Redes 1

Prof. Ruan

Atividade 4

César Kallas RA: 02099224
cesarkallas@gmx.net
http://cesarkallas.soulivre.org

PARTE I – TCPdaytimed: um servidor orientado a conexão iterativo

1. **Estude o texto acima.**
   ok

2. **Estude o código fornecido e responda:**
   a) Para que serve a função ctime()?

   Converte o horário do sistema local para um string de caracteres.

   b) Por que o servidor utiliza dois identificadores de soquetes msock e ssock?

   O sevidor cria dois sockets para escutar conexões tcp de um cliente.
   O primeiro socket é criado normalmente, como passive e depois um segundo socket é criado para escutar conexões vinda do cliente. Assim que o servidor recebe uma conexão do cliente e a

3. **Modifique o código do servidor iterativo TCPdaytimed de modo que ele não feche explicitamente a conexão após enviar uma resposta. Ele continua funcionando corretamente? Explique.**

   Quando o socket de comunicação com o cliente não é fechado, o cliente que conectou e recebeu o "time" fica pendurado no servidor, esperando receber/comunicar algo mais. Assim, o cliente não libera o terminal no qual ele está sendo executado.
   Não fechar o socket com o cliente não prejudica o servidor, porque o socket que o cliente comunica com o servidor é "instanciado" (iniciado) quando o cliente conecta, e cada cliente que conecta inicia um novo socket (ssock / file descriptor). O socket que poderia prejudicar o servidor é o msock, e este não é usado diretamente com o cliente, ele é apenas um canal entre outro socket (ssock).

4. **Monte experimentos que permitam visualizar o número máximo de clientes que o servidor TCPdaytimed é capaz de atender até recusar serviço. Explique como foi realizado o experimento. Analise e procure explicar os resultados obtidos.**

   Esse teste foi feito de 3 maneiras diferentes.

   1. O TCPdaytime foi modificado (TCPdaytime_m) de modo a conectar várias vezes no servidor (sem concorrência) e sem fechar o socket. Dentro de um laço de repetição o cliente iniciava um socket de comunicação com o servidor e não o fechava.
      O resultado foi que o cliente conseguiu iniciar no máximo 1020 sockets com o servidor, esse número pôde ser verificado atráves de um contato no cliente.
      Read number 0000001020
      can't get "tcp" protocol entry

2. O TCPdaytime_m foi modificado para fechar o socket de pedido com o servidor, assim, o cliente abria uma conexão, fazia o pedido e quando recebe-se a resposta fechava o socket, isso dentro de uma repetição infinita, sem concorrência.
   O resultado foi que o cliente consegue abrir e fechar conexão com o servidor sem nenhum problema, esse teste ficou rodando durante um bom tempo (> 10m) sem parar. Não houve problemas.

3. O TCPdaytime_m foi modificado (TCPdaytime_t) de modo que ele conecte inúmeras (10000) vezes no servidor, com concorrência através de threads.
   O resultado foi normal até +- a thread 1520, após isso o sistema mata o processo. Do lado do servidor foi normal, não teve problemas.

Vale ressaltar que o socket é um file descriptor e que o linux tem um número máximo de file descriptor, que pode ser alterado em tempo de execução.

PARTE II – TCPechod: um servidor orientado a conexão concorrente

O arquivo TCPechod.c contém o código de um servidor concorrente para o serviço ECHO, baseado no algoritmo 4, apresentado no texto Projeto e implementação de servidores.

5. **Por que a instancia filha do servidor TCPechod.c, criada para atender a uma nova requisição, fecha o soquete msock mas não o soquete ssock? Quais seriam as conseqüências de deixar o msock aberto?**

   O processo filho fecha o socket msock, porque o msock é usado para receber o pedido de conexão do cliente, a partir do momento que o cliente conecta, a troca de dados será no socket ssock, por isso que o filho não pode fechá-lo, pois não teria como responder para o cliente.

   O socket msock que o filho fecha é herdado do seu pai, a partir do momento do fork, o cliente o herda e como não irá usa-lo (até porque seu pai está o usando a porta de escuta), o cliente o fecha. Se o filho não o usar, irá ocupar memória sem necessidade (desde que ele também não coloque-o em modo de escuta).

6. **Explique o que aconteceria se a chamada (void) signal(SIGCHLD,reaper) for retirada do servidor TCPechod.c.**

   Todos os processos filhos do servidor ficarão rodando (bloqueados) no sistema, mesmo depois de fazerem o servidor, ocupando recursos do SO. Chega um momento que o SO bloquea o processo pai por falta de recursos para criar filhos:

   fork: Resource temporarily unavailable

7. **Por que na função reaper() a primitiva wait3() é chamada com o argumento WNOHANG?**

   Serve para retornar imediatamente se nenhum filho estiver finalizado, ou seja, se o filho ainda estiver ativo não o fique esperando terminar.

8. **Monte experimentos que permitam visualizar o número máximo de clientes que o servidor TCPechod é capaz de atender até recusar serviço. Explique como foi realizado o experimento. Analise e procure explicar os resultados obtidos.**

   O cliente TCPecho.c foi modificado de modo a executar concorrentemente 1000 threads com um write de um buffer pré estabelecido. O cliente conecta no servidor, escreve o buffer, lê a resposta e sai. Isso é feito para todas as threads.

   O resultado foi que o sistema chega em um determinado momento e mata o processo cliente, por falta de recursos para criar mais threads.

   Enquanto o SO não mata o cliente, o cliente com suas threads consegue utilizar perfeitamente o servidor.

Referência

COMER, D. E., STEVENS, D. L., Internetworking With TCP/IP Volume III: Client-Server Programming and Applications, Linux/POSIX Socket Version, Prentice-Hall International 2001, Capítulo 7.

STEVENS,W.R.; "UNIX NETWORK PROGRAMMING - Networking APIs: Sockets and XTI" - Volume 1 - Second Edition - Prentice Hall – 1998

MAN do Linux

Anexo a

Descrição do ambiente em que os testes e as coletas de dados foram feitas, incluindo local e o sistema operacional usado.

Local: casa

Ambiente 1:

Linux opensrc 2.6.8.1 #1 Sat Sep 18 19:09:26 PDT 2004 i686 unknown unknown GNU/Linux

```
model name      : Celeron (Mendocino)
stepping        : 0
cpu MHz         : 300.769
cache size      : 128 KB
```

```
Address                  HWtype  HWaddress           Flags Mask            Iface
cesarkallas.opensrc      ether   00:0B:CD:EC:58:84   C                     eth0
c90c8001.cps.virtua.com  ether   00:00:77:94:0F:D3   C                     eth1
```

Ambiente 2:

Linux cesarkallas 2.6.11.4-21.9-default #1 Fri Aug 19 11:58:59 UTC 2005 i686 athlon i386 GNU/Linux

```
model name      : mobile AMD Athlon(tm) XP2500+
cpu MHz         : 530.125
cache size      : 512 KB
```

```
Address                  HWtype  HWaddress           Flags Mask            Iface
opensrc                  ether   00:C0:DF:F4:C0:AE   C                     eth0
```

Anexo B

ok

Anexo c

ok