情報科学演習 C レポート 1

藤田 勇樹
大阪大学 基礎工学部 情報科学科 ソフトウェア科学コース
学籍番号: 09B16068
メールアドレス: u461566g@ecs.osaka-u.ac.jp
担当教員
小島 英春 助教授
内山 彰 助教授

提出日: 2018 年 4 月 22 日

1　課題 1-1

1.1　ping コマンド

ping コマンドを実行すると，以下のような出力が得られる．

```
$ ping exp101
PING exp101.exp.ics.es.osaka-u.ac.jp (192.168.16.65) 56(84) bytes of data.
64 bytes from exp101.exp.ics.es.osaka-u.ac.jp (192.168.16.65): icmp_seq=1 ttl=64 time=0.134 ms
64 bytes from exp101.exp.ics.es.osaka-u.ac.jp (192.168.16.65): icmp_seq=2 ttl=64 time=0.180 ms
64 bytes from exp101.exp.ics.es.osaka-u.ac.jp (192.168.16.65): icmp_seq=3 ttl=64 time=0.194 ms
64 bytes from exp101.exp.ics.es.osaka-u.ac.jp (192.168.16.65): icmp_seq=4 ttl=64 time=0.209 ms
64 bytes from exp101.exp.ics.es.osaka-u.ac.jp (192.168.16.65): icmp_seq=5 ttl=64 time=0.185 ms

--- exp101.exp.ics.es.osaka-u.ac.jp ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.134/0.180/0.209/0.027 ms
```

オンラインマニュアルで ping について調べたところ，以下の事柄がわかった．ping コマンドでは，引数に指定されたホストに ICMP プロトコルの ECHO_REQUEST を送り，それが正しく送られたか，またそれにかかった時間を出力する．ICMP プロトコルとは，マシンの状態やエラーメッセージなどを送受信するプロトコルで，ECHO_REQUEST は ICMP でやりとりするメッセージの一種である．

実際の出力の意味を説明する．

```
64 bytes from exp101.exp.ics.es.osaka-u.ac.jp (192.168.16.65): icmp_seq=1 ttl=64 time=0.134 ms
```

これは，64 バイトのパケット 1 個目を exp101(IP アドレス 192.168.16.65) に TTL=64 で送り，応答を受け取るまで 0.134 ミリ秒かかったということである．TTL とは Time to Live の略で，パケットがあるノードから別のノードに受け渡される回数の上限を表す．パケットがノードから送信されるたびに TTL は 1 ずつ減り，0 になるとそのパケットは宛先に辿りつけなかったものとして破棄される．

しかし，yahoo.com などの外部ホストに ping を実行したところパケットが届かなかった．これについては 2.3 で考察する．

1.2　ドメイン名と IP アドレス

http://www-higashi.ist.osaka-u.ac.jp/ と http://133.1.17.66/ にアクセスしたところ，全く同じページが表示された．そのため，www-higashi.ist.osaka-u.ac.jp と 133.1.17.66 は同じものを表すと思われる．

1.3 nslookup コマンド

nslookup を man コマンドで調べたところ，nslookup はホスト名に対応する IP アドレスを調べるコマンドであった．実際に，`www-higashi.ist.osaka-u.ac.jp` を実行すると，1.2 節の通り `133.1.17.66` が得られた．また土屋研 `www-ise4.ist.osaka-u.ac.jp` に対応するのは `133.1.16.2` であった．

2 課題 1-2

2.1 arp コマンド

ARP は，IP アドレスがわかっているホストに MAC アドレスを問い合わせるプロトコルである．また，ARP テーブルとは，過去の通信で ARP プロトコルを使用した際に得られた IP アドレスと MAC アドレスの対応をキャッシュとして保存したものである．`arp` コマンドでは ARP テーブルを表示する．実際に `arp -a` コマンドを実行すると以下のような出力が得られる．

```
exp029.exp.ics.es.osaka-u.ac.jp (192.168.16.62) at 00:50:56:b7:0d:47 [ether] on ens192
cups.exp.ics.es.osaka-u.ac.jp (192.168.16.253) at 00:50:56:b7:5b:4b [ether] on ens192
exp036.exp.ics.es.osaka-u.ac.jp (192.168.16.52) at 00:50:56:b7:59:b6 [ether] on ens192
? (192.168.16.254) at 14:18:77:10:31:aa [ether] on ens192
dhcp-01.exp.ics.es.osaka-u.ac.jp (192.168.16.240) at 00:50:56:b7:21:6e [ether] on ens192
svm-01.exp.ics.es.osaka-u.ac.jp (192.168.16.241) at 02:a0:98:c4:b2:cf [ether] on ens192
```

exp029 の端末で `ifconfig` で MAC アドレスを確認してもらうと，確かに同じ MAC アドレスであった．

2.2 ping 後の arp コマンド

`ping exp092` を行った後に再び arp コマンドを実行すると，以下の行が追加されていた．

```
exp092.exp.ics.es.osaka-u.ac.jp (192.168.16.18) at 00:50:56:b7:79:90 [ether] on ens192
```

このことから，ARP テーブルには過去の通信で得られた IP アドレスと MAC アドレスの対応が自動で格納されることがわかる．

2.3 traceroute コマンド

www.ics.es.osaka-u.ac.jp と icsintgw.ics.es.osaka-u.ac.jp にそれぞれ `traceroute` を実行すると，どちらも以下のような出力が得られた．

```
traceroute to icsintgw.ics.es.osaka-u.ac.jp (133.1.240.81), 30 hops max, 60 byte packets
 1  192.168.16.254 (192.168.16.254)  0.375 ms  0.446 ms  0.508 ms
 2  icsintsvgw.ics.es.osaka-u.ac.jp (133.1.240.254)  0.720 ms  0.882 ms  1.006 ms
 3  icsintgw.ics.es.osaka-u.ac.jp (133.1.240.81)  1.777 ms  2.392 ms  9.386 ms
```

ただし，icsintgw.ics.es.osaka-u.ac.jp は終了したが，www.ics.es.osaka-u.ac.jp は icsintgw.ics.es.osaka-

u.ac.jp まで辿った後，応答がないことを表す “* * *” が表示され続けた．

演習室の環境では http/https でしか外部と通信できないが，`traceroute` ではオンラインマニュアルによると ICMP を使用しているため，ゲートウェイ icsintgw.ics.es.osaka-u.ac.jp で通信が止められているものと考えられる．1.1 節で yahoo.com などの適当なドメインへの `ping` が届かなかったのも同様の理由のためであると思われる．

2.4 演習室のネットワーク構成

前節までの結果より，演習室のネットワーク構成は以下の図 1 のようになっていると思われる．

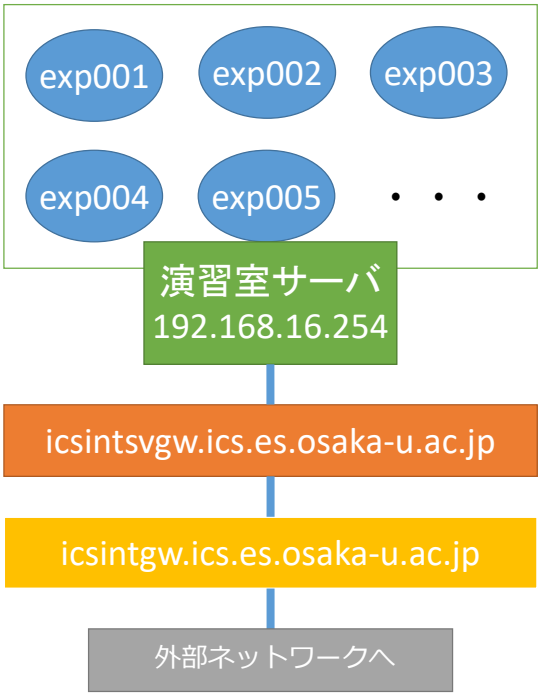

図 1　演習室のネットワーク構成の予想

2.5 netstat コマンド

`netstat -r` を実行すると，以下のような出力が得られた．

```
$ netstat -r
カーネル IP 経路テーブル
受信先サイト     ゲートウェイ     ネットマスク     フラグ   MSS Window  irtt インタフェース
default         192.168.16.254  0.0.0.0         UG        0 0          0 ens192
link-local      *               255.255.0.0     U         0 0          0 ens192
192.168.16.0    *               255.255.255.0   U         0 0          0 ens192
```

オンラインマニュアルによると，`netstat -r` コマンドでは，カーネルのルーティングテーブルを出力す

る．ルーティングテーブルとは，受け取ったパケットの宛先に応じて，次にどのノードにパケットを転送すればよいかを記録してあるものである．例えば，受け取ったパケットの宛先が 133.1.17.66 であれば，一番上の 192.168.16.254 に転送する．

2.6 再び arp コマンド

前節までの作業を行った次の週に `arp -a` コマンドを実行すると，演習室の端末の ARP テーブルが消えていた．このことから，自動的に追加され長時間使用していない ARP テーブルは削除されることがわかる．

3 課題 1-3

3.1 標準ライブラリ関数とシステムコールの違い

C 言語の標準ライブラリ関数とシステムコールの違いについて述べる．課題での例では，fwrite() と write() はともにファイルへの書き込みを行う関数であるが，fwrite() はどの環境でも使用できる一方，write() は POSIX 環境でのみ使用でき，Windows などでは使用できない．ただし，標準ライブラリ関数は環境に依存しない方法であるが，その実装にはシステムコールが使用されているため，生のシステムコールに比べてオーバーヘッドが生じる場合がある．

3.2 strace コマンド

```
$ strace -c echo hello
```

を実行したところ，以下の出力が得られた．

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         1           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0         3           open
  0.00    0.000000           0         5           close
  0.00    0.000000           0         4           fstat
  0.00    0.000000           0         8           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         3         3 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    35         3 total
```

ここでは，“`echo hello`” を実行し，どのシステムコールが何回呼ばれたかを calls の列に出力している．

また，それぞれのシステムコールの種類ごとにかかった時間とその割合も出力される．

また，“ls” では以下の出力が得られた．出力される calls の数はディレクトリによって異なる場合がある．

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         7           read
  0.00    0.000000           0         5           write
  0.00    0.000000           0         9           open
  0.00    0.000000           0        11           close
  0.00    0.000000           0        10           fstat
  0.00    0.000000           0        19           mmap
  0.00    0.000000           0        12           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         2           rt_sigaction
  0.00    0.000000           0         1           rt_sigprocmask
  0.00    0.000000           0         2           ioctl
  0.00    0.000000           0         7         7 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2           getdents
  0.00    0.000000           0         1           getrlimit
  0.00    0.000000           0         2         2 statfs
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    98         9 total
```

echo hello に比べ，いくつかのシステムコールが追加されているほか，open,close などファイル操作の回数が増えている．しかし，“ls” コマンドはファイルの内容を調べるものではないため，他の要因で open や close の回数が増えていると考えた．そこで，open で何を開いているかを調べるため以下のコマンドを実行した．

```
strace ls 2>&1 1>/dev/null | grep open
```

オプションなしの strace コマンドは呼び出されたシステムコールを逐一出力する．また，strace コマンドの出力は調べる対象のコマンドの出力と被らないようにするためか標準エラー出力に出力されている．そのため標準エラー出力のみから “open” のある行を取り出している．このコマンドを実行すると，以下の出力が得られた．

```
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
```

```
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libpcre.so.3", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
open("/proc/filesystems", O_RDONLY)     = 3
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
open(".", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 3
```

ここで，libc.so.6 などの/lib 配下のファイルは，標準ライブラリ関数の実装であり，普段 C 言語で標準ライブラリを使用した際には，実際にはこれらのファイルの内容が実行される．比較のため，“`echo hello`” でも同様の調査を行った．

```
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
```

これらを比較すると，`ls` の方では使用しているライブラリが増えているほか，/proc/filesystems などファイルシステム関連の情報を参照していることがわかる．また，カレントディレクトリを開くのにも open を使用している．

また，“`ping exp001`” では以下の出力が得られた．

```
ping: icmp open socket: Operation not permitted
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0        16           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0        15           open
  0.00    0.000000           0        20           close
  0.00    0.000000           0         2           stat
  0.00    0.000000           0        15           fstat
  0.00    0.000000           0         2           poll
  0.00    0.000000           0        23           mmap
  0.00    0.000000           0        14           mprotect
  0.00    0.000000           0         4           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         1           ioctl
  0.00    0.000000           0         9         9 access
  0.00    0.000000           0         1           dup
  0.00    0.000000           0         1           getpid
  0.00    0.000000           0         5         1 socket
  0.00    0.000000           0         4         2 connect
  0.00    0.000000           0         1           sendto
```

```
  0.00    0.000000           0         1           recvfrom
  0.00    0.000000           0         1           getsockname
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         4           fcntl
  0.00    0.000000           0         2           getuid
  0.00    0.000000           0         1           setuid
  0.00    0.000000           0         1           geteuid
  0.00    0.000000           0         7           capget
  0.00    0.000000           0         1           capset
  0.00    0.000000           0         2           prctl
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   159        12 total
```

echo の出力に比べ，socket(2) や connect(2) などのネットワーク関連のシステムコールが加わっていることがわかる．

4 発展課題

4.1 ネットワーク関連のコマンド

4.1.1 telnet

telnet コマンドは，他のマシンの遠隔操作を行うためのコマンドである．引数に指定した IP アドレスのマシンに接続する．`telnet 127.0.0.1` を実行しパスワードを入力すると Welcome to Ubuntu 16.04.4 LTS ... 等のメッセージが表示された後，通常通りの端末操作が可能となった．また友人のマシンにアクセスしパスワードを入力してもらうと，同様にそのマシンに対しての端末操作が可能となった．

4.1.2 rsh

rsh コマンドは，他のマシンに接続し引数に指定したコマンドを実行するコマンドである．`rsh exp029 ls` を実行しパスワードを入力してもらうと，exp029 のホームディレクトリの内容が出力された．

4.1.3 ftp

ftp コマンドは，FTP サーバに接続しファイルのやり取りをするコマンドである．引数か起動後に open コマンドで FTP サーバのアドレスを指定して接続する．しかし FTP サーバの所在が不明だったため使用できなかった．

finger コマンドおよび talk コマンドは演習室の環境ではインストールされていないため使用できずマニュアルも表示できなかった．

4.2 hello100 回

以下のプログラムを，コメントアウトを切り替えて実行し，strace -c を用いてシステムコールの呼び出し回数を比較する．

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define COUNT 100
int main(int ac, char* av[]){
    int i;
        // [下記 2 行の内、いずれかをコメントアウトする]
        char message[] = "hello";
        // char message[] = "hello\n";
    for(i = 0; i < COUNT; i ++){
        // [下記 2 行の内、いずれかをコメントアウトする]
        fwrite(message, strlen(message), 1, stdout);
        // write(0, message, strlen(message));
    }
    return 0;
}
```

以下の表 1 にそれぞれのプログラムでの strace の出力を述べる．ただし，ここではシステムコールの回数である calls の部分のみ載せる．

表 1　strace の出力

syscall	改行なし,fwrite	改行なし,write	改行あり,fwrite	改行あり,write
read	1	1	1	1
write	1	100	100	100
open	2	2	2	2
close	2	2	2	2
fstat	3	2	3	2
mmap	7	7	7	7
mprotect	4	4	4	4
munmap	1	1	1	1
brk	3	1	3	1
access	3	3	3	3
execve	1	1	1	1
arch_prctl	1	1	1	1

注目すべきは，改行なし,fwrite でのみ write が 1 回しか呼ばれていない点である．改行あり,fwrite では write が 100 回呼ばれていることを踏まえると，fwrite では改行を出力文字として受け取るまでは出力する文字はバッファリングしておき，改行を受け取るかファイルストリームが閉じられる時に出力すると考えられる．

詳しく調べてみると，このバッファリングの方針は setvbuf() 関数で変更できることがわかった．そこで，この関数を追加し_IONBF(バッファリングを一切しない) を設定すると，改行なし,fwrite でも write が 100 回呼び出されるようになった．

この他の違い (fstat と brk の数) はどちらも fwrite と write のどちらを使うかで異なっているので，fwrite の処理の一環として呼び出されていると思われる．