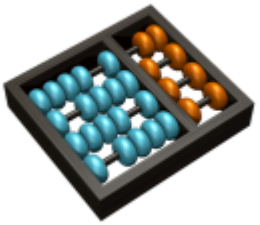


Instituto de Computação
Universidade Estadual de Campinas - Unicamp

Exercício 2

Alunos: Carlos Eduardo da Silva Santos e Felipe Correia Labbate
RA: 195396 e 196699

Campinas, 20 de Setembro de 2022

Sumário

1. Analise os códigos dos programas **cliente.c** e **servidor.c** e explique as funções usadas para comunicação via socket. Procure nas páginas de manual do Linux, a descrição das funções relacionadas ao uso de sockets.

Funções fundamentais para comunicação via socket:

- socket (int *domain*, int *type*, int *protocol*): cria um endpoint para comunicação e retorna um arquivo descritor que se refere a esse endpoint.
- bind (int *sockfd*, const struct sockaddr **addr*, socklen_t *addrlen*): vincula um nome a um socket. Quando um socket é criado através da função *socket*, ele existe em um espaço de nomes (família de endereços), mas não tem um endereço atribuído a ele. A função *bind* atribui ao endereço especificado por *addr* ao socket referido pelo descritor de arquivo *sockfd*. *addrlen* especifica o tamanho em bytes da estrutura de endereço apontada por *addr*.
- listen (int *sockfd*, int *backlog*): escuta conexões em um socket. A função marca o socket referido por *sockfd* como um socket passivo, ou seja, como um socket que será usado para receber solicitações de conexão usando a função *accept*.
- accept (int *sockfd*, struct sockaddr *restrict *addr*, socklen_t *restrict *addrlen*): aceita uma conexão em um socket. Extrai a primeira solicitação de conexão da fila de conexões pendentes para o socket passado como argumento (*listening* socket, socket passivo), cria um novo socket conectado, e retorna um novo descritor de arquivo referente a esse socket. O novo socket criado não está no estado *listening*, e o socket original não é afetado pela função.
- write (int *fd*, const void **buf*, size_t *count*): escreve em um descritor de arquivo. Escreve até *count* bytes do buffer, começando em *buf*, ao arquivo referido pelo descritor de arquivo *fd*.
- close (int *fd*): fecha um descritor de arquivo, de maneira que ele não se refere a nenhum arquivo, podendo ser reutilizado.
- connect (int *sockfd*, const struct sockaddr **addr*, socklen_t *addrlen*): inicia uma conexão em um socket. Conecta o socket referido pelo descritor de arquivo *sockfd* para o endereço especificado por *addr*. O parâmetro *addrlen* especifica o tamanho do parâmetro *addr*.
- read (int *fd*, void **buf*, size_t *count*): lê de um descritor de arquivo. Tenta ler até *count* bytes do descritor de arquivo *fd* no buffer começando em *buf*.

Funções úteis para comunicação via socket:

- bzero: limpa dados da memória.
- htonl: função de conversão.
- htons: função de conversão.
- inet_pton: função de conversão. Converte endereços IPv4 e IPv6 de texto para formato binário.

2. Faça um diagrama de sequência das chamadas das funções identificadas de soquete e fluxo de dados entre o cliente e o servidor.

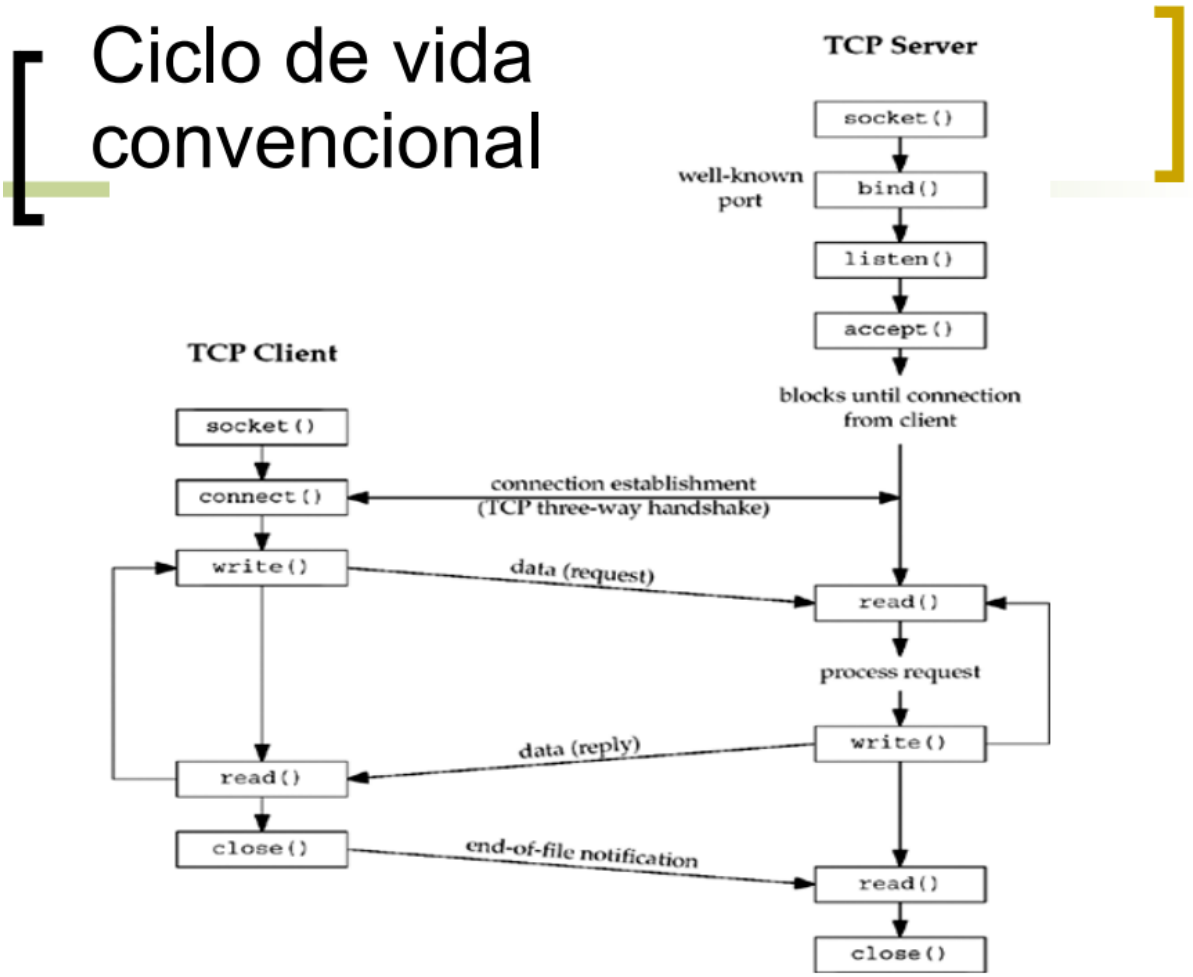
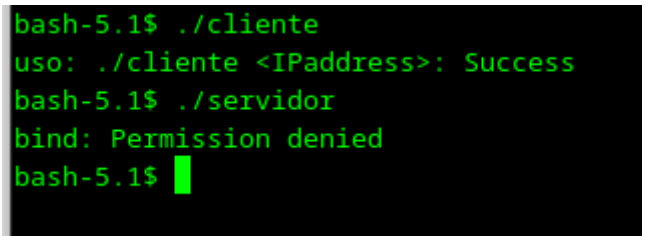


Diagrama apresentado em aula das chamadas de função realizadas e fluxo de dados em uma conexão TCP entre cliente e servidor, sem concorrência (convencional)

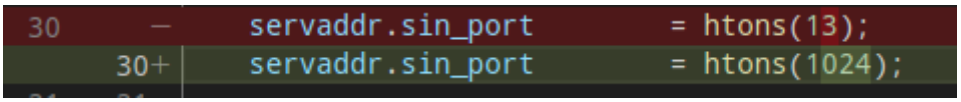
3. Houve algum erro com a função Bind? Em caso afirmativo, qual a sua causa? Se necessário, modifique os programas de forma que este erro seja corrigido e informe quais modificações foram realizadas.

Sim. Erro: “**bind: Permission denied**”, no arquivo **servidor.c**.



Mensagem de erro referente à função *bind*()

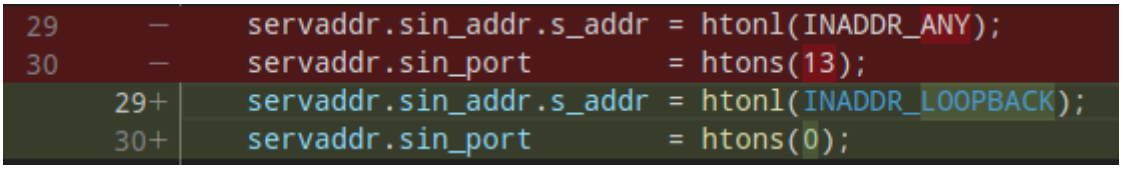
A causa é o número da porta (“servaddr.sin_port = htons(13);”, 13 no caso) originalmente usada. Usar portas abaixo da 1024 requer acesso root, por isso recebemos o erro “bind: Permission denied”.
Para arrumar o erro é necessário alterar o valor da porta usada (“servaddr.sin_port”) para htons(0), onde dessa maneira a própria máquina irá atribuir um valor de porta adequado. Outra opção é alterar o valor da porta para 1024 ou maior, onde no caso será usado o valor da porta indicado.



Modificação feita de forma que o servidor funcione

4. Altere o código do servidor para ser automatizada a escolha da porta e utilize sempre o IP local da máquina que está sendo executado.

- Automatizar a escolha da porta: é necessário alterar o valor da porta usada (“servaddr.sin_port”) para htons(0), dessa maneira ele utilizará a primeira porta livre disponível.
- Sempre utilizar o IP local da máquina: é necessário alterar a definição do IP (servaddr.sin_addr.s_addr) de INADDR_ANY para INADDR_LOOPBACK (127.0.0.1), que faz referência ao *localhost* (IP da máquina que está sendo executado).



Modificações feitas em servidor.c

5. Modifique o programa **cliente.c** para que ele obtenha as informações do socket local (# IP, # porta local) através da função getsockname().

Depois da chamada de *connect* utilizamos a função *getsockname* e obtemos as informações do socket local, conforme código, já modificado, abaixo.

```c
if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    exit(1);
}

servaddr_len = sizeof(servaddr);
if (getsockname(sockfd, (struct sockaddr*)&servaddr, &servaddr_len) < 0) {
    perror("getsockname error");
    exit(1);
}
printf("Local socket is %s:%d\n", inet_ntoa(servaddr.sin_addr), ntohs(servaddr.sin_port));
```

cliente.c modificado

6. Modifique o programa **servidor.c** para que este obtenha as informações do socket remoto do cliente (# IP remoto, # porta remota), utilizando a função getpeername(). Imprima esses valores na saída padrão.

Depois da chamada de *accept* utilizamos a função *getpeername* e obtemos as informações do socket remoto, conforme código, já modificado, abaixo.

```c
if ((connfd = accept(listenfd, (struct sockaddr *)NULL, NULL)) == -1)
{
    perror("accept");
    exit(1);
}

if (getpeername(connfd, (struct sockaddr*)&servaddr, &servaddr_len) < 0) {
    perror("getpeername error");
    exit(1);
}
printf("Received connection from %s:%d\n", inet_ntoa(servaddr.sin_addr), ntohs(servaddr.sin_port));
```

servidor.c modificado

7. Modifique o programa **cliente.c** para que ele capture uma mensagem do stdin, e envie essa mensagem para o servidor. Faça com que o programa **servidor.c** receba e imprima essa mensagem na saída padrão.

Utilizando as funções *send* e *recv* é possível enviar mensagens entre o cliente e o servidor. Foi necessário alterar a função *if(n < 0)* no cliente.c para encerrar a conexão também no cliente quando *n < 0,* já que na versão original do programa, ele exibe o erro ***read error: Connection reset by peer*** ao final da execução do cliente. Isso acontece pois era encerrada a conexão no servidor, mas não era encerrada no cliente.

```
while ((n = read(sockfd, recvline, MAXLINE)) > 0) {
    recvline[n] = 0;
    if (fputs(recvline, stdout) == EOF) {
        perror("fputs error");
        exit(1);
    }

    printf("Write a message to send to server: ");
    scanf("%s", message);
    send(sockfd, message, sizeof(message), 0);
}

if (n < 0) {
    close(sockfd);
}

exit(0);
```

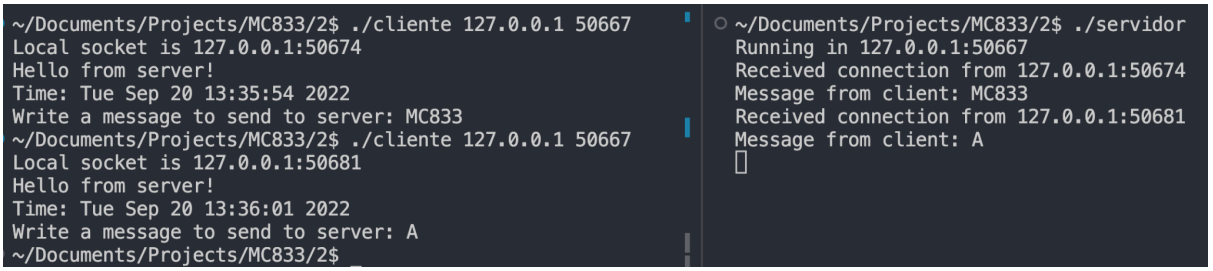
cliente.c modificado

```c
if(recv(connfd, message, sizeof(message), 0) < 0)
{
    perror("recv error");
    exit(1);
}
printf("Message from client: %s\n", message);

close(connfd);
```

servidor.c modificado

Execução dos programas cliente (lado esquerdo) e servidor (lado direito):



Execução do cliente e servidor modificados, lado a lado

8. Existem outras funções para escrever e ler na programação de sockets? Qual a diferença entre essas funções e as usadas nos programas estudados? Indique o uso dessas funções e mostre como podem ser usadas.

Sim. Para enviar existem as funções *send* e *write*, e para receber existem as funções *read* e *recv.*
Send e *recv* possuem o parâmetro do tipo inteiro chamado *flags*. O parâmetro *flags* é a operação lógica OR, bit a bit, de zero ou mais das seguintes flags:

- send: MSG_CONFIRM, MSG_DONTROUTE, MSG_DONTWAIT, MSG_EOR, MSG_MORE, MSG_NOSIGNAL, MSG_OOB
- recv: MSG_DONTWAIT, MSG_ERRQUEUE, MSG_OOB, MSG_PEEK, MSG_TRUNC, MSG_WAITALL

As funções *write* e *read* não possuem o parâmetro *flags*.
Na função *send*, quando o argumento *flags* é igual a zero, seu comportamento é equivalente ao da função *write*.
Na função *recv*, quando o argumento *flags* é igual a zero, seu comportamento é geralmente semelhante ao da função *read*. Existe diferença de comportamento quando um datagrama de comprimento zero estiver pendente. Nesse caso, *read* não tem efeito (o datagrama permanece pendente), enquanto que *recv* consome o datagrama pendente.