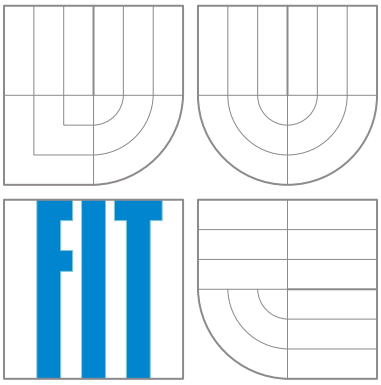


Dokumentace k projektu ISA

Seznam služeb běžících na zadaných počítačích

18. listopadu 2012

Autor: Petr Dvořáček, xdvora0n@stud.fit.vutbr.cz
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah

1 Úvod

V dnešní době disponujeme kvantem programů na skenování TCP portů. Mezi nejznámější patří programy `telnet` a `netcat`. Pomocí niž také můžeme získat bannery bežících služeb. U prvního zmiňovaného nemůžeme zadat výčet adres a portů. [1] Tento problém nám však ulehčí program `netcat`, u kterého můžeme zadat interval portů. [2] Bohužel s výčtem adres si neporadí.

Tento dokument popisuje návrh a implementaci aplikace pro detekci služeb běžících na zadaných počítačích. Navržený program funguje jako konzolová aplikace, která ze souboru, nebo ze standardního vstupu obdrží seznam IP adres či doménových jmen. Posléze se program pokusí přihlásit na vzdálený počítač na zadané TCP porty a v případě úspěchu vypíše na standardní výstup port. Program následně vypisuje všechna obdržená data do ukončení spojení ze strany serveru, či vypršení zadaného času.

V první části dokumentu (kapitola 2) se věnuji timeoutu a neblokujícím soketům. Následuje kapitola implementace (3), kde je rozepsán zvolený způsob řešení. V další kapitole popisuji návod na použití programu (4.1). V závěru (5) uvažuji o možných rozšíření programu.

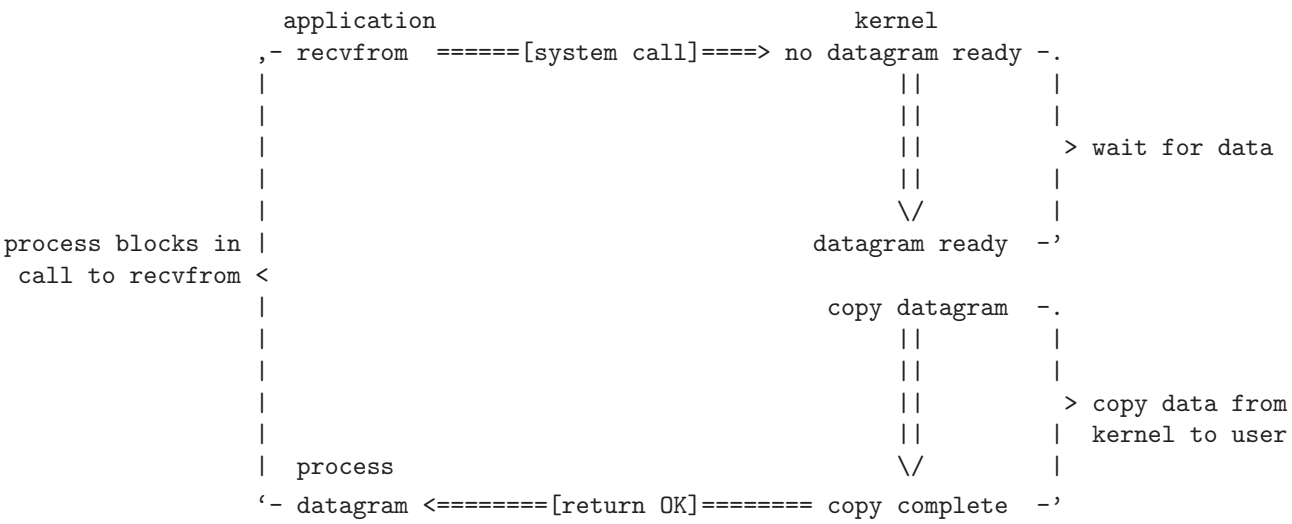
2 Analýza řešení

Jak jsem již v úvodu matně naznačil, hlavní problematikou tohoto projektu je timeout (maximální doba čekání) na data. Jako první řešení mě napadlo využití vláken, kde v jednom by byl časovač a v druhém by se čekalo na data. Časem jsem tento způsob opovrhnul. V knize UNIX Network Programming (viz [3]) jsou doporučena řešení následující:

1. Použít funkci `alarm()`, která vygeneruje signál `SIGALRM` po vypršení zadaného času.
2. Použití I/O multiplexingových funkcí `select()` nebo `poll()`. Tyto funkce mají jeden z parametrů časový limit, po který čekají.
3. Nastavit soketu volby `SO_RCVTIMEO`, nebo `SO_SNDTIMEO`. Tato metoda patří mezi novější, avšak je třeba počítat s tím, že starší OS tuto možnost nepodporují.

V implementaci jsem využil právě druhou možnost – I/O multiplexing. Pomocí tohoto modelu můžeme také spravovat více soketů najednou, což je velkou výhodou pro následné možné rozšíření.

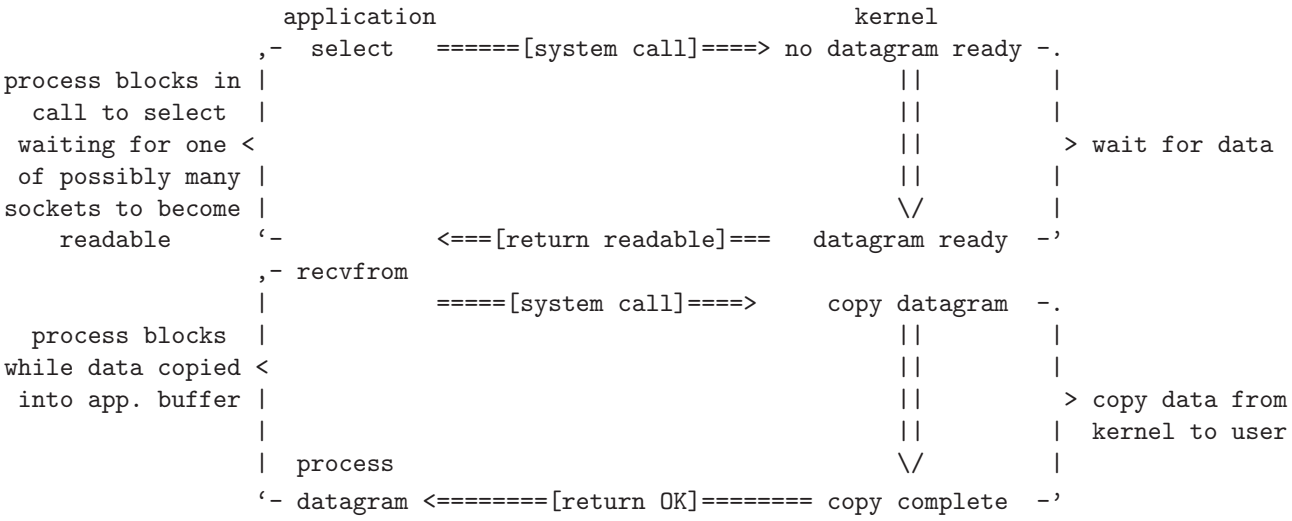
2.1 Blokující soket



Blokující model I/O.

Běžný soket při vytvoření je ve stavu tzv. blokujícím I/O. Z obrázku výše vidíte, že funkce `recvfrom()` blokuje celou dobu proces v aplikaci.

2.2 Multiplexing I/O



I/O multiplexing.

Využívá se k přístupu více soketům najednou. Funkce `select()` blokuje proces do té doby než jádro operačního systému dostane danný datagram. Obdrží-li data do určité doby, může se volat funkce `recvFrom()`. Jinak se jedná o timeout – vypršení maximalní čekací doby.

3 Implementace

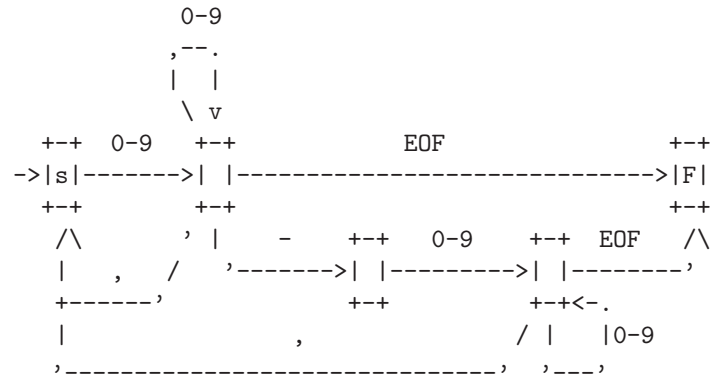
Z důvodu zdkonalení mé angličtiny používám anglické komentáře. Program je k dispozici v souboru `main.cpp`.

V hlavní funkci `main()` se zpracovávávají vstupní parametry příkazové řádky. Vyparsrují se vstupní porty a soubor, čímž dostanu potřebný seznam portů a seznam adres. S těmito seznamy pak pracuje funkce `tcpSearch()`, ta se pak pokusí připojit na porty serverů.

Dojde-li k nějaké chybě, uloží se do proměnné `errorflags`, která je interpretována jako bitové pole. V něm se nachází veškeré chyby, jež nastaly před voláním funkce `tcpsearch()`. Například může nastat situace, kdy jsou nesprávně zadané porty a chybný název souboru zároveň. Tyto chyby se pak tisknou na standardní chybový výstup.

3.1 Parsrovaní portů

Probíhá ve funkci `parsePorts()`, jejímž jediným parametrem je řetězec portů. Gramatika tohoto řetězce je zjednodušeně znázorněna v kapitole 4.3. Přepis do konečného automatu vypadá takto:



Konečný automat syntaxu vstupního řetězce portů.
Přijme-li jakýkoli jiný znak → vypíše se chyba.
Počáteční stav – s.
Konečný stav – F.

Jednotlivé porty se během tohoto parsrovaní ukládají do globální proměnné `ports`.

3.2 Parsrování adres

Ve funkci `parseFile()` se nejdříve otevře soubor. Pak se iteruje soubor po řádcích. Jednotlivý řádek parsuji ve funkci `parseLine()`. V podstatě jde jen o vymazání části řetězce od prvního bílého znaku, do konce

Tedy například z řetězce ve formátu `[text1] [mezera or tabulator] [text2]` vznikne pouze `[text1]`

Po vymazání se adresa uloží do globální proměnné `addresses`.

3.3 Detekce otevřených portů

Detekování jednotlivých portů se děje sekvenčeně. Ve funkci `tcpSearch()` se iteruje oběma seznamy. V jednotlivé iteraci získám kombinaci port – adresa a v ní následně volám `_tcpSearchSocket()`. Tato funkce vytvoří soket, na který se pak pokusí připojit ve funkci `__tcpSearchConnect()`.

V této funkci se nastaví socket na neblokující, byl-li zadán parametr `-a` (viz kapitola 4.1). Pro vytvoření vlastního timeoutu připojení k serveru používám `poll()`. Pak jestli se opravdu program připojil k cílovému počítači využívám `getsockopt()`.

Proběhlo-li připojení se serverem úspěšně, volá se `___tcpSearchGetbanner()`. Byla-li zadána maximalní doba pro čtení banneru, použije se tato doba ve funkci `poll()`, nebo `select()`. Pro přijetí banneru využívám `recvfrom()`.

3.4 Poll() vs Select()

Jak jste si mohli povšimnout, naimplementoval jsem obě funkce pro timeout. Chcete-li používat funkci `poll()`, stačí definovat `POLL_IMPLEMENTATION` v souboru `main.cpp` jinak se použije `select()`.

V podstatě jsou to podobné funkce, které mají na starosti I/O multiplexing. viz 2.2 Funkce `poll()` má výhodu v tom, že dokáže odchytnout situace, které `select()` nemůže. Pro více informací viz refrence [4]. Pomocí funkce `select()` zas můžeme snadněji spravovat více soketů.

Záleží tedy pouze na uživateli, programátorovi, jakou z těchto dvou funkcí zvolí.

4 Návod na použití programu

4.1 Spuštění programu

Program očekává povinný parametr `-p`, po němž následuje seznam portů, na které se pak připojuje. (viz kapitola 4.3 syntaxe portů)

Program vyžaduje ke své činnosti vstupní soubor, který obsahuje seznam adres. Je-li název souboru `-` (pomlčka), použije se standardní vstup. Syntaxe vstupního souboru je uvedena níže.

Nepovinným parametrem je `-t`. Slouží k zadání maximální doby v sekundách, po kterou bude program čekat na získání banneru. Nebyl-li parametr `-t` zadán, čeká se 10 sekund. Kdyby tato doba nebyla definována, v některých případech by program čekal nekonečně dlouho na banner. Může totiž nastat situace, kdy vzdálený počítač po spojení nezasílá banner. (Například HTTP server)

Dalším nepovinným a v zadaní nespecifikovaným parametrem je `-a`. Volá se výhradně v kombinaci s parametrem `-t`! Použije dannou dobu i pro připojení na danný server. Nebyl-li tento parametr zadán, čeká se většinou 75 sekund na spojení se vzdáleným počítačem. Což může být za některých situací nepřívětivé.

Při spuštění programu s parametrem `-h` se tiskne nápověda na standardní výstup.

Příklady spuštění programu:

```
$ ./tcpsearch -p 1-80,873 jmeno_vstupniho_souboru -t 1
$ ./tcpsearch -p 1-80,873 - -t 1 -a <vstup
```

4.2 Syntaxe vstupního souboru

```
<Doménové jméno | IPv4 | IPv6> <Vlastní komentář>
```

Např:

```
2001:67c:1220:8b0::93e5:b013 IPv6
147.229.13.162 IPv4
merlin.fit.vutbr.cz
www.fit.vutbr.cz skolni server
::1 localhost
```

4.3 Syntaxe portů

1. Port může být zadán jen jeden

   ```
   <port> = 1-65535
   ```

   Např:

   ```
   $ ./tcpsearch vstup -p 42
   ```

2. Nebo interval portů

   ```
   <interval> = <port>-<port>
   ```

   Např:

   ```
   $ ./tcpsearch vstup -p 1-80
   ```

3. Nebo výčtem jednotlivých portů nebo intervalů

   ```
   <výčet> = <port | interval>,<výčet>
   <výčet> = <port | interval>
   ```

   Např:

   ```
   $ ./tcpsearch vstup -p 4-9,42
   $ ./tcpsearch vstup -p 22,80,1024-1028,6666-6669
   ```

4.4 Výstup programu

Pro každý prohledávaný záznam vypíše program prohledávanou adresu a pokud bylo specifikováno, tak i doménové jméno. Pro každý otevřený port vypíše aplikace jeho číslo a obdržená data. Tyto informace bude aplikace vypisovat na standardní výstup.

Všechny chybové výpisy se vypisují na standardní chybový výstup.

Např:

```
$ ./tcpsearch -t 1 vstup -p 1-80,873
2001:67c:1220:8b0::93e5:b013
22
SSH-2.0-OpenSSH_4.3
```

5 Závěr

Výše uvedená metoda pro scanování portů není jediná, která by se dala použít. Můžeme použít lepší – TCP half scan, jiným názvem SYN scan. Klient zavolá funkci `connect()`, server mu při otevřeném portu odpoví ACK. Klient mu ovšem pošle RST. Čímž ze strany serveru se pak ukončí spojení a nemá tak velkou režii jak u `connect()`. Bohužel tento druh skenování nemůžeme použít v souvislosti s výpisem bannerů.

Rozšíření programu bych viděl v přidáním UDP protokolu nebo využitím proxy serveru. Při chybě, či spojení by se mohla vypisovat jména známých portů (well known ports). Program by mohl rovněž posílat data po získání určitého banneru.

Program přesně dodržuje požadavky kladené na formát vstupních a výstupních dat, takže může být bezproblémově používán spolu s dalšími programy ve skriptech nebo jiných programech.

Reference

[1] *Telnet(1) – Linux manual page. Man page* [online]. 2000 [cit. 2012-11-13]. Dostupné z: http://www.manpagez.com/man/1/telnet/

[2] *Netcat(1) - Linux manual page. Man page* [online]. 2001 [cit. 2012-11-13]. Dostupné z: http://www.manpagez.com/man/1/nc/

[3] STEVENS, W. Richard, FENNER Bill a RUDOFF Andrew M. *UNIX Network Programing: The Sockets Networking API*. 1. vyd. Boston: Addison-Wesley, 2004. 3. ISBN 0-13-141155-1.

[4] BOEGER, Nathan a TOMINAGA, Mana *FreeBSD system programming - Advenced I-O* [online]. 2001 [cit. 2012-11-13]. Dostupné z: http://www.khmere.com/freebsd_book/html/ch06.html