MAC0422 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 7 de Dezembro de 2020

Roteiro

Entrando em mais detalhes de sockets TCP

Servidores concorrentes

Entrando em mais detalhes de sockets TCP

Servidores concorrentes

Entrando em mais detalhes de sockets TCP

Servidores concorrentes

Entrando em mais detalhes de sockets TCP

Máquina de estados do TCP

- Para o seu correto funcionamento o protocolo TCP segue uma máquina de estados
- Cada lado da conexão tem que estar no estado correto para que o protocolo consiga entregar todas as suas garantias
- Alguns estados mais relevantes:
  - CLOSED
  - LISTEN
  - ESTABLISHED
  - TIME_WAIT

Habilitando a aceitação de conexão

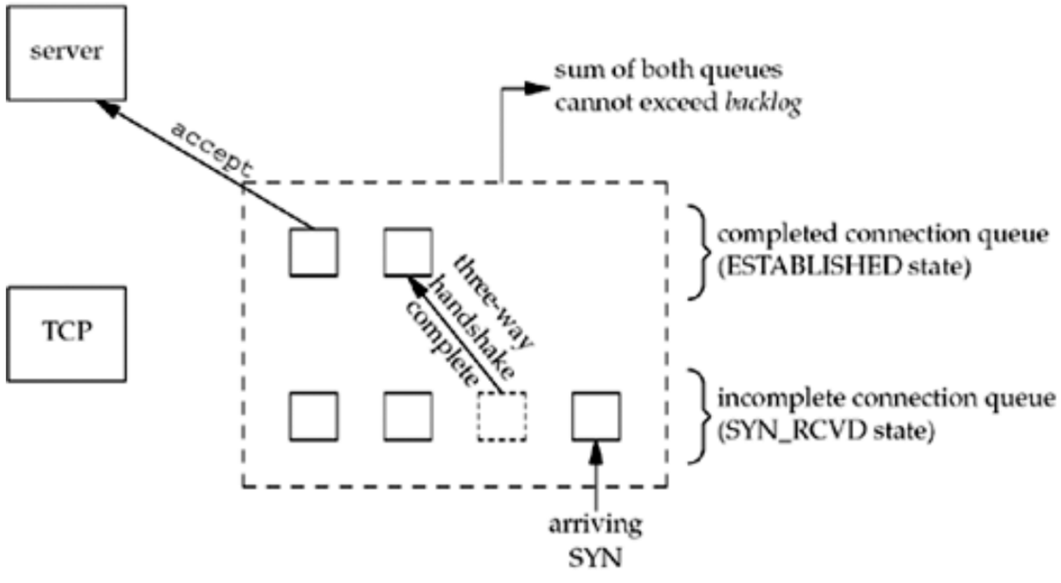
- Por padrão uma máquina não aceita conexões nas suas portas (nem TCP nem UDP)
- É necessário transformar um socket ativo em um socket passivo e isso é feito com a função `listen`
- A função `listen` muda o estado do socket de CLOSED para LISTEN

Procedimento de aceitação de conexão

- O TCP é um protocolo com diversas garantias
  - Entrega confiável
  - Entrega ordenada
  - Uso da rede de forma controlada (Controle de congestionamento)
  - Uso dos hosts de forma controlada (Controle de fluxo)
- O TCP é um protocolo full-duplex
- Para tudo funcionar corretamente diversas variáveis precisam ser inicializadas
- O início da conexão serve para fazer todos os ajustes nas variáveis

3 Way-Handshake

- Quando o `accept` é executado o processo servidor é colocado para dormir (`connect` do lado do cliente é quem inicia o 3WHS)
- O SO é responsável por cuidar do 3 WHS
- Enquanto o socket está fazendo o 3 WHS ele fica em um buffer com tamanho definido pelo parâmetro da função `listen` onde há duas filas: conexões incompletas e conexões completas (o parâmetro do listen define o tamanho da soma das duas filas)

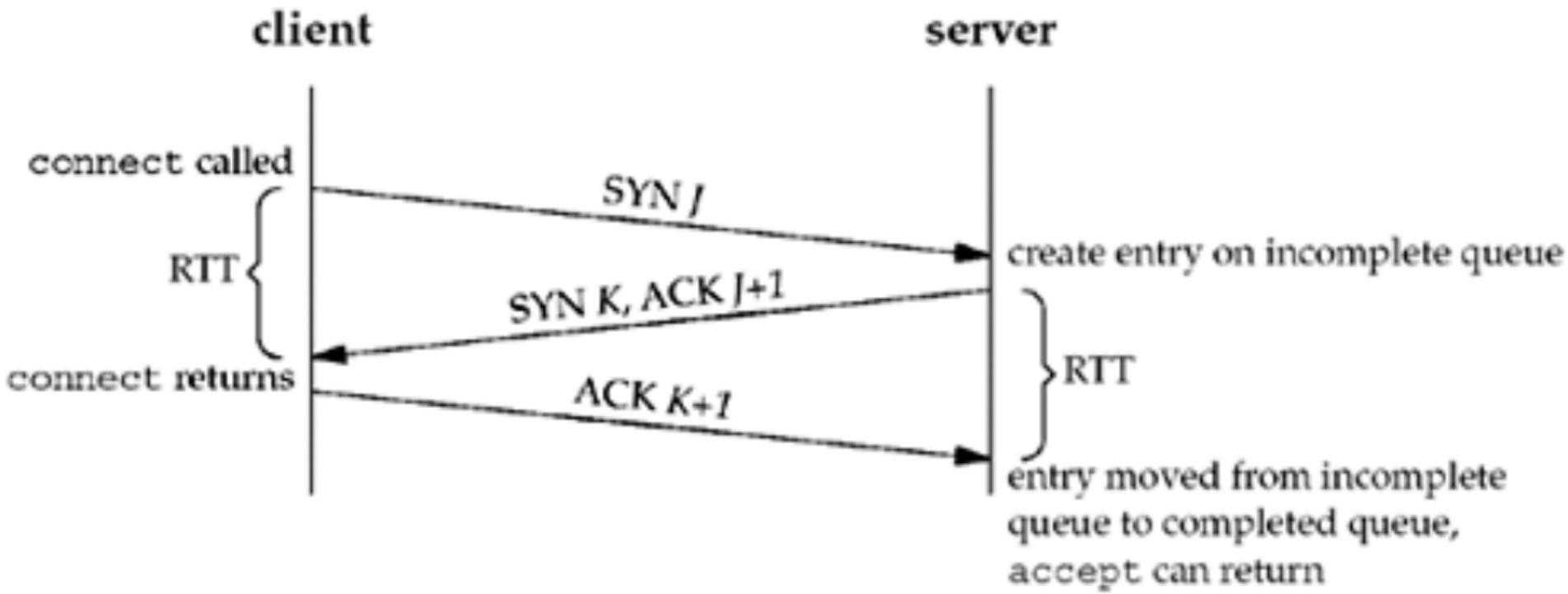


3 Way-Handshake

Entrando em mais detalhes de sockets TCP

Servidores concorrentes

- Quando uma conexão termina o 3WHS, o `accept` acorda e tira a primeira conexão estabelecida da fila de conexões completas

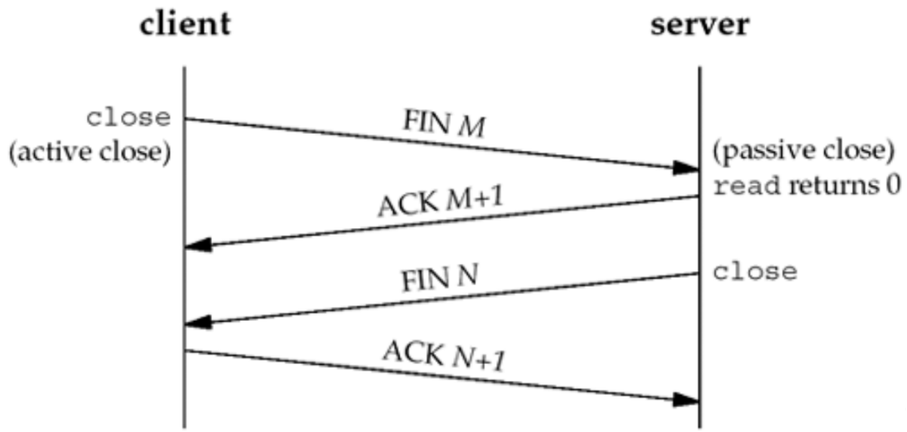


3 Way-Handshake

- Há um limite de tempo que o cliente espera para ter uma primeira resposta do servidor
- Se não há espaço para a conexão do cliente no buffer definido pelo `listen`, esse tempo estoura e o cliente não consegue se conectar
- O parâmetro do `listen` na maioria dos SOs não define diretamente o tamanho do buffer
- Se o tempo estoura, a função `connect` retorna -1 e para verificar o erro é necessário verificar a variável global `errno` com a função `fileno`

Procedimento de finalização de conexão

- O TCP é full-duplex então para a conexão de fato ser finalizada, isso tem que ser feito nos dois sentidos
- O lado que inicia a finalização realiza o fechamento ativo. Isso é feito com um `close` no socket
- O lado que não iniciou o fechamento na grande maioria das vezes (tem como evitar que isso aconteça) vai fazer o mesmo do seu lado e nesse caso ele faz o fechamento passivo pois ele faz como resposta à ação do outro lado
- No lado de quem recebe o fechamento é como se fosse recebido um *End Of File* o que faz o `read` retornar



TIME_WAIT

- Uma observação interessante é que o lado que começou o fechamento da conexão mandou o último pacote que foi um pacote ACK. Qual a garantia de que esse pacote de fato chegou do outro lado?
- O máximo que pode ser feito é esperar para ter certeza que o ACK vai chegar do lado de lá por isso o lado que começou o fechamento fica no estado `TIME_WAIT` por um intervalo de tempo definido pelo SO

TIME_WAIT

Entrando em mais detalhes de sockets TCP

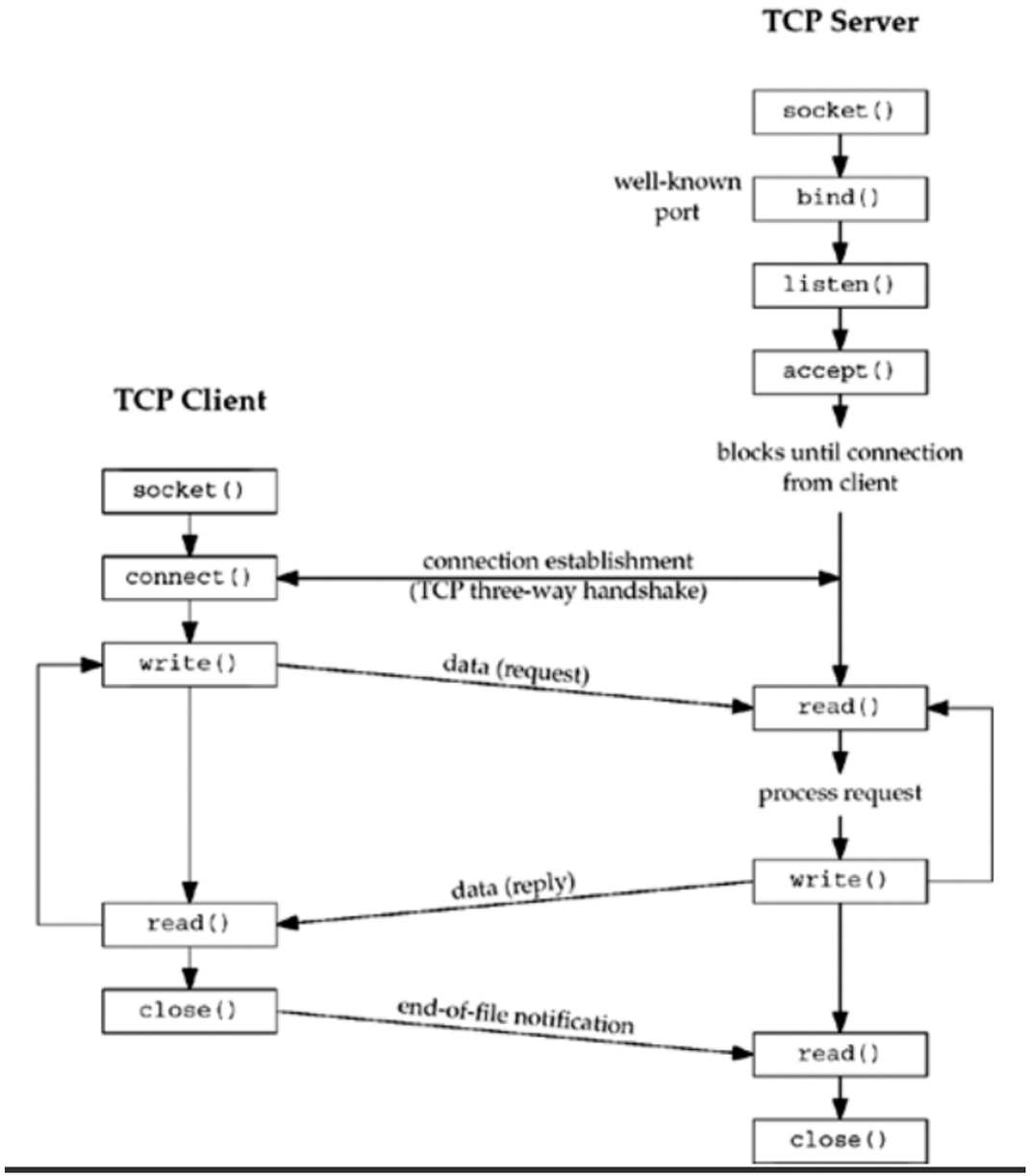
Servidores concorrentes

- Ficando no `TIME_WAIT`, se o ACK não tiver chegado, a outra ponta vai enviar o FIN de novo e é necessário que a conexão ainda exista para que outro ACK seja enviado
- Ficando no `TIME_WAIT` também é uma forma de esperar por pacotes atrasados chegarem. Se isso não acontecer, uma conexão nova pode acabar recebendo pacotes de uma conexão antiga!

O que temos até agora?

Entrando em mais detalhes de sockets TCP

Servidores concorrentes



Servidores concorrentes

Como está o servidor daytime?

- Ele só aceita uma conexão por vez (coloque um `sleep` antes do `close` no servidor e você vai perceber isso)
- No caso do servidor daytime isso não é um problema muito crítico porque as conexão são muito rápidas. Colocando um buffer no listen de tamanho adequado, os vários clientes conectando não vão notar lentidão
- O problema acontece quando as conexões tendem a demorar (um servidor web por exemplo) ou mesmo quando elas não são finalizadas tão cedo, como em um servidor de jogos ou um servidor SSH

Mas como ficam os números das portas?

- Sempre que uma conexão é estabelecida, o par: ip/porta de origem e ip/porta de destino definem uma conexão. Esses valores não pode ser iguais para duas conexões
- Se apenas uma conexão for realizada por computador não tem problema (ip vai ser diferente)
- Mas e se duas ou mais conexões forem realizadas por computador? O único jeito é ter uma porta diferente do lado do cliente e o SO cuida disso

Mas só temos um socket no servidor!

- É necessário criar um socket novo para cada cliente que chega
- Na realidade é necessário ter um código inteiro do servidor para cada cliente que chega pois cada cliente pode ter um comportamento completamente diferente do outro
- O que precisa ser feito de fato é ter um processo novo para cada cliente que chega, independente do anterior mas fazendo a mesma coisa (mesmo código)

Como criar cópias do servidor?

- usando `fork` :-)
- com o `fork` sendo chamado logo depois do accept é possível criar um novo processo para cada cliente e assim o servidor passa a ser concorrente, aceitando várias conexões