é uma tarefa relativamente pesada para o sistema operacional, além de que seria preciso usar um *buffer* para cada conexão. Para o que o servidor se propõe a fazer, é mais adequado o uso de um processo só, assumindo que não é esperado uma uma interação tão rápida e intensa entre o cliente e o servidor para um serviço de echo, além de que é uma tarefa simples que pode ser geren-

ciada facilmente por um único processo e vários servidores. Se muitos usuários conectassem ao servidor que usa `fork`, haveria um problema de escalabilidade muito mais rapidamente que com o servidor atual, pois a criação/manipulação de processos pelo sistema operacional seria uma tarefa muito mais custosa do que a atividade do echo em si.