計算機プログラミングI 講義ノート #2

担当：井上 純一 (情報科学研究科棟 8-13), 赤間 清 (情報基盤センター)
URL：http://chaosweb.complex.eng.hokudai.ac.jp/~j_inoue/PROG2007/PROG2007.html

平成 19 年 4 月 13 日

目 次

6 ファイルシステムとディレクトリ構造

今回は前回の講義で皆さんがアカウントを取得し, これから半年間使っていくことになる計算機室の Linux システムにおける [ファイル] と [ディレクトリ] の概念を詳しく学んで行きます. また, これらを操作するコマンド (命令) を端末からキーボード入力してもらうことで, 計算機がそれらコマンドに対してどのような応答をするのかを実際に確認してもらいます. ある程度コンピュータに慣れた人であれば, この配布資料を読むだけでも何となくわかった気になりますが, はじめは面倒がらず, 必ず資料に書かれたことを逐次入力し, その動作を自分で確かめながら進んでください. 途中, 頻繁に出てくる **練習問題 *.*** は, そこまでに学んだ事項を実際に手を動かしながら理解してもらうために設けたものあり, 学習支援システム LMS (Learninig Manegement System) へ入力して頂く課題 (「※ LMS 入力課題」と断り書きが書いてあります) とは別に作ってあります. また, 各配布資料の最後にあげられている **今週の LMS 入力課題** は, 各回の知識を使ってある程度まとまった作業を行ってもらうための問題であり, 翌週に詳しい解説を行う可能性の高い問題です. 学習ペースと理解度は, 各回, この問題が解けるかどうかを確認することでつかめるようになっています.

6.1 ファイルとその操作

まずはプログラミングにおいて恒常的に必要となるファイルの作成, 表示, コピー (複製), そしてファイル名の変更等について順を追って見ていきましょう.

- **ファイルの作成**

  前回の講義で少し見たように, (テキスト) ファイルを作成するにはテキストエディタ Xemacs を起動させ, データを書き込んだ後に Ctrl-x Ctrl-s を行い, エコー領域に自分の好きな名前を書き込めば良かったわけです. しかし, こうしなくとも Xemacs を起動する段階で自分が付けたい名前を指定して

  ```
  xemacs test.c &
  ```

  とすればよく[1] , こうして開かれた Xemacs の画面の中に文字を書き込み, その後に Ctrl-x Ctrl-s とすれば, 自動的に test.c という名前でキーボードから入力した内容がファイル test.c に保存されることになります. ここで, 入力行の最後につけた & マークの意味は後に詳しく説明しますので, ここでは特に気にしないでください.

  これを確認するために次の練習問題を実際にやってみましょう.

  練習問題 6.1

  test.c という名前で Xemacs を起動し, 下記の文章を打ち込んだ後に保存せよ.

  ```
  #include<stdio.h>
  main()
  {
      printf("Today is 13th April 2007 and Friday! ");
  }
  ```

  ※ このプログラムの意味については後ほど詳しく見ていくことになります.

- **ファイルの表示**

  ファイルの内容を確認するためわざわざテキストエディタ Xemacs を起動させなくても, **less** というコマンドによって, 現在使用しているターミナルにファイルの内容を表示させることもできます. 具体的には

  ```
  less [ファイル名]
  ```

  とすれば指定した [ファイル名] の内容がターミナル上に表示されることになります. less を終了させるには q とすれば OK です. 既に述べたように, この less はファイルの内容を表示させるためだけのコマンドであり, 編集機能はありません. 従って, ファイルの内容を参照するだけでなく, 書き変えたい, 一部を削除したい場合には今までどおりに Xemacs を使ってください.

  それではどのような場合に編集機能を持たない less を使うべきなのかが気になりますが, 例えば, システム管理などで, ファイルの一部を (何らかのアクシデントで) うかつに書き換えてしまうと, その後のシステム動作に悪影響を与えかねないような種類のファイル内容を一時的に参照したい場合, 編集機能を持つエディタよりも less が安全であり, 好ましいことになります.

  練習問題 6.2

  less コマンドを用いて test.c の内容を確認せよ.

[1] いちいち書きませんが, 各コマンドを打ち込んだ後には [Enter] を押すこと

- **ファイルのコピー**
  ときとして同一の内容のファイルをもう一つ複製 (コピー) したい場合があります．このような場合には **cp** コマンド (**cop**y の略) が有用です．例えば, test.c と全く同じ内容のファイル test2.c を作りたいのであれば

```
cp test.c test2.c
```

  とすれば OK です．つまり, より一般的に書けば

```
cp [コピー元のファイル名] [コピー先のファイル名]
```

  となります．実際に上記のコマンドでファイルが複製されることを次の練習問題で確かめてください．

練習問題 6.3
cp コマンドを用いて, test.c の複製を作り, その内容を less コマンドで確認せよ．

- **ファイル名の変更**
  ファイル名を変えたい場合には mv コマンド (**mov**e の略) が便利です．例えば, test2.c というファイル名を test3.c に変更したいのであれば

```
mv test2.c test3.c
```

  とすれば OK です．一般的に書けば

```
mv [変更前のファイル名] [変更後のファイル名]
```

  となります．これを次の練習問題で確かめてください．

練習問題 6.4
mv コマンドにより, test2.c の名前を test3.c に変更せよ．

- **ファイルの削除**
  不要なファイルは, こまめに削除しておきたいものです．そのような場合には **rm** コマンド (**rm**ove の略) が有効です．例えば, test3.c を削除したいような場合には

```
rm test3.c
```

  とすれば OK です．

  ところで, 当たり前ですが, 一端削除してしまったファイルは復活しません．しかし, 削除してしまったファイルが, それまでに何度か編集を繰り返し, その都度「上書き保存」を実行していたファイルであれば, backup (old) file (ヴァージョンの一つ古いファイル) として, 例えば, オリジナルファイルが text.txt というファイル名であれば, text.txt~ というファイル名で, かろうじて (保存する前の古いヴァージョンですが) ファイルが生き残っている場合があります．その際は, 前に学んだ mv コマンドを用いて, その名前 text.txt~ を text.txt へと変えてください．こうすれば最新のヴァージョンは復

活できないにしても, 一つ古いヴァージョンを元に, 引き続き text.txt の編集作業を続けることができます[2] .

練習問題 6.5

rm コマンドで test3.c を削除せよ.

6.2 ディレクトリ構造

ディレクトリとは一言で言うと「データの記憶 (ファイルの保管), 参照や作業を行う場所」のことです. これは Linux のシステムでは次のような「木」の構造をとります. この図の最上部にある root は**ルート**

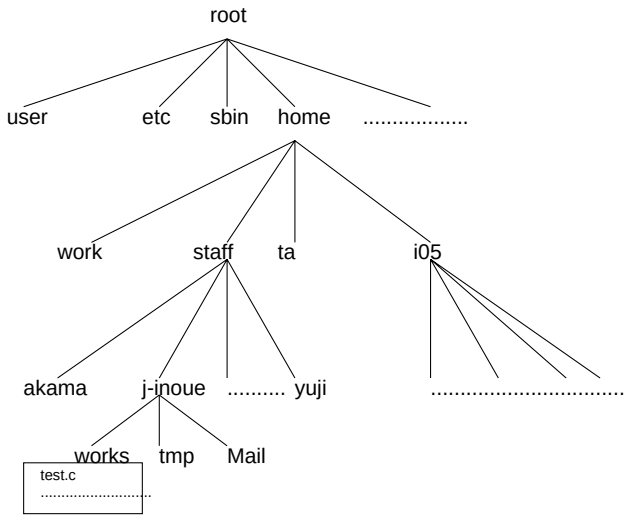

図 10: Linux のディレクトリ構造.

ディレクトリと呼ばれます. 前回の講義では Xemacs に自分の名前を書き込んでもらい, それを name.txt という名前のファイルに保存してもらいました. このファイルはログインした時に自分がいるディレクトリにあるはずであり, **ホームディレクトリ**と呼ばれます. つまり, 各ユーザにはそれぞれ一つのホームディレクトリが割り当てられているわけです. また, 木構造の構成要素的な観点から言えば, (自分のホームディレクトリに何も新しいディレクトリを作っていない状況下では) 各個人のホームディレクトリはこの木構造の末端に位置します図 10 参照.

6.2.1 絶対パスと相対パス

さて, Linux システムのディレクトリ構造は上述のような木構造をとるわけですから, その性質上, **ルートディレクトリから任意のディレクトリへの経路は一意に定まります**. 従って, 一貫したルールを設ければ, 全てのディレクトリは一意に指定されることになるわけです. その規則とは, 例えば, 上図で私 (井上) のホームディレクトリ j-inoue を指定する場合には

```
/home/staff/j-inoue/
```

[2] この意味でも Xemacs でプログラムの編集作業をする際には, こまめにファイルを「上書き保存」すること. Xemacs での「上書き保存」のキー操作は Ctrl-x Ctrl-s です.

とすれば良いし, 私のホームディレクトリにある works というディレクトリを指定したいならば

```
/home/staff/j-inoue/works/
```

とすれば良いわけです. ここで, いずれの場合にも先頭の「/」はルートディレクトリを表す記号であると思ってください. このように, Linux の計算機システムでは任意のディレクトリは「/」から出発して「ディレクトリ名/」を順々に並べていけばよいことになります.

また, 任意のファイルも指定可能です. 例えば, 私のホームディレクトリの下にある works ディレクトリの内の test.c という名前のファイルを指定したいのであれば

```
/home/staff/j-inoue/works/test.c
```

とすればよいことになります. このような形でのディレクトリ/ファイル指定を**フルパス指定 (絶対パス指定)** と呼びます.

ところで, ここですぐに気になるのは, このような方法で全てのディレクトリやファイルを指定する場合, 末端のファイルが木構造の深部へ行けばいくほど長くなり, 非常に使いにくくなる点です. このような不便さを回避するために, 現在自分がいるディレクトリ (**カレントディレクトリ**と呼ぶ) を基準に, 目標とするディレクトリやファイルを指定する**相対パス指定**も利用できます. 例えば, 自分が現在, ディレクトリ j-inoue にいるとすれば, このカレントディレクトリから, ディレクトリ works 内のファイル test.c を指定する場合には

```
./works/test.c
```

とすれば良いわけです. ここで, 「.」はカレントディレクトリを表します. なお, これは省略が可能であり

```
works/test.c
```

としても OK です. また, ホームディレクトリの所有者がホームディレクトリ (j-inoue) からこのファイルを指定する場合には

```
~/works/test.c
```

とします. この「~/」は自分のホームディレクトリを表します.

以上のことを念頭に, 以下でディレクトリとファイルの操作法を具体的に見ていくことにしましょう.

6.2.2　ディレクトリとファイルの操作法

ここからはディレクトリ, ファイルを操作する際のコマンドのいくつかを見ていきます. これらはいずれも比較的使用頻度の高いものなので, この場で覚えて, 今すぐにでも使えるようにしておきましょう.

- **カレントディレクトリに存在するファイルやディレクトリの参照法**
  既に前回使っていますが

  ```
  ls
  ```

  とすれば, カレントディレクトリに存在するファイルの一覧を参照することができます. (**l**i**s**t の略).

- **カレントディレクトリのフルパス指定の参照**
  現在自分のいるディレクトリのフルパス指定はコマンド pwd (**p**resent **w**orking **d**irectory の略) で参

照できます.

```
pwd
```

実際に次の練習問題で動作を確かめてください.

練習問題 6.6
カレントディレクトリで pwd コマンドを実行し, 画面への出力結果を確認せよ.

- **ディレクトリの作成**
  name.txt の他にも多数のファイルを作成した場合 (例えば, test.c, tmp.dat etc.), それらのファイルの中で互いに関連するものどうしをまとめて保存した方がよい場合があります. こうしたときにはホームディレクトリ, あるいは, カレントディレクトリに新たなディレクトリを作成し, そこに互いに関連するファイルを保管することになります. この方法を以下で説明することにしましょう.

  例えば, カレントディレクトリに tmp という名前のディレクトリを作成したいのであれば

```
mkdir tmp
```

  とすれば OK です. (ここで mkdir は **ma**k**e** **dir**ectory の略). このとき, ls コマンドを用いて確認すれば, カレントディレクトリに新しいディレクトリ tmp が存在するのがわかります.

- **ディレクトリの削除**
  ファイルの場合と同様に, 不要になったディレクトリは削除し, 親ディレクトリを整理したくなります. そのときには rmdir (**re**m**o**ve **dir**ectory の略) コマンドが使えます. 実際に上で作った tmp ディレクトリを削除したいのであれば

```
rmdir tmp
```

  とすれば OK です. ここで注意しなければならないことは, **このコマンドが機能するためには, 削除対象のディレクトリ内にファイルが存在せず, 空の状態であることが必要になる**という点です.

- **ディレクトリへのファイルの移動**
  これで新規ディレクトリが作成できましたから, このファイルを格納する「箱」の中にファイルを入れることを考えます. つまり, 新規のディレクトリの中に既存のファイルを入れることを考えましょう. このときには, 「ファイル名の更新」のところで既にみた, mv コマンドが役に立ちます. カレントディレクトリにある test.c をカレントディレクトリの下に作成したディレクトリ tmp に移動したいのであれば

```
mv test.c tmp/
```

  とすればよいことになります. この最後の「/」は tmp がディレクトリであることを明示するためであり, 付けなくても機能します. 次の練習問題をやってみてください.

練習問題 6.7

自分のホームディレクトリに tmp という名前のディレクトリを新たに作成し, そこに test.c, name.txt の 2 つのファイルを移動せよ.

- **ディレクトリ間の移動**

複数のディレクトリ間を行ったり来たりするには cd (**c**hange **d**irctory の略) を用います. この場合, 行き先のディレクトリの指定に関しては前出の「絶対パス指定」「相対パス指定」の双方が可能です. 例えば, 前出の図 10 にある私 (井上) のホームディレクトリ j-inoue に行きたい場合には, 絶対パス指定で

```
cd /home/staff/j-inoue/
```

とすれば良いし, 現在自分のホームディレクトリにいる私がディレクトリ staff の直下にある赤間先生のディレクトリ akama に行きたい場合には相対パス指定で

```
cd ../akama
```

とすればよいことになります. ここで, 「..」は「/ディレクトリ名」の一つ手前のディレクトリを意味します. これを「/ディレクトリ名」の**親ディレクトリ**と呼びます. 具体例を用いてこれを確認するならば, 例えば, ディレクトリ j-inoue で

```
cd works
```

とした後に

```
cd ../
```

と行うと, この時点でカレントディレクトリは j-inoue ということになります. つまり, 「j-inoue は works にとっての親ディレクトリ」ということになります.

ところで上出の赤間先生のホームディレクトリに絶対パス指定で行きたい場合にはもちろん

```
cd /home/staff/akama/
```

とすればよいことは, もうおわかりでしょう. 次の練習問題をやってみてください.

練習問題 6.8

私のホームディレクトリ (j-inoue) の下の tmp というディレクトリに check.txt というファイルを置いてある. これを各自が自分のホームディレクトリに作った新規ディレクトリ tmp にコピーし, このファイルを Xemacs で開くことにより, このファイルに書かれた内容を確認せよ.

6.3 ファイルへのアクセス権限とその変更

上の **練習問題 6.8** をやってみた段階で次のような疑問を持たれた方もいるでしょう. すなわち, 「他人のディレクトリにある任意のファイルがコピーできて, それが読めるということになると, プライバシーの保護, 個人情報の取り扱いという観点から言って, Linux というシステムは好ましくないのではないか ?」と. これに関しては, 各ファイルの所有権, アクセス権限を変更することで, 他人には見せたくないファイルのアクセス・モードを読み込み (書き込み) 禁止にすることができます.

これをみるために, まずは ls コマンドを

```
ls -l
```

のようにオプション「−l」付きで実行してみましょう. すると画面には次のような表示が現れるはずです.

```
-rw-r--r-- 1 j-inoue staff 17 4 月 13 日 11:13 check.txt
```

この表示で先頭の文字「−」を除く (この「−」に d が立っていれば, それはディレクトリであることを意味します) はじめの 9 文字は 3 つずつ 1 組で意味を持ち, 左から「所有者」「グループ所属者」「第 3 者」を意味し, そこに表記されている「r」は「読み込み可」を「w」は「書き込み可」を, また上の例では該当しませんが, 「x」は「実行可」を意味します. 従って, 上の例で言えば, check.txt の所有者である j-inoue は書き込みも読み込みも可能であり 1, グループ内の人たち (staff ディレクトリにホームディレクトリのある人たち) は読み込みにみ可能であり, 第 3 者の人たちに対しても読み込みのみが可能であることを意味しています.

従って, **練習問題 6.8** で皆さんがこのファイル check.txt を読むことができたのはこのファイルが上記のようなアクセス権限であったためであったわけです.

さて, ここで私がこのファイル check.txt を他人に見せたくないと思った場合, どのようにファイルのアクセス権限を変えたらよいのでしょうか ? この場合にはコマンド chmod を用います. (**ch**ange **mod**e の略). 具体的には

```
chmod o-r check.txt
```

とすれば OK です. このモード指定の先頭の「o」は他人 (others) を意味し, 真ん中の「−」は禁止の意味です. 最後の「r」は「読み込み」を意味しますので, これらを 3 つ組み合わせた「o−r」は「第 3 者の読み込みを禁止する」ということになります. また, このファイルを再び第 3 者が読み込み可能にするのであれば

```
chmod o+r check.txt
```

とモード指定の真ん中の「−」を「+」に変更すれば OK です. 「o」の部分は「a」(all, 全て), 「u」(usr, 所有者), 「g」(group members) に変更が可能です. また, 「r」の部分も「w」(書き込み), 「x」(実行) に変更可能です. 実際にこれらを次の練習問題で確かめてください.

練習問題 6.9

自分の tmp ディレクトリにある test.c のアクセス権限モードを確認し, コマンド

```
chmod g+w,o-r test.c
```

を実行した後のアクセス権限モードと比較せよ.

6.4　オンライン・マニュアルの活用

以上確認した各種コマンドの詳細はコマンド man (**man**ual の略) で確認できます. 例えば, less の使い方に関する詳細が知りたければ

```
man less
```

とすれば, less の活用法に関する詳細が閲覧できます. この講義ではフォローしきれない, しかし, 使い勝手が良いコマンドもいくつかありますので, この man でこまめに確認することが Linux を上達するための近道かもしれません. なお, man を停止するには q を押してください[3] .

7　プログラムの開発手順： まずは簡単に説明

ここからは簡単なプログラムを Xemacs により書き, それをコンパイル・リンクすることにより実行形式ファイルに変換し, それを計算機上で実行するという一連の流れを確認していくことにします. なお, 今回学ぶ事項は教科書の 8 ページまでに書かれた内容です.

7.1　計算機に仕事をさせるための流れ

何かの仕事を計算機にさせる場合, 下に示した一連の作業を行うことになります.

プログラムファイル (*.c) ⇒ **コンパイル, リンク** ⇒ **実行形式 (a.out)**

以下で各々のプロセスを簡単に見ていくことにしましょう.

- **Step1： Xemacs による C プログラムの作成**
  このステップではデータ構造を定義し, そのデータに対してどのような操作をどのような順序で行うかを, 一貫した文法の下に書き下すことを行います. 例えば, 先に書いてもらった test.c を思い出していただくと

```
#include<stdio.h>
main()
{
   printf("Today is 13th April 2007 and Friday!");
}
```

  でしたが, これは C 言語による最も簡単な典型です. このプログラムの意味は「Today is 13th April 2007 and Friday!を画面に表示する」という意味です.

  この例に見るように, C 言語によるプログラムは一般的に

[3] man は調べたいコマンドと共に man *** と呼ばれるごとに, システム上に置いてある該当するコマンド *** の説明が書かれたテキストファイルを読み込みに行き, そのファイルを less コマンドで画面表示する働きをします. 従って, 画面のスクロール法も less と同じであり, 終了方法も less の停止方法と同じ q となります.

```
main()
 {
     文 1;
     文 2;
         ..............
         ............

.....................
  }
```

のような形をしています．ここで, 先頭の main() というのは main という名前の**関数**の始まりを意味します．この関数の働きは「{」から「}」までの文によって定義され, **関数**はデータの入出力, 計算, 計算順序の制御などといった基本的な動作を表す文を並べることにより定義されます．また, この main 関数はプログラムの中にただ一つだけ存在しなければなりません．

さて, もう一度 test.c を見てみると, このプログラムの場合の main 関数の定義範囲は 3 行目の「{」から 5 行目の「}」までです．次に「{」から「}」に挟まれた「文」に関してですが, この文 (statement) は数値や文字に対して演算を行ったり, 関数の実行を指示したり, 条件によって次の処理を選択したり, あるいは一定の条件を満たしている間, ある処理を繰り返すといった動作を表します．文は 1 行である必要はなく, セミコロン「;」で終わる文字列です．以上のことをふまえて再度 test.c を見てみると, 先頭の

```
#include<stdio.h>
```

ですが, これはプログラムで共通に用いる各種関数や変数の定義を一つのファイルにまとめて stdio.h という名前をつけておき, それをこのプログラム (test.c) の中に取り込んで用いることを表しています．このような形式でプログラムの中に取り込む「.h」という**拡張子** (ファイルの種類・属性を識別するためのタグ) のついたファイルを**ヘッダファイル**と呼びます．この一行は必要な場合に書けば良いのですが, 必要の無い場合に書いたとしても何も問題は無いので, 通常はこの 1 行を必ず書き込むようにしましょう．

プログラム test.c で次に問題になるのが

```
printf("Today is 13th April 2007 and Friday!");
```

ですが, これは関数 printf() であり, 一般に

```
printf("画面に表示させたい文字列");
```

のような形式をとり, 「"...."」で囲まれた部分を画面上に表示させる関数です．従って, main 関数の中でこの printf 関数を呼ぶことにより, プログラム全体としては「Today is 13th April 2007 and Friday!を画面に表示する」という働きをすることになるのです．

さて, これを実行するためには, このプログラムをコンパイル (およびリンク) し, **実行可能ファイル (executable file)** に変換しなければなりません．

- **Step 2：コンパイル・リンクの方法**
  test.c をコンパイル・リンクして実行可能形式に変換するには

```
gcc test.c -lm
```

  とします．ここで, 最後のオプション「−lm」はサインやコサインなどの数学関数を用いるときには付けなければなりません．数学関数を用いない場合にこのオプションをつけても問題は無いので, 通常はこれをつけるようにしましょう．このコンパイルが成功するとカレントディレクトリに

```
a.out
```

  という名前のファイルができているはずです．次の練習問題で確かめましょう．

練習問題 7.1
test.c に関して各自がコンパイルを実行してみよ．次いで, ls コマンドを用いてカレントディレクトリに a.out が存在するか確認せよ．また, test.c の printf 関数の最後のセミコロン (;) を削除したプログラムを test2.c として保存し, この test.c を同様にコンパイルしたとき, どのようなエラー・メッセージが画面に表示されるかを確認せよ．

- **Step 3：実行**
  コンパイルで実行形式ファイル a.out が作成された場合, この実行形式のファイルを実行するには

```
./a.out
```

  とすれば OK です．より一般的には

```
./[実行可能形式のファイル名]
```

  となります．早速, 次の練習問題をやってみてください．

練習問題 7.2
test.c をコンパイルすることにより作成された a.out を実行し, 結果を確認せよ．

練習問題 7.3
次のプログラムを test2.c という名前で作成し, コンパイルしてできる実行可能形式 a.out を実行してみることでこのプログラムの意味を考えよ．

```
#include<stdio.h>
main()
{
  double a,b,x,y;
  a=5;
  b=3;
  x = a+b;
  y = a*b;
```

```
    printf("sum=%lf prod=%lf\n",x,y);
}
```

今週の LMS 入力課題

1[s] 間に自由落下する距離 [m] を計算し, その答えを画面上に表示するプログラムを作成せよ. ただし, 重力の加速度を 9.8[m/s^2] とすること.