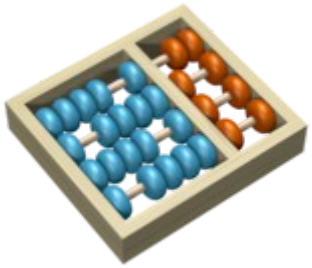


Exercício 3: backlog e processos zumbis

Aluna: Naomi Takemoto

RA: 184849

Instituto de Computação

Universidade Estadual de Campinas

Novembro de 2020

Instruções para execução do programa

Para o exercicio 3 e 4, será necessário executar o script zombies.sh

./zombies.sh

ou

bash zombies.sh

- No exercício 3, deve-se alterar no código do servidor o valor da macro **BACKLOG** para cada valor desejado e depois rodar o script mencionado acima. O script chamará o make file então não é necessário chamar o comando make. O resultado será salvo no arquivo **zombies_output.txt**
- Para limpar os executáveis:
  - **make clean**

Demais instruções não essenciais estão no **README** junto ao código.

Excercício 1

Com o conhecimento adquirido em aula explique qual a relação entre backlog e número de conexões.

Em tese, o backlog seria a soma do número de conexões em duas filas:

- A fila de conexões incompletas, aquelas que não terminaram o processo de 3-Way Handshake. (3WH)
- A file de conexões completas, que terminaram o 3WH

No entanto, a relação do backlog com o número de conexões depende do sistema operacional, conforme mostra a tabela:

Figure 4.10. Actual number of queued connections for values of *backlog*.

backlog	Maximum actual number of queued connections				
	MacOS 10.2.6 AIX 5.1	Linux 2.4.7	HP-UX 11.11	FreeBSD 4.8 FreeBSD 5.1	Solaris 2.9
0	1	3	1	1	1
1	2	4	1	2	2
2	4	5	3	3	4
3	5	6	4	4	5
4	7	7	6	5	6
5	8	8	7	6	8
6	10	9	9	7	10
7	11	10	10	8	11
8	13	11	12	9	13
9	14	12	13	10	14
10	16	13	15	11	16
11	17	14	16	12	17
12	19	15	18	13	19
13	20	16	19	14	20
14	22	17	21	15	22

Figura reproduzida do livro Unix Networking Programming, Volume 1 – The Sockets Networking API (3rd Edition).

Exercício 2

Pesquise como está implementado o backlog de um socket TCP no kernel linux (para versão 2.2 ou mais recentes), e em seguida indique o valor padrão do backlog. Comprove sua resposta através de figuras no seu relatório.

Segundo https://www.edufukunari.com.br/how-tcp-backlog-works-in-linux/

A partir da versão 2.2 do kernel do Linux, o backlog passou a se referir ao número máximo de conexões completas. Ou seja o tamanho da fila de conexões que terminaram o processo de 3WHS.

No entanto, segundo a man page do comando listen, transcrito abaixo, o backlog se refere ao número de conexões incompletas:

"The backlog argument defines the maximum length to which the queue of

pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED or, if the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connection succeeds."

Essa informação foi obtida para o kernel 5.4 (comando uname -r), do sistema operacional Ubuntu 20.04.1 LTS. Segundo, outra fonte https://levelup.gitconnected.com/linux-kernel-tuning-for-high-performance-networking-high-volume-incoming-connections-196e863d458a

é provável que a man page esteja desatualizada e, segundo ela, o número máximo de conexões pendentes é setado na seguinte variável do kernel:

net.ipv4.tcp_max_syn_backlog. Assim o backlog se refereria ao número de conexões completas.

Nessa versão existem duas filas, uma para conexões pendentes (SYN recebido e SYN + ACK enviado pelo servidor para o cliente) e outra para pendendes, para as quais o processo de 3WHS terminou (ACK do cliente foi recebido e processado).

Exercício 3

Realize experimentos a fim de verificar quantos clientes conseguem de imediato conectar-se ao servidor no passo anterior, comece com backlog em 0. Indique e comprove a partir de qual valor de backlog as conexões não ocorrem imediatamente. Elabore um esquema para tentar conectar 10 clientes de forma simultânea (Veja as dicas logo abaixo).

Dica 1: escreva scripts que executam várias instâncias do cliente em um pequeno intervalo de tempo ou utilize algum programa que permita o controle de vários terminais simultaneamente, como o cssh.

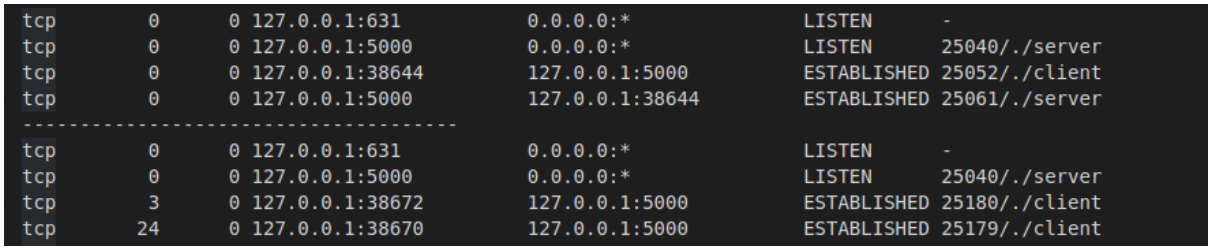
Dica 2: utilize o netstat para descobrir o número de clientes que conseguem realizar o 3WHS de imediato e estabelecer a conexão (basta contar as linhas da saída do netstat correspondentes a conexões ao servidor que estejam em um estado que comprove a finalização do 3WHS -- lembre-se do diagrama de estados do TCP).

Para este exercício foi criado o arquivo zombies.sh que

1. Inicia um servidor e o coloca em backgound
2. Inicia o netstat logo após a criação do servidor, salvando o resultado em um arquivo.
3. Instancia 10 clientes rapidamente em um loop
4. Chama novamente o netstat, salvando o resultado em arquivo.

Os arquivos gerados nos experimentos estão na pasta **echo/backlog**. Os arquivos da pasta são nomeados zombie_output<#>.txt, onde o <#> indica o valor de backlog utilizado para realizar o experimento.

Um trecho do output para o backlog em 0 pode ser visto na imagem a seguir (considerou-se somente a saída do primeiro netstat – até a separação com traços).



Comparando os números de conexões estabelecidas com o valor esperado segundo a tabela mencionada no exercício 1, para o sistema linux:

Backlog	**Número de conexões estabelecidas**	**Esperado**
0	2	3
1	0	4

2	4	5
3	5	6
4	6	7
5	8	8

Observa-se que o experimento acima mostrou que o número de conexões estabelecidas não ultrapassou o valor esperado (obtido a partir da tabela do exercício 1) para nenhum dos experimentos executados que evidencia a corretude dos dados mostrados na tabela do livro texto.

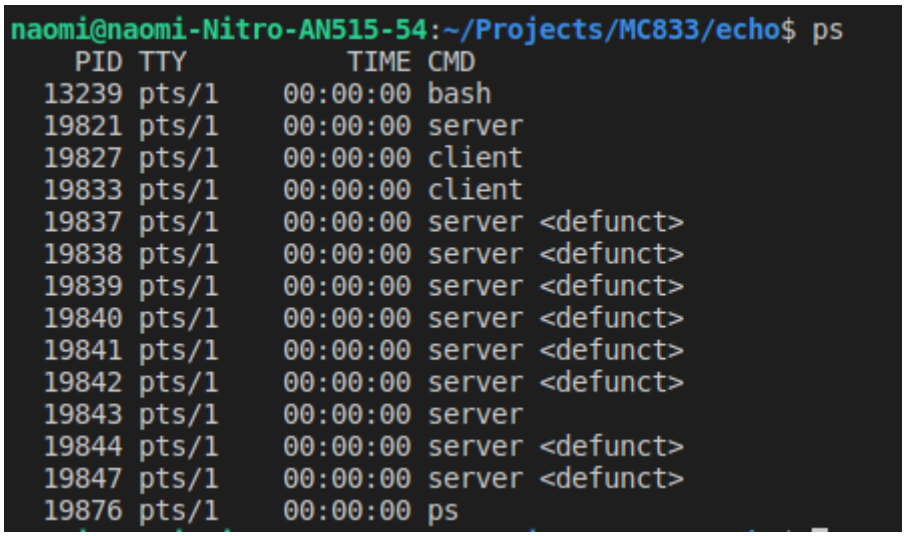
Exercício 4

É correto afirmar que o código na versão atual gera processo zumbi? Explique. Se a sua resposta foi sim, então altere o código da questão 3 de modo que os processos criados pelo fork sejam corretamente finalizados ao invés de permanecerem no estado zumbi quando um cliente encerra sua conexão.

Dica: Consulte o livro texto da disciplina para verificar como resolver este problema.

Quando um processo filho termina sua execução, um sinal SIGCHLD é enviado para o proceso pai que deve ler essa chamada de saída, retirando o filho da tabela de processos. Se isso não ocorrer então os filhos que terminaram a execução, continuam nessa tabela se tornando “processos zumbis”.

No estado atual do programa isso ocorre, já que o sinal SIGCHLD não está sendo tratado propriamente. Ver a figura a seguir:



Para tratar esse problema, implementou-se um signal handler para SIGCHLD, conforme explicado no livro-texto da disciplina.

```c
// Signal handler to avoid Zombies
void sig_chld(int signo)
{
  pid_t pid;
  int stat;
  pid = wait(&stat);
  printf("child %d terminated\n", pid);
  return;warning: implicit declaration of function ‘Signal
}
```

O resultado da execução mostra o que o handler foi de fato chamado e ao final não foram mais criados processos zumbis:

```
----------------------------------------
child 21874 terminated
child 21875 terminated
child 21873 terminated
child 21877 terminated
child 21879 terminated
child 21882 terminated
child 21883 terminated
child 21876 terminated
child 21881 terminated
naomi@naomi-Nitro-AN515-54:~/Projects/MC833/echo$ ps
    PID TTY          TIME CMD
  21842 pts/1    00:00:00 bash
  21858 pts/1    00:00:00 server
  21865 pts/1    00:00:00 client
  21880 pts/1    00:00:00 server
  22280 pts/1    00:00:00 ps
```