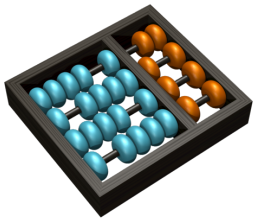

Exercício 5

Aluno: Felipe S. P. Carvalho e Arthur Maia Mendes

RA: 146040 e 135013

Instituto de Computação

Universidade Estadual de Campinas

Campinas, 20 de Outubro de 2018.

Sumário

1 Questão 1

O comportamento do argumento do backlog em soquetes TCP foi alterado com o Linux 2.2. Agora, é especificado o comprimento da fila para sockets completamente estabelecidos que aguardam para serem aceitos, em vez do número de solicitações de conexão incompletas. O comprimento máximo da fila para soquetes incompletos pode ser definido usando o parâmetro do kerlnel chamado `tcp_max_syn_backlog`. Isso significa que as versões atuais do Linux utilizam duas filas distintas: uma fila SYN com um tamanho especificado por uma configuração de todo o sistema (parâmetro do kerlnel `tcp_max_syn_backlog`) e uma fila de aceitação com um tamanho especificado pela aplicação.

O parâmetro que especifica a fila de aceitação é dado pelo parâmetro backlog passado no comando listen. Conforme a documentação encontrada em [1], o valor do backlog define o comprimento máximo para o qual a fila de conexões pendentes para o sockfd pode crescer.

2 Questão 2

Como solicitado, o código servidor foi modificado de modo que se imprima somente o `IP` e `PORTA` do cliente ao se conectar. As linhas adicionadas/modificadas no arquivo `socket_helper.c` são demonstradas no código a seguir:

```
...
int Accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen) {
    int connfd;

    if ((connfd = accept(sockfd, addr, addrlen)) == -1) {
        perror("accept");
```

```
        exit(1);
    }

    struct sockaddr_in *sin = (struct sockaddr_in *) addr;

    printf("<\%s - \%d>\\n", inet_ntoa(sin->sin_addr),(int) ntohs(sin->sin_port));

    return connfd;
}
...
```

Executamos o programa servidor e conectamos três diferentes clientes à ele, obtendo o seguinte output no console servidor:

```
personal@kiss-mbp-3 ~/L/U/2/m/g/t/codigo> ./servidor 8000
<127.0.0.1 - 50062>
<127.0.0.1 - 50066>
<127.0.0.1 - 50081>
```

3 Questão 3

Como pedido, foi modificado o código do servidor de modo que o valor `backlog` passado para a função `listen` seja um argumento na linha de comando. As linhas adicionadas/modificadas no arquivo `servidor.c` são demonstradas no código a seguir:

```
...
int main (...) {
    int       ...,
              listenq;
    ...
    if (argc != 3) {
        ...
```

```
        strcat(error," <Port, Backlog>");
        ...
    }
    ...
    listenq = atoi(argv[2]);
    ...
    Listen(listenfd, listenq);
    ...
    }
...
```

Como pedido, foi modificado o código a fim de retardar a remoção dos sockets da fila de conexões completas. As linhas adicionadas/modificadas no arquivo `servidor.c` são demonstradas no código a seguir:

```
...
for ( ; ; ) {
        ...
        connfd = Accept(...);
        sleep(5);
        ...
        }
...
```

4 Questão 4

Foram verificados o número de clientes com conseguem de imediato conectar-se ao servidor modificado no passo anterior. Foi usado o arquivo `fila.sh` para a tentativa de conexão de vários clientes simultaneamente. O servidor e o cliente foram executados numa máquina com sistema operacional macOS Mojave versão 10.14. No macOS o arquivo equivalente ao

`tcp_max_syn_backlog` é o `somaxconn`, cujo valor default é 128. O valor de `somaxconn` foi verificado através do comando `sysctl kern.ipc.somaxconn`.

Para verificar o número de conexões não estabelecidas foi usado o comando `netstat -ant | grep -c SYN_SENT` cuja saída é o número de conexões em estado `SYN_SENT`. Conexões nesse estado indicam que não houve resposta ao `SYN` por parte do servidor. De fato, quando a fila de backlog está cheia o servidor ignora novos `SYN`. As conexões que foram consideradas, por outro lado, são mostradas com estado `ESTABLISHED`. As imagens a seguir ilustram um exemplo com `backlog` com valor 5. O comando `netstat -ant | grep -c SYN_SENT` dá como saída o número de conexões em estado `SYN_SENT` e o valor condiz com o observado na lista completa de conexões tcp ilustrada através do `netstat -p tcp`.

```
[arthurmaiamendes@Arthurs-MBP:~/OneDrive/_Unicamp/2018.2/MC833/mc833/trabalho-5/codigo$ netstat -p tcp
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  localhost.60898        localhost.irdmi        SYN_SENT
tcp4       0      0  localhost.60897        localhost.irdmi        SYN_SENT
tcp4       0      0  localhost.60896        localhost.irdmi        SYN_SENT
tcp4       0      0  localhost.60895        localhost.irdmi        SYN_SENT
tcp4       0      0  localhost.irdmi        localhost.60894        ESTABLISHED
tcp4       0      0  localhost.60894        localhost.irdmi        ESTABLISHED
tcp4       0      0  localhost.irdmi        localhost.60893        ESTABLISHED
tcp4       0      0  localhost.60893        localhost.irdmi        ESTABLISHED
tcp4       0      0  localhost.irdmi        localhost.60892        ESTABLISHED
tcp4       0      0  localhost.60892        localhost.irdmi        ESTABLISHED
tcp4       0      0  localhost.irdmi        localhost.60891        ESTABLISHED
tcp4       0      0  localhost.irdmi        localhost.60890        ESTABLISHED
tcp4       0      0  localhost.60891        localhost.irdmi        ESTABLISHED
tcp4       0      0  localhost.60890        localhost.irdmi        ESTABLISHED
tcp4       0      0  localhost.irdmi        localhost.60889        ESTABLISHED
tcp4       0      0  localhost.60889        localhost.irdmi        ESTABLISHED
```

No experimento, inicialmente, variou-se o valor `backlog` de 0 a 10 com a tentativa de conexão de 10 clientes simultaneamente. Em seguida, aumentou-se o número de clientes tentando se conectar para 140. Para esse novo valor de clientes, testou-se valores de `backlog` de 0, 120 e 135. A tabela a seguir ilustra os resultados obtidos:

Valor de backlog	Conexões solicitadas	Conexões simultâneas concluidas
0	10	10
1	10	2
2	10	3
3	10	4
4	10	5
5	10	6
6	10	7
7	10	8
8	10	9
9	10	10
10	10	10
120	140	121
135	140	129
0	140	129

Concluimos que para um valor de backlog igual 0, o sistema define o valor de backlog como igual ao valor de somaxconn e permite somaxconn+1 conexões; para valores inferiores a somaxconn, o sistema permite backlog+1 conexões; e para valores iguais ou superiores a somaxconn, o sistema permite somaxconn+1 conexões.

5 Questão 5

Durante os experimentos do passo anterior, executamos um sniffer para interceptar as connexões TCP através do comando `lsof -i 4tcp`. Após interceptar as conexões, foi-se possível analisar as flags atribuídas nos segmentos TCP capturados para clientes que não conseguem conectar-se ao servidor:

```
COMMAND     PID     USER   FD   TYPE             DEVICE SIZE/OFF NODE NAME
cliente     275 personal    0u  IPv4 0xea396e79b73117ab      0t0  TCP localhost:62608->localhost:sunwebadmin (ESTABLISHED)
cliente     276 personal    0u  IPv4 0xea396e79d595e10b      0t0  TCP localhost:62609->localhost:sunwebadmin (ESTABLISHED)
cliente     277 personal    0u  IPv4 0xea396e79ce8473cb      0t0  TCP localhost:62606->localhost:sunwebadmin (ESTABLISHED)
cliente     278 personal    0u  IPv4 0xea396e79aca0d7ab      0t0  TCP localhost:62607->localhost:sunwebadmin (ESTABLISHED)
cliente     279 personal    0u  IPv4 0xea396e79e411d7ab      0t0  TCP localhost:62600->localhost:sunwebadmin (ESTABLISHED)
cliente     280 personal    0u  IPv4 0xea396e79aa9473cb      0t0  TCP localhost:62614->localhost:sunwebadmin (ESTABLISHED)
cliente     281 personal    0u  IPv4 0xea396e79e416f7ab      0t0  TCP localhost:62601->localhost:sunwebadmin (ESTABLISHED)
cliente     283 personal    0u  IPv4 0xea396e79e3fe6e4b      0t0  TCP localhost:62602->localhost:sunwebadmin (ESTABLISHED)
cliente     284 personal    0u  IPv4 0xea396e79b4b403cb      0t0  TCP localhost:62605->localhost:sunwebadmin (ESTABLISHED)
cliente     286 personal    0u  IPv4 0xea396e79d4b863cb      0t0  TCP localhost:62611->localhost:sunwebadmin (ESTABLISHED)
cliente     287 personal    0u  IPv4 0xea396e79de8fca6b      0t0  TCP localhost:62603->localhost:sunwebadmin (ESTABLISHED)
cliente     288 personal    0u  IPv4 0xea396e79b7310e4b      0t0  TCP localhost:62604->localhost:sunwebadmin (ESTABLISHED)
cliente     289 personal    0u  IPv4 0xea396e79d7db510b      0t0  TCP localhost:62610->localhost:sunwebadmin (ESTABLISHED)
cliente     293 personal    0u  IPv4 0xea396e79e0d083cb      0t0  TCP localhost:62619->localhost:sunwebadmin (SYN_SENT)
cliente     295 personal    0u  IPv4 0xea396e79c496fa6b      0t0  TCP localhost:62618->localhost:sunwebadmin (SYN_SENT)
cliente     296 personal    0u  IPv4 0xea396e79c32227ab      0t0  TCP localhost:62620->localhost:sunwebadmin (SYN_SENT)
cliente     297 personal    0u  IPv4 0xea396e79d51353cb      0t0  TCP localhost:62621->localhost:sunwebadmin (SYN_SENT)
cliente     298 personal    0u  IPv4 0xea396e79bf2d07ab      0t0  TCP localhost:62622->localhost:sunwebadmin (SYN_SENT)
cliente     299 personal    0u  IPv4 0xea396e79e410d7ab      0t0  TCP localhost:62623->localhost:sunwebadmin (SYN_SENT)
cliente     300 personal    0u  IPv4 0xea396e79e410ea6b      0t0  TCP localhost:62625->localhost:sunwebadmin (SYN_SENT)
cliente     301 personal    0u  IPv4 0xea396e79e410e10b      0t0  TCP localhost:62624->localhost:sunwebadmin (SYN_SENT)
cliente     302 personal    0u  IPv4 0xea396e79e410ce4b      0t0  TCP localhost:62626->localhost:sunwebadmin (SYN_SENT)
cliente     303 personal    0u  IPv4 0xea396e79d4dba7ab      0t0  TCP localhost:62627->localhost:sunwebadmin (SYN_SENT)
cliente     304 personal    0u  IPv4 0xea396e79d4dbc3cb      0t0  TCP localhost:62628->localhost:sunwebadmin (SYN_SENT)
cliente     305 personal    0u  IPv4 0xea396e79d4dbba6b      0t0  TCP localhost:62629->localhost:sunwebadmin (SYN_SENT)
cliente     306 personal    0u  IPv4 0xea396e79d4dbb10b      0t0  TCP localhost:62630->localhost:sunwebadmin (SYN_SENT)
cliente     307 personal    0u  IPv4 0xea396e79c3221e4b      0t0  TCP localhost:62632->localhost:sunwebadmin (SYN_SENT)
cliente     308 personal    0u  IPv4 0xea396e79c32243cb      0t0  TCP localhost:62631->localhost:sunwebadmin (SYN_SENT)
cliente     309 personal    0u  IPv4 0xea396e79c322310b      0t0  TCP localhost:62633->localhost:sunwebadmin (SYN_SENT)
cliente     310 personal    0u  IPv4 0xea396e79aa94610b      0t0  TCP localhost:62634->localhost:sunwebadmin (SYN_SENT)
cliente     311 personal    0u  IPv4 0xea396e79aa9457ab      0t0  TCP localhost:62635->localhost:sunwebadmin (SYN_SENT)
cliente     312 personal    0u  IPv4 0xea396e79b671910b      0t0  TCP localhost:62638->localhost:sunwebadmin (SYN_SENT)
cliente     313 personal    0u  IPv4 0xea396e79bf2d1a6b      0t0  TCP localhost:62636->localhost:sunwebadmin (SYN_SENT)
cliente     314 personal    0u  IPv4 0xea396e79bf2cfe4b      0t0  TCP localhost:62637->localhost:sunwebadmin (SYN_SENT)
cliente     315 personal    0u  IPv4 0xea396e79b6719a6b      0t0  TCP localhost:62639->localhost:sunwebadmin (SYN_SENT)
cliente     316 personal    0u  IPv4 0xea396e79b671a3cb      0t0  TCP localhost:62640->localhost:sunwebadmin (SYN_SENT)
cliente     318 personal    0u  IPv4 0xea396e79e0f9d3cb      0t0  TCP localhost:62642->localhost:sunwebadmin (SYN_SENT)
cliente     319 personal    0u  IPv4 0xea396e79e0f9ae4b      0t0  TCP localhost:62643->localhost:sunwebadmin (SYN_SENT)
cliente     321 personal    0u  IPv4 0xea396e79e0f9b7ab      0t0  TCP localhost:62641->localhost:sunwebadmin (SYN_SENT)
cliente     322 personal    0u  IPv4 0xea396e79cefb2e4b      0t0  TCP localhost:62645->localhost:sunwebadmin (SYN_SENT)
```

No experimento realizado, após o número de máximo de clientes que podem estar conectados simultaneamente for excedido, todos os outros clientes ficam em estado `SYN_SENT`. Isso indica que: ou SYN foi enviado pelo cliente e não chegou ao servidor, ou servidor não respondeu a ele, ou o servidor optou por respondê-lo sem acompanhá-lo. No caso deste experimento, sabemos que extrapolamos o número de conexões em relação ao parâmetro `backlog`. Sendo assim, o servidor optou por não responder aos clientes, gerando o status acima.

6 Questão 6

Sim, através do comando `ps -a` é possível identificar processos zumbis. Para exemplificar, utilizaremos os programas cliente e servidor fornecidos para este exercício. Após o cliente encerrar a conexão com servidor, executamos o comando `ps -a` em um terminal, obtendo o seguinte resultado:

```
personal@kiss-mbp-3 ~/L/U/2/m/g/t/codigo> ps -a
  PID TTY           TIME CMD
14545 ttys000    0:48.56 ng serve
14546 ttys000    0:26.22 /usr/local/Cellar/node/8.1.4/bin/node /Users/kis/WebstormProjects/professional_health_web/node_modules/@ngtools/webpa
45761 ttys000    0:00.05 /bin/bash --rcfile /Applications/WebStorm.app/Contents/plugins/terminal/jediterm-bash.in --login -i
45782 ttys000    0:00.54 fish
23127 ttys001    0:00.04 /bin/bash --rcfile /Applications/IntelliJ IDEA.app/Contents/plugins/terminal/jediterm-bash.in --login -i
23149 ttys001    0:00.76 fish
 5186 ttys002    0:00.02 /Applications/Xcode.app/Contents/Developer/usr/bin/git diff
 5187 ttys002    0:00.01 /usr/bin/less
11259 ttys002    0:00.02 /Applications/Xcode.app/Contents/Developer/usr/bin/git diff
11260 ttys002    0:00.01 /usr/bin/less
77972 ttys002    0:00.04 /bin/bash --rcfile /Applications/WebStorm.app/Contents/plugins/terminal/jediterm-bash.in --login -i
77988 ttys002    0:02.76 fish
 4054 ttys003    0:00.02 login -pf personal
 4055 ttys003    0:00.01 -bash
 4061 ttys003    0:00.91 fish
49115 ttys003    0:00.00 ./servidor 8000
49190 ttys003    0:00.00 (servidor)
 6982 ttys004    0:00.02 login -pfl personal /bin/bash -c exec -la bash /bin/bash
 6983 ttys004    0:00.02 -bash
 6989 ttys004    0:00.10 fish
47404 ttys006    0:00.03 login -pfl personal /bin/bash -c exec -la bash /bin/bash
47405 ttys006    0:00.02 -bash
47411 ttys006    0:00.23 fish
49265 ttys006    0:00.00 ps -a
```

O processo de PID 49190, CMD (`servidor`), é o processo zumbi. Quando um cliente conecta-se no servidor, é criado um novo processo utilizando-se a chamada de sistema `fork()`. No término do processo filho, um sinal `"SIGCHLD"` é gerado e entregue ao pai pelo kernel. O processo pai, ao receber `"SIGCHLD"`, verifica o status do processo filho na tabela de processos. Quando o processo pai coleta o status, essa entrada é excluída. Assim, todos os vestígios do processo filho são removidos do sistema. Se o pai decidir não aguardar o término do filho e executar sua tarefa subsequente, no término do filho, o status de saída não será lido. Portanto, permanece uma entrada na tabela de processos, mesmo após o término da criança. Esse estado do processo filho é conhecido como o estado de zumbi, ocorrendo nos exemplos servidor e programa deste exercício.

7 Questão 7

Quando o processo filho é encerrado, um sinal "SIGCHLD" correspondente é entregue ao pai, se chamarmos de `signal (SIGCHLD, SIG_IGN)`, então o sinal SIGCHLD é ignorado pelo sistema, e a entrada do processo filho é deletada da tabela de processos . Assim, nenhum zumbi é criado. No entanto, neste caso, o pai não pode saber sobre o status de saída do filho.

Como solicitado, o código servidor foi modificado de modo que o problema seja resolvido. Inserimos a chamada de função `signal` no trecho de código referente ao processo pai:

```
...
int main (...) {
    ...

        Close(connfd);

        exit(0);
      }

      signal(SIGCHLD,SIG_IGN);

      Close(connfd);
   }

   return(0);
}
...
```

Depois essa alteração, realizamos um conexão com o novo servidor. Após o cliente encerrar a conexão, executamos o comando `ps -a` em um terminal, obtendo o seguinte resultado:

```
personal@kiss-mbp-3 ~/L/U/2/m/g/t/codigo> ps -a
  PID TTY           TIME CMD
14545 ttys000    0:48.60 ng serve
14546 ttys000    0:26.22 /usr/local/Cellar/node/8.1.4/bin/node /Users/kis/WebstormProjects/professional_health_web/node_modules/@ngtools/webpa
45761 ttys000    0:00.05 /bin/bash --rcfile /Applications/WebStorm.app/Contents/plugins/terminal/jediterm-bash.in --login -i
45782 ttys000    0:00.54 fish
23127 ttys001    0:00.04 /bin/bash --rcfile /Applications/IntelliJ IDEA.app/Contents/plugins/terminal/jediterm-bash.in --login -i
23149 ttys001    0:00.76 fish
 5186 ttys002    0:00.02 /Applications/Xcode.app/Contents/Developer/usr/bin/git diff
 5187 ttys002    0:00.01 /usr/bin/less
11259 ttys002    0:00.02 /Applications/Xcode.app/Contents/Developer/usr/bin/git diff
11260 ttys002    0:00.01 /usr/bin/less
77972 ttys002    0:00.04 /bin/bash --rcfile /Applications/WebStorm.app/Contents/plugins/terminal/jediterm-bash.in --login -i
77988 ttys002    0:02.76 fish
 4054 ttys003    0:00.02 login -pf personal
 4055 ttys003    0:00.01 -bash
 4061 ttys003    0:00.92 fish
54219 ttys003    0:00.00 ./servidor 8000
 6982 ttys004    0:00.02 login -pfl personal /bin/bash -c exec -la bash /bin/bash
 6983 ttys004    0:00.02 -bash
 6989 ttys004    0:00.11 fish
47404 ttys006    0:00.03 login -pfl personal /bin/bash -c exec -la bash /bin/bash
47405 ttys006    0:00.02 -bash
47411 ttys006    0:00.37 fish
54289 ttys006    0:00.00 ps -a
personal@kiss-mbp-3 ~/L/U/2/m/g/t/codigo>
```

Note que o processo zumbi, que seria representado por um novo processo com CMD (`servidor`), não está presente.

Bibliografia

[1] https://linux.die.net/man/2/listen