D0036D Nätverksprogrammering

Laboration 1

Pontus Stenlund

Ponste-5@student.ltu.se

Handledare: Örjan Tjernström

Contents

Del 1 - Lär känna protokollstacken med nätverkskommandon

Nätverkskommandon

Arp (Adress Resolution Protocol)

Används för att koppla samman IP-adress med en MAC-Adress. När ett paket tillägnat en maskin på ett nätverk kommer till en gateway, så frågar gatewayen ARP programmet att hitta en MAC-adress som matchar IP-adressen.

ARP arbetar både på Host-To-Network lagret, och internetlagret.

Ipconfig (Internet Protocol Configuration)

Ipconfig visar nuvarande TCP/IP nätverks konfigureringsvärden och med parametrar så kan ipconfig modifiera DHCP och DNS inställningar. Arbetar på internetlagret.

Ping

Skickar Internet Control Message Protocol (ICMP) paket till en host och väntar på ekot när det kommer fram. Ger information om hur lång tid det tar för paketet att komma fram.

Arbetar på internetlagret.

Route print

Används för att visa IP routing table i commandotolken. Arbetar på internetlagret.

Tracert

Används för att kolla vilken väg som ett paket tar till sin destination genom att sända ett ICMP paket till destinationen. Vägen som visas är routrar till destinationen. Visar även tiden det tar till de olika routrarna. Arbetar i internetlagret.

Nslookup (Name Server Lookup)

Används för att skicka förfrågningar till DNS servrar för att få DNS information, inklusive IP-adress till en viss dator. Informationen kan då användas för att diagnostisera DNS inframstrukturen. Arbetar i applikationslagret.

Netstat

Visar aktiva TCP anslutningar, portar som datorn lyssnar på, Ethernet statistik, IP routing table, IP statistik. Används för att felsöka nätverket och för att bestämma hur stor trafik nätverket har. Arbetar i transportlagret.

Datornamn/ip-inställningar

Jag använde mig av ipconfig /all för att få reda på datornamn, ip-adress, mac-adress etc.

Här är det jag har fått reda på.

```
Windows IP Configuration


   Host Name . . . . . . . . . . . . : B0992

   Primary Dns Suffix  . . . . . . . : ltuad.ltu.se

   Node Type . . . . . . . . . . . . : Hybrid

   IP Routing Enabled. . . . . . . . : No

   WINS Proxy Enabled. . . . . . . . : No

   DNS Suffix Search List. . . . . . : ltuad.ltu.se

                                       its.ltu.se

Ethernet adapter Ethernet:


   Connection-specific DNS Suffix  . : ltuad.se

   Description . . . . . . . . . . . : Realtek PCIe GBE Family Controller

   Physical Address. . . . . . . . . : DC-FE-07-13-A0-1B

   DHCP Enabled. . . . . . . . . . . : Yes

   Autoconfiguration Enabled . . . . : Yes

   IPv4 Address. . . . . . . . . . . : 130.240.52.70(Preferred)

   Subnet Mask . . . . . . . . . . . : 255.255.252.0

   Lease Obtained. . . . . . . . . . : den 31 augusti 2016 05:11:02

   Lease Expires . . . . . . . . . . : den 31 augusti 2016 17:11:02

   Default Gateway . . . . . . . . . : 130.240.52.1

   DHCP Server . . . . . . . . . . . : 130.240.19.2

   DHCPv6 IAID . . . . . . . . . . . : 81591815

   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-1F-56-FC-A7-DC-FE-07-13-A0-1B

   DNS Servers . . . . . . . . . . . : 130.240.82.81

                                       130.240.82.82

                                       130.240.82.83

   NetBIOS over Tcpip. . . . . . . . : Enabled
```

```
Tunnel adapter 6TO4 Adapter:

   Connection-specific DNS Suffix  . : ltuad.se
   Description . . . . . . . . . . . : Microsoft 6to4 Adapter
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2002:82f0:3446::82f0:3446(Preferred)
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 318767104
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-1F-56-FC-A7-DC-FE-07-13-A0-1B
   DNS Servers . . . . . . . . . . . : 130.240.82.81
                                       130.240.82.82
                                       130.240.82.83
   NetBIOS over Tcpip. . . . . . . . : Disabled
```

Finns det andra enheter på nätverket som du inte kan se eller känna till?

Ja det kan finnas enheter som vi inte kan se eller känna till. Enheten kan vara avstängd eller så kan brandväggen vara inställd på att inte svara på t.ex. ping.

Hur många datorer kan det finnas på det lokala subnätet?

Det kan finnas 1022 datorer. För att få fram det så tog gick jag in på http://subnet-calculator.samuraj-cz.com/ och fyllde i IP-adressen och nätmasken.

Del 2 – En inblick I applikationsprotokoll

HTTP

```
HTTP/1.1 200 OK

Date: Wed, 31 Aug 2016 12:01:21 GMT

Server: Apache/2

Last-Modified: Fri, 28 Aug 2015 07:27:01 GMT

ETag: "2158-51e5a027a1b40"

Accept-Ranges: bytes

Content-Length: 8536

Content-Type: text/html

Connection: close
```

```
<!-- saved from url=(0067)http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_m
akefiles.html -->

            <html><head><meta http-equiv="Content-Type" content="text/html;
charset=windows-1252"></head><body>

                          </p><p>This is an edited copy of the website
<a  href="http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html">"
http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html"</a>.

                                                      <h2>
Using make and writing Makefiles</h2>


                           make is a Unix tool to simplify building pr
ogram executables from many

                    modules.  make reads in rules (specified as a list o
f target entries)

            from a user created Makefile.  make will only re-build things t
hat

   need to be re-built (object or executables that depend on files that

                                                    have be
en modified since the last time the objects or executables were

                                             built).
```

</p><p><a href="http://www.gnu.org/software/make/manual/make.html">GNU make Manual</a>. A complete reference for writing makefiles from simple to

advanced features.

</p><p>For small projects, writing makefiles by hand

is easy. </p><p>

<a name="using">

</a></p><h3><a name="using">Using make</a></h3><a name="using">

<ol>

<li>Create a Makefile listing the rules for building the executable the

file should be named 'Makefile' or 'makefile'. This only has to

be done once, except when new modules are added to the program, the

Makefile must be updated to add new module dependencies to existing

rules and to add new rules to build the new modules.

</li><li> After editing program file(s), rebuild the executable by typing make:

<pre>% make

A specific target in the Makefile can be executed by typing:

% make target_label

For example, to execute the rm commands in the example makefile below, type:

% make clean

</pre>

</li></ol>

```
                              </a><a name="creating">
                                        <h3>Creating
a Makefile</h3>

      A Makefile typically starts with some variable definitions which
are
 then followed by a set of target entries for building specific targets
                                           (typic
ally .o &amp; executable files in C and C++, and .class files in Java)
                                         or execut
ing a set of command associated with a target label.
                                  <p>
                                   The following is the ge
neric target entry form:
           </p><pre># comment
                             # (note: the &lt;tab&gt; in the comma
nd line is necessary for make to work)
                           target:  dependency1 dependency2 ...

&lt;tab&gt; command


        for example:
             #
              # target entry to build program executable from
program and mylib
        # object files
              #
               program: program.o mylib.o
                                     gcc -o program p
rogram.o mylib.o
       </pre>



         </a><a name="C">
                         <h3> Example simple Makefiles for a C (or
C++)</h3>
```

The most simple Makefile for compiling a C (or C++) program

from a single .c file, with make and make clean rules,

looks something like this:

<pre> # build an executable named myprog from myprog.c

all: myprog.c

gcc -g -Wall -o myprog myprog.c

clean:

$(RM) myprog

</pre>

A slightly more generic version using makefile variables

(just change the variable definitions to build differnet executables

or with differnt compilers or compiler flags):

<pre> # the compiler: gcc for C program, define as g++ for C++

CC = gcc

# compiler flags:

# -g adds debugging information to the executable file

# -Wall turns on most, but not all, compiler warnings

CFLAGS = -g -Wall

# the build target executable:

TARGET = myprog

all: $(TARGET)

```
$(TARGET): $(TARGET).c
$(CC) $(CFLAGS) -o $(TARGET) $(TARGET).c

clean:
$(RM) $(TARGET)
</pre>
```

An example of building an executable from multiple .o files:

```
<pre>#
# This is an example Makefile for a countwords program.  This
# program uses both the scanner module and a counter module.
# Typing 'make' or 'make count' will create the executable file.
#

# define some Makefile variables for the compiler and compiler flags
# to use Makefile variables later in the Makefile: $(<var_name>)
#
#  -g    adds debugging information to the executable file
#  -Wall turns on most, but not all, compiler warnings
#
# for C++ define  CC = g++
CC = gcc
CFLAGS  = -g -Wall

# typing 'make' will invoke the first target entry in the file
# (in this case the default target entry)
```

```
                                                   # you
can name this target entry anything, but "default" or "all"
                                         # are the most commo
nly used names by convention
                   #
                    default: count

                             # To create the executable file count
we need the object files
                 # countwords.o, counter.o, and scanner.o:
                                             #
                                              count:  count
words.o counter.o scanner.o
                      $(CC) $(CFLAGS) -o count countwords.o counter.o
scanner.o

      # To create the object file countwords.o, we need the source
                                               # files cou
ntwords.c, scanner.h, and counter.h:
                         #
                         countwords.o:  countwords.c scanner.h count
er.h
     $(CC) $(CFLAGS) -c countwords.c

                           # To create the object file counter.o, we
need the source files
               # counter.c and counter.h:
                                 #
                                  counter.o:  counter.c counter.h
     $(CC) $(CFLAGS) -c counter.c

                         # To create the object file scanner.o, we ne
ed the source files
             # scanner.c and scanner.h:
                               #
                               scanner.o:  scanner.c scanner.h
```

```
                                                            $
(CC) $(CFLAGS) -c scanner.c


                    # To start over from scratch, type 'make clean'.  Thi
s
 # removes the executable file, as well as old .o object
                                           # files and *~ backup fi
les:
   #
    clean:
            $(RM) count *.o *~
                          </var_name></pre>
                                       </a><a name="adv">
                                                    <h3> Anothe
r makefile (using makedepend and more advanced make syntax)</h3>
                                                This is an easie
r to use and modify makefile, but it is slightly more
                                          difficult to read than the
simple one:
         <pre>#
              # 'make depend' uses makedepend to automatically generate depe
ndencies
      #            (dependencies are added to end of Makefile)
                                                    # 'make'
build executable file 'mycc'
                        # 'make clean'  removes all .o and executable fi
les
  #


   # define the C compiler to use
                          CC = gcc


                                # define any compile-time flags
                                                        CFLAGS
= -Wall -g
```

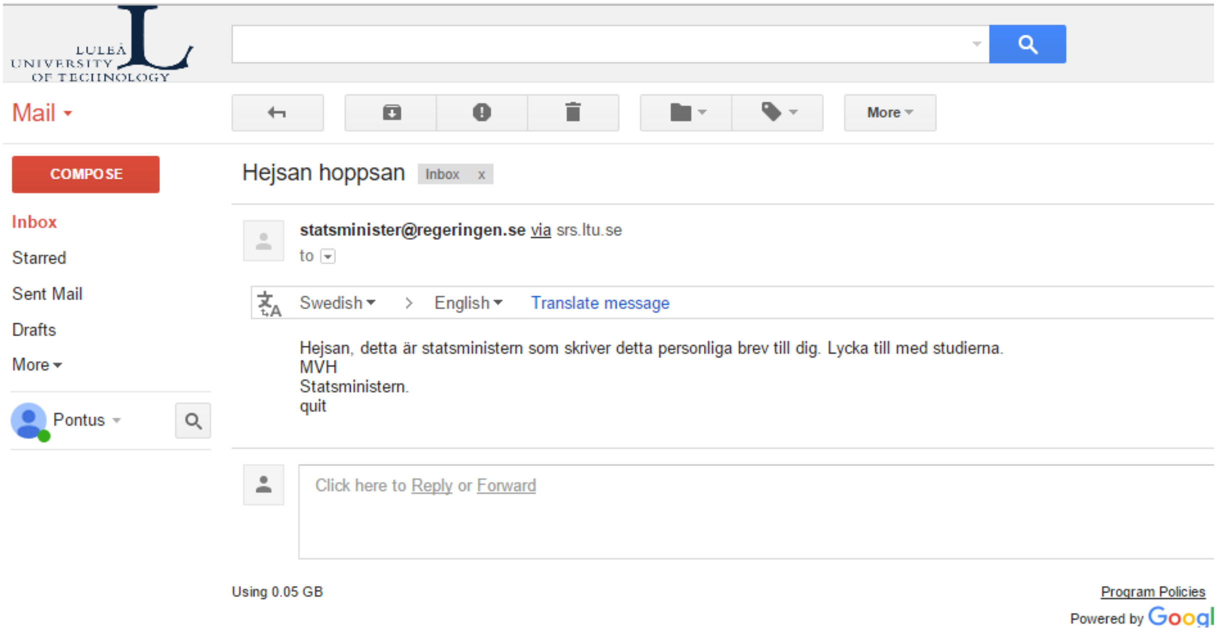
```
# define any directories containing header files other than /usr/include
#
INCLUDES = -I/home/newhall/include  -I../include

# define library paths in addition to /usr/lib
#   if I wanted to include libraries not in /usr/lib I'd specify
#   their path using -Lpath, something like:
LFLAGS = -L/home/newhall/lib  -L../lib

# define any libraries to link into executable:
#   if I want to link in libraries (libx.so or libx.a) I use the -llibname
#   option, something like (this will link in libmylib.so and libm.so:
LIBS = -lmylib -lm

# define the C source files
SRCS = emitter.c error.c init.c lexer.c main.c symbol.c parser.c

# define the C object files
#
# This uses Suffix Replacement within a macro:
#   $(name:string1=string2)
#         For each word in 'name' replace 'string1' with 'string2'
# Below we are replacing the suffix .c of all words in the macro SRCS
# with the .o suffix
#
```

```
                                                            O
BJS = $(SRCS:.c=.o)


           # define the executable file
                            MAIN = mycc


                                  #
                                  # The following part
of the makefile is generic; it can be used to
                            # build any executable just by ch
anging the definitions above and by
                    # deleting dependencies appended to the file
from 'make depend'
           #


           .PHONY: depend clean


                      all:    $(MAIN)
                                     @echo  Simple comp
iler named mycc has been compiled


                   $(MAIN): $(OBJS)
                                 $(CC) $(CFLAGS) $(INCL
UDES) -o $(MAIN) $(OBJS) $(LFLAGS) $(LIBS)


                         # this is a suffix replacement rule fo
r building .o's from .c's
               # it uses automatic variables $&lt;: the name of the pr
erequisite of
        # the rule(a .c file) and $@: the name of the target of the rule (a
.o file)
      # (see the gnu make manual section about automatic variables)
                                                       .c.o:

$(CC) $(CFLAGS) $(INCLUDES) -c $&lt;  -o $@
```

```
                                 clean:
                                          $(RM) *.o *~ $(MAIN
)

 depend: $(SRCS)
                 makedepend $(INCLUDES) $^

                                  # DO NOT DELETE THIS LINE -- ma
ke depend needs it
            </pre>

                 </a><a name="java">
                              <h3> An example simple Makefile for a
Java</h3>
       <pre>#
           # A simple makefile for compiling three java classes
                                               #

                                                # define a
makefile variable for the java compiler
                           #
                            JCC = javac

                                   # define a makefile variable
for compilation flags
               # the -g flag compiles with debugging information
                                                 #
                                                 JFLAGS =
-g

 # typing 'make' will invoke the first target entry in the makefile
                                                # (the defa
ult one in this case)
               #
               default: Average.class Convert.class Volume.class
```

```
                                                # this ta
rget entry builds the Average class
                              # the Average.class file is dependent on the
Average.java file
                    # and the rule associated with this entry gives the command to
create it
          #
           Average.class: Average.java
                                          $(JCC) $(JFLAGS) Average.java


                                                  Conver
t.class: Convert.java
                              $(JCC) $(JFLAGS) Convert.java


                                            Volume.class: Volume.j
ava
            $(JCC) $(JFLAGS) Volume.java


                                # To start over from scratch, type 'make
clean'.
          # Removes all .class files, so that the next make rebuilds them
                                                  #
                                                   clean:

$(RM) *.class
                    </pre>
                         </a><a



                                 </body></html>
```

SMTP



Hejsan hoppsan

statsminister@regeringen.se via srs.ltu.se
to

Hejsan, detta är statsministern som skriver detta personliga brev till dig. Lycka till med studierna.
MVH
Statsministern.
quit

Del 3 - Användning av verktyg för nätverksprotokollanalys

Reflektion: Man kan använda wireshark för att se vad ens program skickar för data och tar emot för data. Detta är användbart för att förhindra att programmet skickar ut känslig information eller att en användare kan skicka data som skadar programmet/datorn.