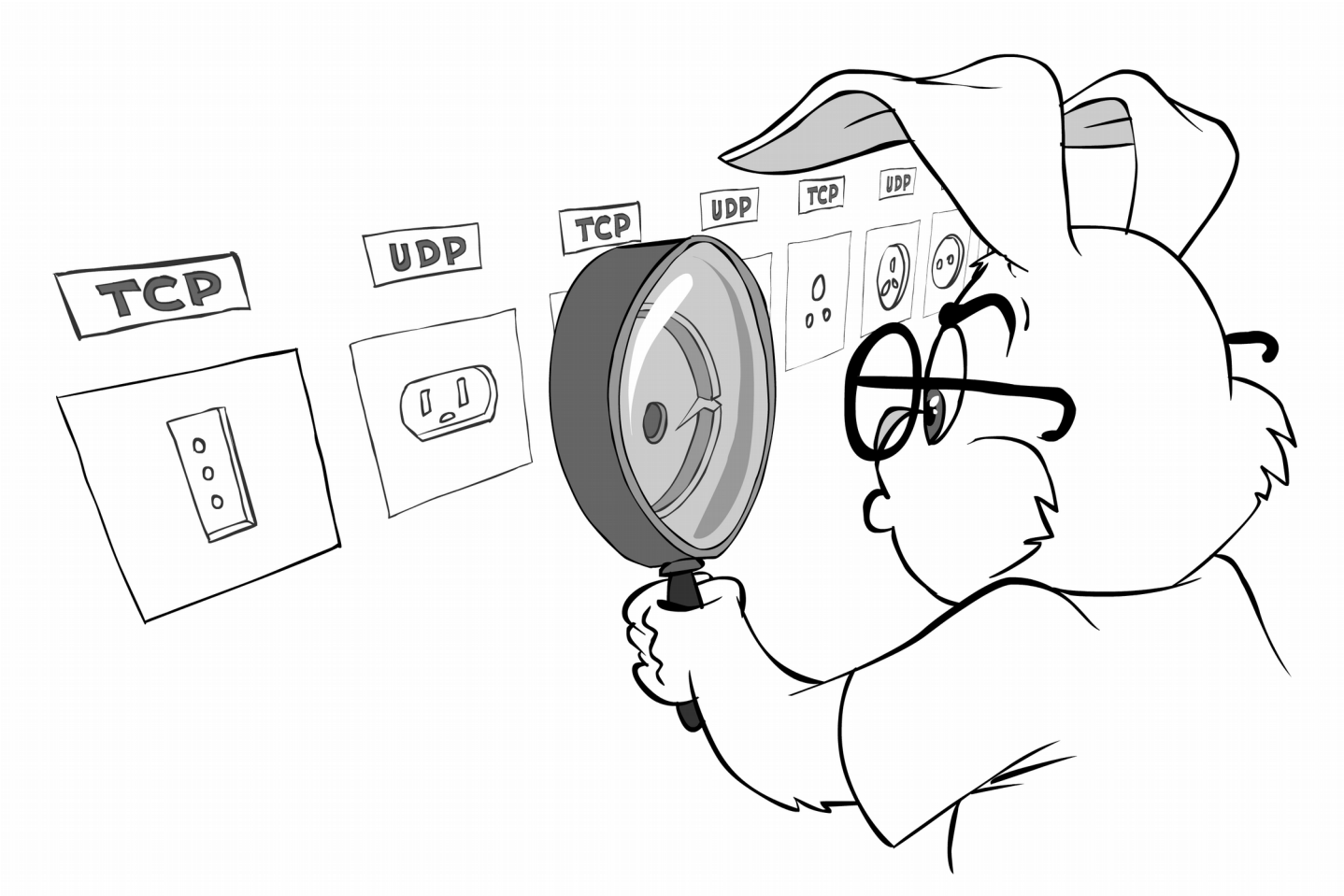
DEFAULT HACKERCLUB

Sockets sem bloqueio na prática



Por padrão, os sockets TCP estão no modo de bloqueio. Por exemplo, quando você chama recv() para ler de um fluxo, o controle não é retornado ao programa até que pelo menos um byte de dados seja lido no site remoto. Esse processo de aguardar que os dados apareçam é chamado de bloqueio. O mesmo vale para write(), connect(), etc. Quando você as executa, a conexão "bloqueia" até que a operação seja concluída.

É possível definir um descritor para que ele seja colocado no modo sem bloqueio (NonBlocking). Quando colocado no modo sem bloqueio, você nunca espera que uma operação seja concluída. Esta é uma ferramenta inestimável se você precisar alternar entre muitos sockets conectados diferentes e quiser garantir que nenhum deles faça com que o programa seja "travado".

Um soquete é colocado no modo sem bloqueio chamando a função fcntl() da seguinte maneira:

```
flags = fcntl(sd, F_GETFL, 0);
if(fcntl(sd, F_SETFL, flags | O_NONBLOCK) == -1){
    printf("Erro na função fcntl() ao criar socket sem bloqueio");
    exit(-1);
}
```

Abaixo faremos um scanner de portas como exemplo do uso de sockets sem bloqueio, então, quando a função connect() é chamada para um socket TCP sem bloqueio, o processo de estabelecimento de conexão é iniciado (o primeiro pacote do handshake TCP de três vias é enviado) e o erro **EINPROGRESS** é retornado imediatamente. O scanner de portas deve estar atento a esse erro, o que significa que o estabelecimento da conexão foi iniciado e está em andamento. Em casos raros, quando o servidor está no mesmo host que o cliente, uma conexão pode ser estabelecida imediatamente, portanto, mesmo para sockets sem bloqueio, é necessário monitorar a função connect() para garantir que ela seja executada com êxito.
O estado do socket é monitorado usando a função select() e as macros **FD_ZERO**, **FD_SET** e **FD_ISSET**. Se um socket se torna imediatamente pronto para operações de leitura ou gravação, uma conexão com a porta remota foi estabelecida, isso significa que a porta está no modo de escuta.

O scanner monitora três estados de socket:
- estado = 0 ---- O socket não foi criado
- estado = 1 ---- Um socket foi criado
- estado = 2 ---- O socket está no modo de escuta

O scanner vai aguardar os argumentos na linha de comando do host, intervalo de portas e o tempo em segundos para aguardar o socket estar pronto. Uma outra observação é que será necessário os pacotes **libc6-dev-i386** e **gcc-multilib** para compilar o código.

gcc -m32 Scanner.c -o Scanner

Link do código: https://github.com/RyoonIvo/port-scanner-example/blob/master/Scanner.c

---

```
/* Scanner de portas utilizando sockets NonBlocking

 * Compilar: gcc Scanner.c -m32 -o Scanner
 * Pacotes necessários se estiver com Linux de 64 bit: libc6-dev-i386 e gcc-multilib
 * By Ryoon Ivo
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <time.h>
#include <errno.h>
#include <netdb.h>

#define MAX_SOCK 50 // O número máximo de sockets verificados em uma passagem

// Função para saída de informações sobre portas abertas
portaAberta(int porta){
    struct servent *srvport;
    srvport = getservbyport(htons(porta), "tcp");
    if (srvport == NULL)
        printf("Aberta: %d - Desconhecido\n", porta);
    else
        printf("Aberta: %d - (%s)\n", porta, srvport->s_name);
    fflush(stdout);
}

int main(int argc, char *argv[]){
    // Uma estrutura para monitorar os estados do socket
    struct usock_descr{
    int sd;                    // Descritor de socket
    int state;                 // Estado do socket
    long timestamp;            // Tempo de abertura do socket em milissegundos
    unsigned short remoteport; // Porta remota
    };

    struct usock_descr sockets[MAX_SOCK];
    struct hostent* hp;
    struct sockaddr_in servaddr;
    struct timeval tv = {0,0};
    fd_set rfds, wfds;
    int i, flags, max_fd;
    int porta, portaInicial, portaFinal;
    int MAXTIME;

    if(argc != 5){
        printf("Modo de uso: %s (IP) (portaInicial) (portaFinal) (tempoEmSegundos)\n",
argv[0]);
        printf("Exemplo: ./Scanner 192.168.0.1 1 1000 10 \n");
        exit(-1);
    }
    // Atribui o IP/host a variável hp
    hp = gethostbyname(argv[1]);
    // Se o alvo (hp) não estiver disponível vai executar a mensagem de erro
    if(hp == NULL){
        printf("Host indisponível \n");
        exit(-1);
    }
```

```
    // Atribuindo argumentos da linha de comando as variáveis
    portaInicial = atoi(argv[2]); // Porta de inicio do scan
    portaFinal = atoi(argv[3]) + 1; // Porta final do scan
    MAXTIME = atoi(argv[4]); // Tempo em segundos para esperar o socket ficar pronto

    printf("Escaneando portas... \n");
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr = *((struct in_addr *)hp->h_addr);

    // Configura todos os sockets para estado 0
    porta = portaInicial;
    for(i = 0; i < MAX_SOCK; i++)
        sockets[i].state = 0;

    // Loop principal é executado até que todas as portas sejam verificadas
    while(porta < portaFinal){
        // Criando um socket, configurando para o modo sem bloqueio e
        // definindo seu estado como 1
        for(i = 0; (i < MAX_SOCK) && (porta < portaFinal); i++){
            if (sockets[i].state == 0){
                if((sockets[i].sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1){
                        printf("Falha na função socket() \n");
                        exit(-1);
                    }
                flags = fcntl(sockets[i].sd, F_GETFL, 0);
                if(fcntl(sockets[i].sd, F_SETFL, flags | O_NONBLOCK) == -1){
                        printf("Erro na função fcntl() ao criar socket sem bloqueio \n");
                        exit(-1);
                    }
                sockets[i].state = 1;
            }
        }

        for (i = 0; (i < MAX_SOCK) && (porta < portaFinal); i++){
            // Verificando os sockets no estado 1 e tentando se conectar com a porta
remota
            if (sockets[i].state == 1){
                servaddr.sin_port = ntohs(porta);
                if (connect(sockets[i].sd, (struct sockaddr *)&servaddr, sizeof
(servaddr)) == -1){
                    // A chamada connect() terminou em um erro diferente de EINPROGRESS
                    // então fecha o socket e define o estado como 0
                    if (errno != EINPROGRESS){
                        shutdown(sockets[i].sd, 2);
                        close(sockets[i].sd);
                        sockets[i].state = 0;
                    }
                // A chamada connect() retornou o erro EINPROGRESS;
                // então defina o estado do socket como 2 para aguardar o estabelecimento
da conexão
                else
                        sockets[i].state = 2;
                }
                // A conexão foi estabelecida imediatamente; isto é, a porta está aberta,
enviando suas informações para a tela.
                else{
                        portaAberta(porta);
                        // O socket pode ser fechado e seu estado configurado para 0
                    shutdown(sockets[i].sd, 2);
                    close(sockets[i].sd);
                    sockets[i].state = 0;
                }
            // Lembrando a hora em que a solicitação de conexão foi feita e a porta
remota sendo sondada
            sockets[i].timestamp = time(NULL);
            sockets[i].remoteport = porta;
            porta++; // Passando para a próxima porta para escanear
```

```
            }
        }

    // Zerando os conjuntos de descritores
    FD_ZERO(&rfds);
    FD_ZERO(&wfds);
    max_fd = -1;

    for (i = 0; i < MAX_SOCK; i++) {
        // Se o socket estiver no modo de escuta
        // coloque nos conjuntos correspondentes para a verificação seguinte
        if (sockets[i].state == 2) {
            FD_SET(sockets[i].sd, &wfds);
            FD_SET(sockets[i].sd, &rfds);
            if (sockets[i].sd > max_fd)
                    max_fd = sockets[i].sd;
      }
    }

    // Verificando o estado dos sockets
    select(max_fd + 1, &rfds, &wfds, NULL, &tv);

    for (i = 0; i < MAX_SOCK; i++){
        if (sockets[i].state == 2){
        // Verificando se o socket fornecido está no conjunto de descritores
        // e pronto para operações de leitura ou gravação
            if (FD_ISSET(sockets[i].sd, &wfds) || FD_ISSET(sockets[i].sd, &rfds)){
                int error;
                socklen_t err_len = sizeof(error);
                // Verificando um erro de conexão
                if (getsockopt(sockets[i].sd, SOL_SOCKET, SO_ERROR, &error, &err_len) < 0
|| error != 0){
                    // Se houver um erro de conexão, fecha o socket e define seu estado
como 0
                    shutdown(sockets[i].sd, 2);
                    close(sockets[i].sd);
                    sockets[i].state = 0;
                }
                else{
                    // Se não houver erro, a conexão foi estabelecida com sucesso :)
                        portaAberta(sockets[i].remoteport);
                        // O socket pode ser fechado e seu estado configurado para 0
                    shutdown(sockets[i].sd, 2);
                    close(sockets[i].sd);
                    sockets[i].state = 0;
                }
            }
            else{
            // Se o soquete não estiver pronto para operações de leitura ou gravação
            // verifique há quanto tempo ele está nesse estado
            // se o tempo limite em segundos especificado na linha de comando expirar
            // feche o socket e defina seu estado para 0
                    if ( (time(NULL) - sockets[i].timestamp) > MAXTIME){
                    shutdown(sockets[i].sd, 2);
                    close(sockets[i].sd);
                    sockets[i].state = 0;
                }
            }
        }
    }
    }

    return 0;
}
```