

1. **Pesquise como está implementado o backlog (conexões completas + conexões incompletas) de um socket TCP no kernel linux (para versão 2.2 ou mais recentes). O que representa o valor do backlog passado no listen? E o que representa o parâmetro do kernel chamado "tcp_max_syn_backlog"?**

- **Dica: use o comando man.**

O valor do backlog passado no listen determina o tamanho da fila usada para implementar o sistema. Após o kernel versão 2.2 o comportamento do backlog mudou, o valor do backlog especifica o tamanho da fila de sockets com conexão estabelecida completamente, ao invés do numero de requisições incompletas.

O tamanho máximo para a fila de sockets incompletos é definido pode ser definido por:
`/proc/sys/net/ipv4/tcp_max_syn_backlog`.

2. **Modifique o código do servidor para imprimir somente o IP e a porta do cliente.**
3. **Modifique o código do servidor de modo que o valor do backlog passado para a função `listen` seja um argumento na linha de comando. Você deve também modificar o código a fim de retardar a remoção dos sockets da fila de conexões completas.**
 - **Dica: Use a função `sleep` para isso.**
4. **Realize experimentos a fim de verificar quantos clientes (de um total de 10) conseguem de imediato conectar-se ao servidor modificado no passo anterior com os seguintes valores de backlog: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10. Elabore algum esquema para tentar conectar os 10 clientes de forma simultânea (Veja as dicas logo abaixo). Relate os experimentos realizados, máquinas utilizadas para executar o código do servidor, e do(s) cliente(s) e os resultados obtidos (coloque os comando executados junto com suas saídas). Os resultados obtidos condizem com o esperado? Consulte o arquivo `/proc/sys/net/ipv4/tcp_max_syn_backlog` e verifique o valor de backlog definido na mesma, tente exceder o número de clientes simultâneos definido e verifique o que ocorre.**
 - **Dica 1: Utilize o script [aqui](#) ou escreva seu próprio script. 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](#). Se optar por utilizar o seu ele deve ser entregue.**
 - **Dica 2: utilize o `netstat` para descobrir o número de clientes que conseguem realizar o 3-WSH 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 3WSH -- lembre-se do diagrama de estados do TCP).**

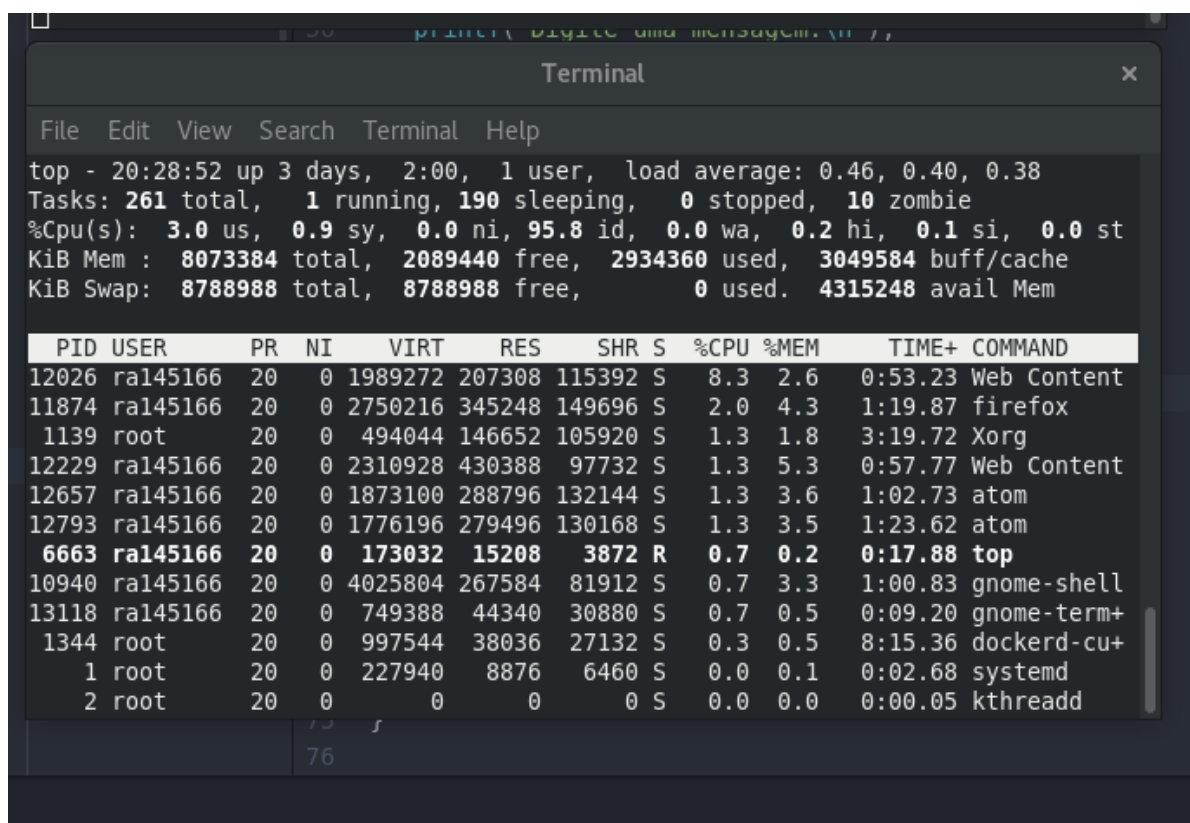
Quando rodamos com backlog = 1, 4 clientes conseguiram se conectar simultaneamente. Quando rodamos com backlog = 5, 8 clientes conseguiram se conectar simultaneamente. Portanto, condizem mas dentro da tolerância mostrada no slide 6.

5. Caso seja executado um sniffer no lado do servidor durante os experimentos do passo anterior, quais flags estarão atribuídos nos segmentos TCP capturados para um cliente enquanto ele não conseguir conectar-se ao servidor? Por que isso ocorre?

A flag que estará atribuída é a flag SYN, pois ela é a primeira enviada de um cliente para o servidor, assim como a SYN-ACK é a primeira que o servidor responde ao cliente. Como o cliente ainda está na espera por conexão a flag atribuída a seu pacote TCP é apenas a SYN.

6. É possível identificar processos zumbis por algum comando no terminal? Se sim, mostre um exemplo.

O comando Top. Segue print de sua execução:



```
top - 20:28:52 up 3 days, 2:00, 1 user, load average: 0.46, 0.40, 0.38
Tasks: 261 total, 1 running, 190 sleeping, 0 stopped, 10 zombie
%Cpu(s): 3.0 us, 0.9 sy, 0.0 ni, 95.8 id, 0.0 wa, 0.2 hi, 0.1 si, 0.0 st
KiB Mem : 8073384 total, 2089440 free, 2934360 used, 3049584 buff/cache
KiB Swap: 8788988 total, 8788988 free, 0 used. 4315248 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
 12026 ra145166   20   0 1989272 207308 115392 S   8.3   2.6   0:53.23 Web Content
 11874 ra145166   20   0 2750216 345248 149696 S   2.0   4.3   1:19.87 firefox
   1139 root        20   0 494044   146652 105920 S   1.3   1.8   3:19.72 Xorg
 12229 ra145166   20   0 2310928 430388  97732 S   1.3   5.3   0:57.77 Web Content
 12657 ra145166   20   0 1873100 288796 132144 S   1.3   3.6   1:02.73 atom
 12793 ra145166   20   0 1776196 279496 130168 S   1.3   3.5   1:23.62 atom
   6663 ra145166   20   0 173032   15208   3872 R   0.7   0.2   0:17.88 top
 10940 ra145166   20   0 4025804 267584  81912 S   0.7   3.3   1:00.83 gnome-shell
 13118 ra145166   20   0 749388   44340  30880 S   0.7   0.5   0:09.20 gnome-term+
   1344 root        20   0 997544   38036  27132 S   0.3   0.5   8:15.36 dockerd-cu+
      1 root        20   0 227940   8876   6460 S   0.0   0.1   0:02.68 systemd
      2 root        20   0      0      0      0 S   0.0   0.0   0:00.05 kthreadd
```

7. Utilize os arquivos originais e modifique de modo que os processos criados pelo fork sejam corretamente finalizados ao invés de permanecerem no estado zumbi quando um cliente encerre sua conexão. Explique qual é o problema de ter processos zumbis.

O processo zumbi ocupa um numero de processo, e o numero de processos que o sistema operacional pode endereçar no ID é limitado. Como eles ocupam pouquíssima memória, são apenas uma referência de um processo que existia, o maior problema é ocuparem o ID.