Atividade 4: Backlog e Processos Zumbis

***** Data de entrega: 14 Outubro de 2025*
**** Este trabalho deve ser realizado utilizando a linguagem C. Trabalhos que utilizarem bibliotecas externas poderão receber nota zero na implementação, pois todos os trabalhos serão compilados utilizando o padrão destas linguagens. Isto vale para quem utiliza Windows, certifique-se de compilar e executar em máquinas com Linux/Unix antes de entregar a tarefa.*

**** Lembre-se de justificar e comprovar suas respostas no relatório.*
**** Relatório arquivo deverá ser em formato PDF*
**** Detalhes da implementação:*

O código não pode apresentar nenhum warning quando compilado. Para cada warning exibido na compilação será descontado 10% da nota deste trabalho. (Certifique-se de compilar seu código com a flag -Wall do gcc).
Adicione todos os comentários necessários no código para indicar o que foi feito para
realizar cada uma das questões e identificar as diferentes funções. A ordem do seu código será levada em consideração.

**** Entregáveis: relatório e códigos. Regra para atribuição de nota: Relatório 50% e Código 50%.*

Exercício

Este exercício tem como finalidade investigar a relação entre o parâmetro backlog da chamada listen(), as configurações do kernel do sistema operacional e o impacto direto no desempenho e na capacidade de um servidor concorrente em lidar com múltiplas conexões simultâneas. Adicionalmente, o exercício aborda a correta gestão de processos filhos para evitar a criação de "zumbis".

1. Antes de modificar o código, investigue e descreva com suas próprias palavras a dinâmica das filas de conexão TCP no Kernel Linux. Sua explicação deve responder às seguintes questões:
    - Qual o papel da "fila de conexões incompletas" (SYN queue) e da "fila de conexões completas" (accept queue)?
    - Como o parâmetro backlog passado para a função listen() se relaciona com o tamanho dessas filas?
    - Qual a função do parâmetro do kernel /proc/sys/net/ipv4/tcp_max_syn_backlog e como ele interage com o backlog definido na aplicação?
2. Modifique o código de um servidor TCP concorrente (baseado em fork()) para que ele se torne uma ferramenta de teste flexível e robusta.
    - O valor do backlog deve ser passado como argumento na linha de comando.
      ```
      int backlog = atoi(argv[2]);

      Listen(listenfd, backlog);
      ```

- Adicione um terceiro argumento na linha de comando para configurar um tempo de espera. Esse tempo será usado para retardar a remoção dos sockets da fila de conexões completas, antes de fechar a conexão.

  sleep(sleep_time); // atraso em segundos no processo filho

  Assim, o servidor agora deve ser executado com três parâmetros:

  ./servidor <PORTA> <BACKLOG> <TEMPO_SLEEP>.

- Primeiramente, execute o servidor **sem** qualquer tratador de sinais para SIGCHLD. Conecte vários clientes em sequência e observe o estado dos processos do servidor utilizando o comando ps aux | grep <nome_do_servidor>. Identifique e capture uma evidência (screenshot) da existência de processos em estado "Z" (zumbi).

- Em seguida, implemente um tratador de sinais para o SIGCHLD que evite o acúmulo de processos zumbis, garantindo que o processo pai aguarde (waitpid) adequadamente pelos filhos que terminaram.

- Compile e execute o servidor novamente com a solução implementada. Repita o mesmo teste de conectar vários clientes e utilize o comando ps mais uma vez para comprovar que os processos zumbis não estão mais sendo gerados. Capture uma nova evidência que demonstre a ausência de processos "Z".

- Explique como são tratados esses processos zumbis.

3. Desenvolva um script (pode usar bash com & ou xargs -P) para automatizar os testes de carga no seu servidor. O objetivo é analisar o impacto de diferentes valores de backlog.

   Execute o servidor com TEMPO_SLEEP definido. Em seu script, varie o valor de BACKLOG de 0 a 10. Para cada valor de backlog, crie **10 clientes simultaneamente** tentando se conectar.

   - O número de clientes que conseguem estabelecer a conexão imediatamente (estado ESTABLISHED), pode ser verificado com o comando ss ou netstat.

   - Anote os resultados em uma tabela: backlog, conexões imediatas, conexões rejeitadas.

   - Compare os resultados com o valor lido em /proc/sys/net/ipv4/tcp_max_syn_backlog.

4. Rode um sniffer (ex: tcpdump -i lo tcp port 8080) durante os testes. Verifique quais **flags TCP** aparecem nos pacotes dos clientes que não conseguem conectar. Explique por que aparecem apenas **SYN retransmitidos** quando a fila está cheia.

Dicas e observações finais:

- Consulte os slides da disciplina e os capítulos 4 e 5 do livro texto guia para resolver os pontos anteriores.
- Use máquinas diferentes (servidor da VM netlabs e o cliente pode ser o host) se quiser simular latência de rede real; em localhost o comportamento tende a ser mais rápido e pode mascarar efeitos de fila por aceitação rápido. Indique claramente no relatório as máquinas utilizadas e sua estratégia.
- Para garantir simultaneidade maior, use xargs -P ou parallel para disparar processos em paralelo.
- Sempre documente os comandos exatos usados e cole as saídas no relatório (stdout/stderr).
- Para uma solução robusta, utilize sigaction(). Como o kernel pode agrupar sinais SIGCHLD, seu tratador deve usar um laço (while) com waitpid() para garantir que todos os processos zumbis sejam eliminados. Investigue a opção WNOHANG para criar um laço que não bloqueie o servidor e considere a utilidade da flag SA_RESTART.
- Tenha cuidado com parâmetros de kernel: altere tcp_max_syn_backlog apenas temporariamente com sysctl -w e restitua o valor padrão depois.

O relatório deve conter:

- Código modificado do servidor (com as mudanças destacadas).
- Scripts/estratégias usadas para lançar clientes simultâneos.
- Resultados experimentais (tabela e observações).
- Valor de tcp_max_syn_backlog e interpretação.
- Análise dos pacotes observados no sniffer.
- Conclusões sobre backlog, tcp_max_syn_backlog e processos zumbis.