



Exercício 3

Aluno: Alan Freitas Ribeiro

RA: 193400

Instituto de Computação
Universidade Estadual de Campinas

Campinas, 01 de Novembro de 2022.

Sumário

- 1) Questão 1 ... 3
- 2) Questão 2 ... 3
- 3) Questão 3 ... 5
- 4) Questão 4 ... 6
- 5) Questão 5 ... 8

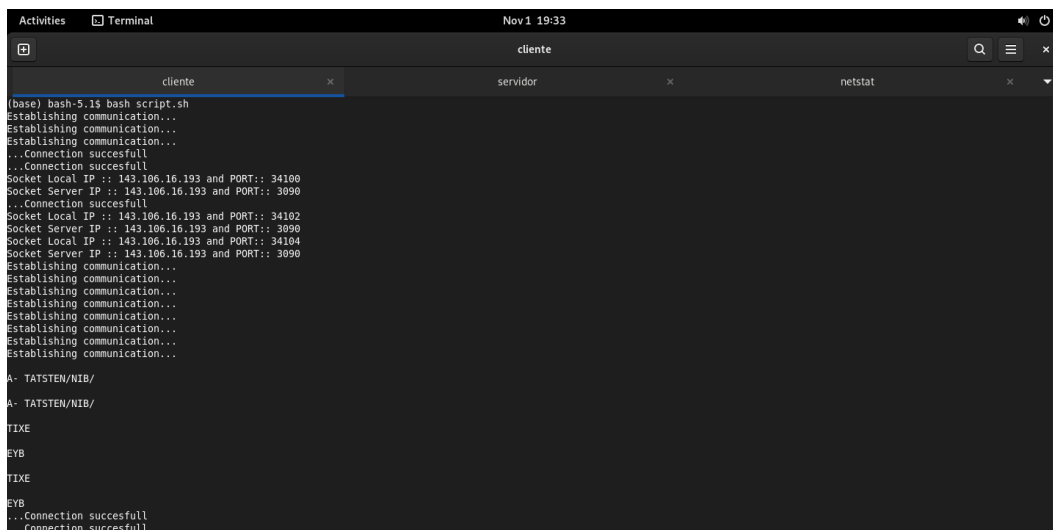
1 – Com o conhecimento adquirido em aula explique qual a relação entre backlog e número de conexões e explique o funcionamento das duas filas mantidas durante o TCP 3WHS.

O backlog é responsável por determinar o número de conexões pendentes que a fila irá manter, sendo que, quando temos múltiplas conexões, o servidor irá processar essas conexões na fila, de modo que o tamanho do backlog irá determinar quantas conexões ativas + conexões incompletas poderão estar na fila. Durante o TCP 3WHS, temos duas filas, uma fila para conexões incompletas (fila SYN) e fila das conexões já completas, conexões com o estado "SYN RECEIVED" são adicionadas para a fila de conexões incompletas e depois que seu estado é mudado para "ESTABLISHED" elas são movidas para a fila de conexões já completas, onde a função "accept" irá utilizá-las.

2 – Com o objetivo de identificar o uso do backlog e sua relação com o número de conexões, faça com que o servidor do exercício anterior receba o valor do backlog como parâmetro que será passado para a função listen. Crie um script que permita verificar quantos clientes conseguem de imediato conectar-se ao servidor (conexões em estado

ESTABLISHED) para valores de backlog desde 0 até 10. Elabore um esquema para tentar conectar 10 clientes de forma simultânea e faça uma tabela indicando os resultados observados em relação ao backlog usado e o número de conexões estabelecidas.

Figura 1 – Execução do programa com parâmetro do backlog em 1



```
(base) bash-5.1$ bash script.sh
Establishing communication...
Establishing communication...
Establishing communication...
...Connection successful
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34100
Socket Server IP :: 143.106.16.193 and PORT:: 3090
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34102
Socket Server IP :: 143.106.16.193 and PORT:: 3090
Socket Local IP :: 143.106.16.193 and PORT:: 34104
Socket Server IP :: 143.106.16.193 and PORT:: 3090
Establishing communication...
Establishing communication...
Establishing communication...
Establishing communication...
Establishing communication...
Establishing communication...
Establishing communication...
A- TATSTEN/NIB/
A- TATSTEN/NIB/
TIXE
EYB
TIXE
EYB
...Connection successful
...Connection successful
```

Figura 2 – Execução do programa com parâmetro do backlog em 10

```
(base) bash-5.1$ bash script.sh
Establishing communication...
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34124
Socket Server IP :: 143.106.16.193 and PORT:: 3090
...Connection successful
Establishing communication...
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34126
Socket Server IP :: 143.106.16.193 and PORT:: 3090
Socket Local IP :: 143.106.16.193 and PORT:: 34122
Socket Server IP :: 143.106.16.193 and PORT:: 3090
Establishing communication...
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34128
Socket Server IP :: 143.106.16.193 and PORT:: 3090
Establishing communication...
Establishing communication...
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34130
Socket Server IP :: 143.106.16.193 and PORT:: 3090
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34132
Socket Server IP :: 143.106.16.193 and PORT:: 3090
Establishing communication...
Establishing communication...
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34134
Socket Server IP :: 143.106.16.193 and PORT:: 3090
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34136
Socket Server IP :: 143.106.16.193 and PORT:: 3090
Establishing communication...
...Connection successful
Socket Local IP :: 143.106.16.193 and PORT:: 34138
Socket Server IP :: 143.106.16.193 and PORT:: 3090
```

Como podemos perceber pela figura 1 e 2, quando o parâmetro do backlog é setado para 1, conseguimos apenas 3 conexões por vez, de modo que quando aumentamos o parâmetro para 10, todas as conexões foram estabelecidas de vez, houve uma demora entre as conexões pois colocamos um sleep de 10 segundos na primeira linha do laço do servidor. Variamos os parâmetros entre 1 e 11 e o resultado encontrado condiz com a tabela mostrada em aula. Para executar todas as conexões ao mesmo tempo, criamos um script simples no arquivo “script.sh”.

3 – Explique o quê é um processo zumbi, quando eles são gerados em conexões de sockets TCP com servidor concorrente e porquê devem ser tratados corretamente.

Um processo zumbi, é um processo em que sua execução já foi finalizada mas ainda está registrado na tabela de processos do sistema esperando pelo processo pai removê-lo de lá. Eles são gerados quando o processo pai é clonado, gerando um processo filho, e o processo pai não espera pelo fim do processo filho e continua executando suas próximas tarefas, deste modo, quando o processo filho termina e envia o sinal de término, o processo pai não lê esse sinal e não remove o processo filho da tabela de processos, deixando assim, o processo filho em estado zumbi permanentemente. Eles devem ser tratados corretamente pois, apesar de serem "dummys" e não gastar muito recurso do sistema, eles continuam ocupando espaço na tabela de processos que é finita, ou seja, se muitos processos forem gerados e não terminados, pode ser que a tabela fique cheia e então o sistema não consiga mais criar novos processos.

4 – Explique e mostre se alguns processos zumbi são gerados pela versão atual do seu código. Altere o código para que os processos criados pelo fork sejam corretamente finalizados ao invés de permanecerem no estado zumbi quando um cliente encerra sua conexão. Indique as mudanças feitas no código e explique o como essas alterações ajudam no tratamento desses processos.

Figura 3 – Execução do programa antes do tratamento correto dos processos

```
Activities Terminal Nov 1 19:49 netstat
cliente x servidor netstat
(base) bash-5.1$ ps -fe | grep 3090
ra193400 17075 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17078 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17084 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17085 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17086 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17088 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17089 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17090 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17091 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17092 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17093 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17094 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17095 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17096 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17097 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17098 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17100 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17101 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17102 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17104 9594 0 19:17 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17502 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17512 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17513 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17516 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17517 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17518 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17521 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17523 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17526 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17527 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17528 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17530 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17531 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 17532 9594 0 19:19 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 20036 9594 0 19:30 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 20039 9594 0 19:30 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 20040 9594 0 19:30 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 20041 9594 0 19:30 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 22006 9594 0 19:41 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 22007 9594 0 19:41 pts/0 00:00:00 ./cliente 143.106.16.193 3090
ra193400 23314 15261 0 19:49 pts/2 00:00:00 grep 3090
```

Figura 4 – Execução do programa depois do tratamento correto dos processos

```
Activities Terminal Nov 1 20:43 netstat
servidor x cliente netstat
(base) bash-5.1$ ps -fe | grep 3090
ra193400 33591 33059 0 20:41 pts/0 00:00:00 ./servidor 3090
ra193400 34013 33091 0 20:43 pts/2 00:00:00 grep 3090
```

```
Activities Terminal Nov 1 20:43 servidor
servidor x cliente netstat
(base) bash-5.1$ ./servidor 3090
Server started with IP :: 143.106.16.193 and PORT:: 3090
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34204 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34202 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34216 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34214 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34208 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34224 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34218 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34220 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34206 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34212 at time :: Tue Nov 1 20:42:19 2022
CONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34222 at time :: Tue Nov 1 20:42:19 2022
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34216 at time :: Tue Nov 1 20:42:19 2022
child 33697 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34206 at time :: Tue Nov 1 20:42:19 2022
child 33693 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34222 at time :: Tue Nov 1 20:42:19 2022
child 33700 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34218 at time :: Tue Nov 1 20:42:19 2022
child 33698 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34212 at time :: Tue Nov 1 20:42:19 2022
child 33695 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34224 at time :: Tue Nov 1 20:42:19 2022
child 33701 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34220 at time :: Tue Nov 1 20:42:19 2022
child 33699 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34204 at time :: Tue Nov 1 20:42:19 2022
child 33689 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34202 at time :: Tue Nov 1 20:42:19 2022
child 33690 terminated
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34208 at time :: Tue Nov 1 20:42:19 2022
DISCONNECTED -> Remote Socket from client IP :: 143.106.16.193 and PORT:: 34214 at time :: Tue Nov 1 20:42:19 2022
child 33696 terminated
child 33694 terminated
```

Antes do tratamento correto, nosso programa chamava o `fork` mas não tinha nenhum mecanismo para que o processo pai esperasse o sinal de encerramento dos processos filhos, gerando assim, os processos zumbis. Para tratar esse problema, criamos a função `void sig_chld(int signal)` que chama `waitpid(-1, &stat, WNOHANG)` e a função `signal(SIGCHLD, sig_chld)` para que o processo pai espere pelo encerramento dos processos filhos, evitando assim o problema descrito na questão 3.

5 – Explique a diferença entre as funções `wait` e `waitpid` que são normalmente usadas no tratamento da terminação de processos filhos do servidor concorrente. Alguma dessas funções é melhor que a outra? Justifique.

A principal diferença entre o `wait()` e o `waitpid()` é seu comportamento. O `wait()` espera o processo filho terminar para poder criar um novo processo filho, já o `waitpid()` tem comportamento variável, dependendo dos parâmetros passados, neste caso em que usamos o `"waitpid(-1, &stat, WNOHANG)"`, ele não irá esperar pelos processos filhos terminarem para executar novos processos, mas irá prevenir que no fim, não tenha processos zumbis. Não há uma função melhor do que a outra, e sim, a que melhor serve no caso de aplicação, para este caso da nossa aplicação, o `waitpid()` se encaixa melhor, já que ele permite uma

execução “paralela” entre as conexões.