課題 No.3 の解答例

1. パスとハードリンク

   (a) 自分のホームディレクトリのパスを調べる．
   使用開始時のカレントディレクトリはホームディレクトリなので，pwd コマンドで確認できる．

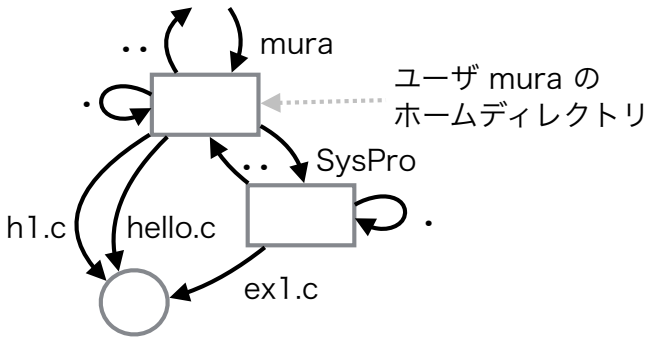
```
$ pwd
/Network/Servers/iemac.ie.tokuyama.ac.jp/Volumes/Users/i0/sigemura
```

   (b) /tmp にカレントディレクトリを移動させる．
   cd コマンドで移動し，結果を pwd コマンドで確認する．

```
$ cd /tmp
$ pwd
/tmp
```

   (c) /tmp 以下に，図のディレクトリやファイルを作る．

   (d) ls -l を用いてファイルの種類やリンク数を確認する．




```
$ mkdir mura
$ cd mura
$ emacs hello.c
$ cat hello.c
#include <stdio.h>
int main() { printf("hello\n"); }
$ ln hello.c h1.c
$ mkdir SysPro
$ ls -l
total 16
drwxr-xr-x  2 sigemura  wheel  68 May  2 16:24 SysPro
-rw-r--r--  2 sigemura  wheel  53 May  2 16:24 h1.c
-rw-r--r--  2 sigemura  wheel  53 May  2 16:24 hello.c
$ ln hello.c SysPro/ex1.c
$ ls -l SysPro/
total 8
-rw-r--r--  3 sigemura  wheel  53 May  2 16:24 ex1.c
```

(e) 相対パスを用いて hello.c の内容を表示する.

hello.c はカレントディレクトリにあるので簡単だ.

```
$ cat hello.c
#include <stdio.h>
int main() { printf("hello\n"); }
```

(f) 絶対パスを用いて hello.c の内容を表示する.

hello.c はカレントディレクトリにあるので，カレントディレクトリの絶対パスを確認して，それに付け加えると簡単にできる.

```
$ pwd
/tmp/mura
$ cat /tmp/mura/hello.c
#include <stdio.h>
int main() { printf("hello\n"); }
```

(g) h1.c を利用した相対パスを用いて hello.c の内容を表示する.

h1.c は hello.c のリンクなので，単に次のようにすれば良い.

```
$ cat h1.c
#include <stdio.h>
int main() { printf("hello\n"); }
```

(h) ex1.c を利用した相対パスを用いて hello.c の内容を表示する.

ex1.c は SysPro ディレクトリにある hello.c のリンクなので，次のようにすれば良い.

```
$ cat SysPro/ex1.c
#include <stdio.h>
int main() { printf("hello\n"); }
```

(i) カレントディレクトリを SysPro に変更する.

cd コマンドで移動し，結果を pwd コマンドで確認する.

```
$ cd SysPro
$ pwd
/tmp/mura/SysPro
```

(j) ex1.c を利用した相対パスを用いて hello.c の内容を表示する.

ex1.c はカレントディレクトリにあるので簡単だ.

```
$ cat ex1.c
#include <stdio.h>
int main() { printf("hello\n"); }
```

(k) hello.c を利用した相対パスを用いて hello.c の内容を表示する.

カレントディレクトリは/tmp/mura/SysPro なので，/tmp/mura/hello.c を特定する相対パスは../hello.c になる.

```
$ cat ../hello.c
#include <stdio.h>
int main() { printf("hello\n"); }
```

(l) 課題のために作成したファイルやディレクトリを全て削除する．

```
$ cd ../..
$ rm mura/SysPro/ex1.c
$ rmdir mura/SysPro
$ rm mura/hello.c
$ rm mura/h1.c
$ rmdir mura
```

(m) ディレクトリのハードリンクができるか試す．
試すとエラーが発生しリンクできないことが分かる．
（実は，ファイル木が複雑になりすぎるので禁止されている．）

```
$ mkdir DIR
$ ln DIR dir
ln: DIR: Is a directory
```

2. シンボリックリンク

(a) シンボリックリンクを作ってみる．

(b) シンボリックリンクを `ls -l` を用いて確認する．

(c) シンボリックリンクを用いてファイルをアクセスできることを確認する．
以下の作業は `mura` ディレクトリで行うものとする．`hello.c` は予め `mura` ディレクトリに作ってあるとする．

```
$ cat hello.c
#include <stdio.h>
int main() { printf("hello\n"); }
$ ln -s hello.c h1.c                          <--シンボリックリンクを作る
$ ls -l
total 16
lrwxr-xr-x 1 sigemura  staff   7 May 12 15:53 h1.c -> hello.c
-rw-r--r-- 1 sigemura  staff  55 May 12 15:52 hello.c
$ cat h1.c                                                  <--使用できる
#include <stdio.h>
int main() { printf("hello\n"); }
```

(d) リンク切れのシンボリックリンクを使用するとどうなるか確認する．

```
$ ln -s helllo.c h2.c                          <--ファイル名を間違った
$ ls -l                                          (エラーにはならない)
total 24
lrwxr-xr-x 1 sigemura  staff   7 May 12 15:53 h1.c -> hello.c
lrwxr-xr-x 1 sigemura  staff   8 May 12 15:59 h2.c -> helllo.c   <--!!
-rw-r--r-- 1 sigemura  staff  55 May 12 15:52 hello.c
```

```
$ cat h2.c
cat: h2.c: No such file or directory          <--使用した時点でエラー
```

(e) ディレクトリに対するリンクを作って使用できることを確認する.

```
$ rm h1.c h2.c                    <--もう使用しないので消す
$ mkdir ../sige                   <--muraディレクトリの隣にsigeを作る
$ mv hello.c ../sige              <--hello.cを../sige/hello.c に移動
$ ln -s ../sige sigeLink          <--sigeディレクトリへのリンクを作る
$ ls -l                           <--リンクができたか確認する
total 8
lrwxr-xr-x 1 sigemura  staff  7 May 12 17:31 sigeLink -> ../sige
$ ls ../sige
hello.c
$ ls sigeLink                     <--../sige の内容が見える
hello.c
```

(f) 他のディレクトリにあるファイルをリンクして使用できることを確認する.

```
$ ln -s ../sige/hello.c ex1.c     <--hello.cへのリンクを作る
$ cat ex1.c                       <--リンクを使ってアクセスできる
#include <stdio.h>
int main() { printf("hello\n"); }
$ cat sigeLink/hello.c            <--ディレクトリへのリンクを組合せても
#include <stdio.h>                    アクセスできる
int main() { printf("hello\n"); }
```

(g) シンボリックリンクのループを作る.

```
$ ln -s b.txt a.txt
$ ln -s a.txt b.txt
$ ls -l
total 16
lrwxr-xr-x 1 sigemura  staff   5 May 12 20:21 a.txt -> b.txt
lrwxr-xr-x 1 sigemura  staff   5 May 12 20:21 b.txt -> a.txt
```

(h) ループしているシンボリックリンクを使用するとどうなるか確認する.

```
$ cat a.txt
cat: a.txt: Too many levels of symbolic links  <--エラーになる
```

エラーメッセージ (Too many levels of symbolic links) から, シンボリックリンクをたどる回数に制限を設けていることが想像できる. 何回までなら大丈夫なのか確かめてみよう. (シンボリックリンクを作るために 1 行スクリプトを書いた)

```
$ j=0; i=0; while [ $i -ne 40 ]; do i=`expr $i + 1`; \
  eval `printf "ln -s %02d.txt %02d.txt" $i $j`; j=$i; done
$ ls -l
total 320
lrwxr-xr-x  1 sigemura  staff  6 May 12 20:33 00.txt -> 01.txt
lrwxr-xr-x  1 sigemura  staff  6 May 12 20:33 01.txt -> 02.txt
lrwxr-xr-x  1 sigemura  staff  6 May 12 20:33 02.txt -> 03.txt
```

```
... 途中省略 ...
lrwxr-xr-x  1 sigemura  staff  6 May 12 20:33 39.txt -> 40.txt
$ echo 40 > 40.txt
$ cat 00.txt
cat: 00.txt: Too many levels of symbolic links
$ cat 01.txt
cat: 01.txt: Too many levels of symbolic links
$ cat 02.txt
cat: 02.txt: Too many levels of symbolic links
... 途中省略 ...
$ cat 07.txt
cat: 07.txt: Too many levels of symbolic links
$ cat 08.txt
40                                  <--エラーが発生しない！！
```

07.txt から 39.txt まで 33 回シンボリックリンクをたどる時は「たどる回数が多すぎて」エラーになる．しかし，08.txt から 39.txt まで 32 回シンボリックリンクをたどる時は 40.txt の内容が正常に表示された．macOS では一度のパス解析中に，最高 32 回[1]シンボリックリンクをたどることができる[2]．

3. 保護モード

(a) テキストファイル (hello.c) と実行可能ファイル (a.out) を準備する．
hello.c は予め作ってあるとする．

```
$ cc hello.c
$ ls -l
total 32
-rwxr-xr-x 1 sigemura  staff  8432 May 12 21:13 a.out
-rw-r--r-- 1 sigemura  staff    53 May 12 21:12 hello.c
```

(b) chmod で保護モードを変化してみる[3]．

(c) ls -l で変化を確認する．

```
$ chmod u-w hello.c
$ ls -l hello.c
-r--r--r--  1 sigemura  staff    53 May 12 21:12 hello.c
$ chmod u+w hello.c
$ ls -l hello.c
-rw-r--r--  1 sigemura  staff    53 May 12 21:12 hello.c
$ chmod 400 hello.c
$ ls -l hello.c
-r--------  1 sigemura  staff    53 May 12 21:12 hello.c
$ chmod 644 hello.c
$ ls -l hello.c
-rw-r--r--  1 sigemura  staff    53 May 12 21:12 hello.c
```

[1] macOS のバージョン毎に違うかもしれない．
[2] シンボリックリンクのループを検出することが難しいので，このような対策がされている．
[3] OS のバージョンによって細部が異なる可能性がある．

(d) 保護モードを変化させてファイルの読出しや実行ができるか確認する．

```
$ ./a.out                                  <-- a.out は実行できる
hello
$ chmod u-x a.out                          <-- 所有者の x を消すと
$ ./a.out
-bash: ./a.out: Permission denied          <-- a.out は実行でなくなる
$ chmod u+x a.out                          <-- 所有者の x を復活すると
$ ./a.out
hello                                      <-- a.out が実行できる
$ cat hello.c
#include <stdio.h>                         <-- hello.c を読出しできる
int main() { printf("hello\n"); }
$ chmod u-r hello.c                        <-- 所有者の r を消すと
$ cat hello.c
cat: hello.c: Permission denied            <-- hello.c を読出しできない
$ chmod u+r hello.c                        <-- 所有者の r を復活すると
$ cat hello.c
#include <stdio.h>                         <-- hello.c を読出しできる
int main() { printf("hello\n"); }
```

(e) ディレクトリの保護モードは何の意味を持つか考える．

ディレクトリ（dir）の保護モードを変更しながら，ファイルの作成・表示・削除を試す．

```
$ mkdir dir                                <-- ディレクトリ dir を作る
$ ls -ld dir                               <-- dirの中ではなくdir自体
drwxr-xr-x 2 sigemura  staff   68 May  15 09:28 dir
$ echo aaaa > dir/a.txt                    <-- dir にファイルを作る
$ ls -l dir                                <-- dir の中を確認
-rw-r--r-- 1 sigemura  staff    5 May  15 09:33 a.txt
$ cat dir/a.txt                            <-- dir のファイルを表示
aaaa
$ rm dir/a.txt                             <-- dir のファイルを消す
```

保護モードの意味は以下の通りだと分かるはず．

保護モード	意味
r	ディレクトリからリンク名一覧を読み出せる
w	ディレクトリに変更ができる
x	ディレクトリ以下に入って行ける (ディレクトリに cd できる，ディレクトリ以下のファイルが開ける，ディレクトリ以下のファイルの属性情報が読める等)