MC833 - PROGRAMAÇÃO DE REDES DE COMPUTADORES

Exercício 7 - Estudo do backlog e tratamento de processos zumbis

Fernando Luis de Oliveira Costa - RA: 091188
Oscar dos Santos Esgalha Neto - RA: 108231

1.

O argumento **backlog** define o comprimento máximo que a fila pode crescer para receber conexões pendentes.

tcp_max_syn_backlog é o número máximo de solicitações de conexão na fila que ainda não recebeu uma confirmação de conexão com o cliente.

2.

Obs: As alterações feitas no servidor se encontram no arquivo servidor_passo2.c

As únicas modificações feitas no código foram:

- **O valor do baklog no Listen que é passado como argumento na linha de comando:**

```
Listen(listenfd, atoi(argv[2]));
```

- **Retardando o fechamento das conexões clientes:**

```
sleep(5);
// fecha a conexão do processo filho
Close(connfd);
```

3.

Máquina do cliente: Ubuntu 14.04 64-bit
Máquina do servidor: Ubuntu 14.04 64-bit
Exemplo de saída para o netstat com backlog = 0:

```
vagrant@vagrant-ubuntu-trusty-64:~$ netstat -ntp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:1024          127.0.0.1:48305         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48301         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48308         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48291         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48300         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48307         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48298         TIME_WAIT   -
tcp        0      0 127.0.0.1:48309         127.0.0.1:1024          ESTABLISHED 1731/cliente
tcp        0      0 127.0.0.1:1024          127.0.0.1:48303         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48309         ESTABLISHED 1732/servidor
tcp        0      0 127.0.0.1:1024          127.0.0.1:48293         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48299         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48306         TIME_WAIT   -
tcp        0      0 10.0.2.15:22            10.0.2.2:60772          ESTABLISHED -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48302         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48294         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48297         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48296         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48295         TIME_WAIT   -
tcp        0      0 10.0.2.15:22            10.0.2.2:60770          ESTABLISHED -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48304         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48290         TIME_WAIT   -
tcp        0      0 127.0.0.1:1024          127.0.0.1:48292         TIME_WAIT   -
tcp        0      0 10.0.2.15:22            10.0.2.2:60771          ESTABLISHED -
```

Bash script para executar 10 clientes simultaneamente:

```
#!/bin/bash
for i in `seq 1 10`;
do
  (./cliente 127.0.0.1 1024 < input) &
done
```

Tabela de conexões:

Backlog	Número de conexões
0	1
1	2
2	2
3	3
4	3

5	4
6	4
7	5
8	5
9	6
10	6

4.

Um sniffer dectaria as seguintes flags para um cliente que não consegue se conectar no servidor: tcpHalfOpenDrop, tcpListenDrop e tcpListenDropQ0. Essas flags indicam que a fila de conexões do servidor está cheia e não é possível fazer o 3WHS para estabelecer mais uma conexão.

5.

Obs: As alterações feitas no servidor se encontram no arquivo servidor_passo5.c

As modificações feitas no código foram:

- **Chamar a função waitpid ante do for:**

```
// chama waitpid()
  Signal(SIGCHLD, sig_chld);
  // Loop infinito
  for ( ; ; ) {
```

- **Tratamento do sinal:**

```
if ( (connfd = Accept(listenfd, &clientaddr)) < 0) {
        if (errno != EINTR) {
                err_sys("accept error");
        }
}
```

Obs: Vale ressaltar que as implementações das funções chamadas (Signal, sig_chld) também foram feitas nos arquivos socket_utils.h e socket_utils.c:

```
// Função que trata o sig_chld
void sig_chld(int signo) {
        pid_t pid;
```

```
        int stat;
        while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
                printf("child %d terminated\n", pid);
        return;
}

// função para tratar erro
void err_sys(const char* x)
{
   perror(x);
   exit(1);
}

// função para tratar o sinal
Sigfunc * Signal (int signo, Sigfunc *func)
{
        struct sigaction act, oact;
        act.sa_handler = func;
        sigemptyset (&act.sa_mask); /* Outros sinais não são bloqueados*/
        act.sa_flags = 0;
        if (signo == SIGALRM) { /* Para reiniciar chamadas interrompidas */
#ifdef SA_INTERRUPT
                act.sa_flags |= SA_INTERRUPT; /* SunOS 4.x */
#endif
        } else {
#ifdef SA_RESTART
                act.sa_flags |= SA_RESTART; /* SVR4, 4.4BSD */
#endif
        }
        if (sigaction (signo, &act, &oact) < 0)
                return (SIG_ERR);
        return (oact.sa_handler);
}
```