UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2017/1
AULA PRÁTICA: SOCKETS C

---

1. OBJETIVO DA AULA PRÁTICA:

Esta aula prática tem por objetivo ilustrar o uso da API de sockets UDP para a comunicação entre processos remotos. Você deverá implementar uma aplicação bastante simples que utilize a API sockets na linguagem C. A aplicação permitirá que um processo servidor execute operações matemáticas quando um processo cliente enviar solicitações para esse servidor.

O desenvolvimento da aplicação será realizado passo a passo, como descrito abaixo. Esta aula prática irá lhe proporcionar a oportunidade de entender como alguns conceitos apresentados na aula 11 são implementados na prática. Sinta-se a vontade para utilizar os conhecimentos de programação adquiridos em aplicações mais sofisticadas. Essa mesma aplicação será abordada em aulas práticas futuras, porém utilizando outras APIs de comunicação entre processos, de forma que o aluno possa melhor entender as diferenças entre esses mecanismos.

Como material de apoio, recomenda-se o seguinte tutorial:
http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html

2. CALCULADORA REMOTA

Você deverá implementar uma aplicação cliente-servidor em sockets UDP na linguagem C, onde um servidor deverá disponibilizar a funcionalidade de uma calculadora, e onde o cliente irá solicitar a execução de operações matemáticas usando Datagramas UDP.

Passos

1. **Faça o download da implementação base:** baixe da página do moodle o arquivo **aula-12.zip**, e descompacte-o. Verifique no diretório descompactado quais arquivos estão presentes e examine o conteúdo deles.

2. **Especifique a estrutura de endereçamento do servidor:** você deverá criar uma variável do tipo `struct sockaddr_in` no arquivo **server_calculadora.c**. A estrutura deverá definir os diversos campos necessários para identificar um socket em uma rede IP, incluindo:

   ```
   serv_addr.sin_family = <Address_Family>;
   serv_addr.sin_port = <Port>;
   serv_addr.sin_addr.s_addr = <Address>;
   bzero(&(serv_addr.sin_zero), 8);
   ```

   Atenção: lembre-se de que em sockets cabe ao programador garantir que os bytes em tipos de dados numéricos sejam interpretados na ordem correta. Dessa forma você deve estar atento ao uso de funções que garantam a portabilidade do seu código, como por exemplo `htons`, `ntohs`, etc. Consulte o tutorial acima para maiores detalhes.

3. **Crie o socket e associe ele à estrutura de endereçamento definida no passo anterior:** o servidor deverá abrir um socket para que possa receber operações matemáticas requisitadas por programas clientes. Para isso, use a função `socket(int domain, int type, int protocol)` para criar um socket, especificando `AF_INET` como família de endereçamento e `SOCK_DGRAM` como tipo de serviço. Faça o tratamento adequado para testar condições de erro. Após, associe o socket recém criado ao endereço especificado no passo anterior utilizando a função `bind(int socket, struct sockaddr *address, socklen_t address_len)`.

4. **Implemente o laço de processamento do servidor**: o servidor deverá permanecer em um laço, recebendo requisições do socket, processando o cálculo desejado, e enviando os resultados para os clientes. Para ler dados do socket, utilize a função `recvfrom`:

   ```
   recvfrom(int socket, void *restrict buffer, size_t length, int flags,
            struct sockaddr *restrict address, socklen_t *restrict address_len)
   ```

   Assuma que os bytes que definirão a operação solicitada pelo cliente corresponderam a uma estrutura contendo os campos operador, operando1 e operando2, conforme definido a seguir:

   ```
   struct operacao
   {
       char operador;
       float operando1;
       float operando2;
   }
   ```

   O servidor deverá interpretar os bytes recebidos do socket corretamente e, de acordo com a operação solicitada, computar o resultado correspondente.

   Para retornar resultado da operação matemática ao cliente, utilize a função `sendto`:

   ```
   sendto(int socket, const void *buffer, size_t length, int flags,
          const struct sockaddr *dest_addr, socklen_t dest_len)
   ```

   O resultado da operação deve ser enviado como um dado do tipo float. Novamente, lembre-se de que em sockets cabe ao programador garantir que os bytes em tipos de dados numéricos sejam interpretados na ordem correta. Dessa forma você deve estar atento ao uso de funções que garantam a portabilidade do seu código. Consulte o tutorial acima para maiores detalhes.

5. **Faça o tratamento de erros e feche o socket**: caso qualquer erro aconteça durante a operação do socket, o mesmo deve ser fechado e o programa servidor deve ser encerrado.

6. **Primeira compilação:** nesse ponto, compile o código produzido até aqui utilizando o comando

   ```
   %>gcc –o server_calculadora server_calculadora.c
   ```

   Houve algum erro de compilação? Resolva qualquer problema no código antes de prosseguir para o próximo passo.

7. **Comece a definição do programa cliente:** neste ponto, você deverá implementar o código do cliente UDP. Para isso, o programa dever solicitar via linha de comando o `<endereço>` e a `<porta>` onde o servidor está executando.

8. **Especifique a estrutura definindo o endereço do servidor:** você deverá criar uma variável do tipo `struct sockaddr_in` no arquivo **client_calculadora.c**. Desta vez, no entanto, a estrutura não será usada para amarrar o socket local, mas sim como identificação do programa servidor. A estrutura deverá definir os diversos campos necessários para identificar o socket servidor, incluindo:

   ```
   serv_addr.sin_family = <Address_Family>;
   serv_addr.sin_port = <Port>;
   serv_addr.sin_addr.s_addr = <Address>;
   bzero(&(serv_addr.sin_zero), 8);
   ```

Atenção: utilize os dados para `<endereço>` e `<porta>` informados via linha de comando para auxiliar no preenchimento da estrutura. Além disso, lembre-se que em sockets cabe ao programador garantir que os bytes em tipos de dados numéricos sejam interpretados na ordem correta. Dessa forma você deve estar atento ao uso de funções que garantam a portabilidade do seu código. Consulte o tutorial acima para maiores detalhes.

9. **Crie o socket para comunicação com o servidor:** o cliente deve abrir um socket para que possa enviar operações matemáticas ao servidor. Para isso, use a função `socket(int domain, int type, int protocol)` para criar um socket, especificando `AF_INET` como família de endereçamento e `SOCK_DGRAM` como tipo de serviço. Faça o tratamento adequado para testar condições de erro. **Não** é necessário que o cliente associe (`bind`) o socket a qualquer endereço.

10. **Implemente o laço de processamento do cliente**: o cliente deverá permanecer em um laço, enviando requisições ao servidor através do socket, e recebendo os resultados. Para enviar dados através do socket utilize a função `sendto`, e para receber dados utilize a função `recvfrom` (de forma similar a que foram usadas no passo 4).

    a. **Receber dados via teclado:** a cada iteração do laço, solicite que o usuário forneça o operador (`char`) e dois operandos (`float`), e armazene os dados em uma `struct operacao` (definida no passo 4), que será enviada pelo socket.

    b. **Garantir portabilidade**: cabe ao programador garantir que os bytes em tipos de dados numéricos sejam interpretados na ordem correta, e dessa forma você deve estar atento ao uso de funções que garantam a portabilidade do seu código.

    c. **Tratar erros**: caso qualquer erro aconteça durante a operação do socket, o mesmo deve ser fechado e o programa cliente deve ser encerrado.

11. **Segunda compilação:** compile o código do cliente:

```
%>gcc –o client_calculadora client_calculadora.c
```

    Houve algum erro de compilação? Resolva qualquer problema no código antes de prosseguir para o próximo passo.

12. **Dispare a aplicação Servidor**:

```
%>./server_calculadora
```

13. **Dispare a aplicação Cliente**:

```
%>./client_calculadora <hostname> <porta>
```

14. **Problemas?** Revise os passos anteriores e corrija!

Yay! Se você chegou até aqui, well done. Mostre seu programa executando para o professor, e seu objetivo na aula de hoje terá sido cumprido.

Mas espere!!! Ainda há uma serie de aspectos interessantes que podem ser explorados utlizando aplicações sockets em C. Caso você ainda tenha tempo em aula, ou caso prefira fazer em casa, algumas atividades adicionais são sugeridas como exercício:

- Ao invés de realizar chamadas no seu servidor local, tente localizar e fazer chamadas no servidor do colega sentado ao lado.

- Experimente disparar mais de um processo cliente simultaneamente. Provavelmente eles executam muito rapidamente. Você pode introduzir algum *delay* artificial na execução das operações implementadas pelo servidor, ou implementar cálculos mais computacionalmente intensivos (e.g., calcular o n-ésimo termo da série de Fibonacci, etc). Como pode ser observado, o servidor tratará cada cliente iterativamente. Para que o servidor possa atender múltiplos clientes concorrentemente, faça com que a cada vez que o servidor receber uma operação, ele crie uma nova thread para computar o resultado e responder ao cliente. Responda:

    - As operações `sendto` e `recvfrom` precisarão ser protegidas por semáforos ou mutex para que possam ser acessadas por múltiplas threads?

    - Como o desempenho do servidor concorrente se compara ao desempenho do servidor iterativo?

- Altere a aplicação para que ela utilize sockets TCP ao invés de sockets UDP.