第 1 回 UNIX ゼミ

ゼミ担当者　：　折戸　俊彦，荒久田　博士，鈴木　和徳
指導院生　　：　片浦　哲平，谷口　義樹
開催日　　：　2003 年 4 月 15 日

ゼミ内容: 本ゼミでは，研究活動での UNIX の使用において，最低限必要となる知識および操作のスキル取得を目的とする．本研究室では最適化の研究に並列計算機を用いることが多いが，その際に，並列計算機の利用および管理のための LINUX の知識が必要不可欠となる．そこで，本ゼミではそれらの利用が可能となるように，LINUX 上でディレクトリ操作，ファイル操作，パーミッション設定，エディタの利用方法などについて学ぶ．

1　UNIX(LINUX) とは

1.1　UNIX とは

UNIX とは，1969 年に米国 AT & T Bell 研究所で，Dennis Ritchie と Ken Thompson によって開発された OS である．後に，Ritchie が開発した C 言語によって 1972〜1974 年ごろに書き直された．UNIX はソースコードが比較的コンパクトであったのと，ライセンスが安価に配布されたために大学や研究機関などを中心に普及していった．ちなみに現在，商標としての「UNIX」は，The Open Group[1]が所有しており，一定の仕様を満たした OS のみが「UNIX」の名称を使うことができる．

1.2　Linux とは

Linux とは，1991 年にヘルシンキ大学の Linus B. Torvalds 氏によって開発された UNIX クローン（互換）の OS である．Linux は既存のオペレーティングシステムのコードを流用せず，なにもないところから書き起こされたものであり，GPL というライセンス体系に基づき，自由に改変，再配布を行うことができるようになっている．知的システムデザイン研究室で主に利用されているのは，Debian GNU/Linux というディストリビューションである．Linux と呼んでいるのはカーネル自体であり，そのカーネルは基本的にディストリビューションに依存しているわけではない．各ディストリビューションの違いは，日本語化されている度合いやインストーラの完成度，各ソフトの初期設定状態などである．

1.3　Windows や MacOS との違い

UNIX 系 OS は，Windows や MacOS と異なる．これらの OS は根本的に，シングルユーザの OS であり，それに対して UNIX は，マルチユーザの OS である．マルチユーザとは 1 台のコンピュータを複数人で同時に使用できるということである．それに対して 1 台のコンピュータを，同時に 1 人が利用することをシングルユーザという．また，UNIX はマルチタスクの OS でもあり，UNIX においては同一システム上で同時に複数の人が作業可能な設計になっている．マルチユーザ，マルチタスクのシステムにはセキュリティの概念が必要になってくるが，セキュリティも過度に複雑にならないようシンプルに設計されており，そのシンプルさゆえに十分なセキュリティ管理のしやすさを UNIX では実現している．UNIX では GUI（Graphical User Interface）には X を利用しているが，X は UNIX にとって必要不可欠なものではないので，取り外すことができる．その場合には，CUI（Character User Interface）によるオペレーションを行う．ちなみに，MacOS X は内部構造に UNIX 系のシステムが組み込まれており，UNIX のディレクトリ構造やマルチユーザを実現している．

1.4　なぜ Linux を用いるか

Linux は次のような特徴を持っている．

- 比較的性能の低いコンピュータでも軽快に動作する
- ネットワーク機能やセキュリティに優れている
- オープンソースである
- 安定している

これらの理由により，学術機関を中心に普及してきたが，近年は企業のインターネットサーバなどの用途にも広く採用されている．

Linux に対して，Linux がテキストモードの CUI でしか扱えないというように，誤解している人もいるが，Linux では GUI 操作が可能である．日常的に Linux を使っている人で，コマンドラインしか使っていないというような人は少なく，一般に X というシステムで画像を表示し，GNOME[2]や KDE の統合環境で作業を行っている．使用するアプリケーションや，アプリケーションを動作させる環境によって，どちらのデスクトップ環境が優れているかは一概には言えない．さらに，Linux で

[1] http://www.opengroup.org

[2] GNU Object Model Environment

は日本語が使えないという誤解もあるが最近の Linux 上での日本語の扱い方に関しては，飛躍的に向上している．

1.5 オープンソースとは

Linux のアプリケーションは，一般的に GPL というライセンスに基づいて無料で配布されている．よってユーザは Web から公開されたアプリケーションを持ってきて，自分の扱うマシンに組み込むことができる．このことは，Linux が発展していくのに大きな役割を果たしている．Windows や MacOS に関する無料のソフトウェアも存在するが，それらはフリーソフトウェアと呼ばれるものであり，Linux が GPL により配布されるソフトウェアは，オープンソースソフトウェアである．オープンソースソフトウェアは，ソフトウェアと共にソフトウェアの設計図となるソースコードが公開されおり，類似品を作成することや，そのソフトウェアで使われている技術を転用することが可能である．ソースが公開されているので，ユーザはソースを変更して自分の環境にあわせることが容易に可能となる．GPL ライセンスの主な特徴として，以下のようなものが挙げられる [1)]．

- ソフトウェアは必ずソースプログラムとともに配布，複製される．もしソースプログラムをつけずに配布する場合は，ソースプログラムを確実に入手できる手段を提供することが義務付けられる．
- ソフトウェアを，使用，複製，変更，配布したり，新しいフリーソフトウェアの一部として利用できること．
- 変更，改良されたソフトウェアは GPL に従って配布されること．
- プログラムの全部あるいは一部を用いて作られたソフトウェアは GPL に従って配布されること．
- 基本的に無保証であり，そのソフトウェアが原因でトラブルが生じても作者に責任はないこと．

1.6 文字コードについて

日常的に Linux を利用するときに引っかかりやすい問題として，Linux と Windows と MacOS では，文字コードが違うことが挙げられる．漢字コードは，Linux では EUC-JP によるエンコードが一般的であり，Windows や MacOS では，SJIS によるエンコードが一般的に使われている．日本語を扱う場合に，何も考えずにそれぞれの形式を扱ってくれる「秀丸」などの高機能エディタを使う場合は良いが，そうでない場合は注意が必要である．それぞれの OS は Table 1 のようになっている．

Linux で作ったファイルを Windows で開く場合，文字コードを EUC からシフト JIS に，改行コードを LF から CR+LF に変えるとよい．

Table 1　各 OS の文字コードと改行コード

OS	文字コード	改行コード
Linux	EUC	LF
Windows	シフト JIS	CR+LF
MacOS	シフト JIS	CR

1.7 root ユーザ

システム管理者はそのシステム上で強力な権限を持っている．システム管理者がその操作を誤って行った場合，システム全体に与える被害は非常に大きな物となってしまう．そのためシステム管理者といえども，普段から絶大な権限をもつユーザーとして UNIX を利用するのは大変危険である．そこで，システム管理者は普段は一般のユーザー ID で UNIX システムを利用し，必要のあるときだけ，システム管理用のユーザー ID を利用して作業を行う．この作業時に利用するシステム管理用のユーザを**「root」**と呼ぶ．システム管理者はこの root というユーザー ID を利用しているときだけ，システム管理者としての権限を利用できるようになる．この root というユーザー ID を利用するためのパスワードは，そのシステムの最高機密といっても過言ではない．

2 UNIX という考え方

OS を使いこなすためには，その背後にある哲学を理解することが必要である．この章では UNIX という OS の考え方を紹介する．

2.1 9つの定理

1. スモール・イズ・ビューティフル
   巨大で複雑なプログラムの開発者は，将来が予測可能で，そして現在とそう大きくは変わらないという勝手な思い込みを前提としている．一方，小さなプログラムの開発者が考えるのは，明日作られるものは今日作っているものとは違うということでしかない．

2. 一つのプログラムには一つのことをうまくやらせる
   最初の仕様から変更のないプロジェクトなど稀である．

3. できるだけ早く試作する
   試作によって何がうまくいくかが分かり，さらに重要なことには何がうまくいかないかが分かる．

4. 効率より移植性を優先する
   プログラムを速くすることに時間をかけないこと．最も効率のよい方法は，ほとんどの場合移植性に欠ける．

5. 数値データはASCIIフラットファイル[3]に保存する
   ASCIIテキストは共通の交換形式．ASCIIテキストは簡単に読めて編集できる．

6. ソフトウェアを挺子として使う
   よいプログラマはよいコードを書く．偉大なプログラマはよいコードを借りてくる．

7. シェルスクリプトによって梃子の効果と移植性を高める
   シェルスクリプトはCよりも移植性が高い．

8. 過度の対話的インターフェイスを避ける
   拘束的プログラムはユーザを人間と想定している．人間の限界によって動作を制限されるようなシステムは，潜在能力をフルに発揮できない．

9. すべてのプログラムをフィルタとして設計する
   データの合成ではなく，与えられたデータを選択的に通過させることに集中する．

2.2 総括

小さなプログラムというものは，人間にとって大きな「何か」よりも分かりやすい．つまり，理解しやすければ，当然保守も容易となるので，最終的には対費用効果が大きくなる．また，読み込み，実行，開放のいずれもすばやく行うことができ，効率が高まる．小さなプログラムは，ほかのツールと簡単に結合できる．単独ではたいしたことができなくても，ほかの小さなプログラムと組み合わせることによって，新しいアプリケーションを間単に作成することもできる．

小さなプログラムを作成するためには，目的をしぼらなければならない．すなわち，一つのことをうまくこなすことに専念しなければならない．大きくて複雑な単体型プログラムでは，ほかのアプリケーションとの共同作業に莫大な労力をささげなければならない．

世界のソフトウェアは絶えず進化し続けており，将来のニーズにまで対応するソフトウェアを作ることなど到底不可能である．できることといえば，今日のニーズに対応するソフトウェアを作ることである．開発者は設計仕様書の作成に何ヶ月も費やす無駄をやめ，早期に試作を作るべきである．試作が他人の手に早く届くことによって，システムはより良く変貌していく．変更が必要な場合では，すべてが完成してしまった最終段階において変更するよりも，万事が流動的な初期段階において変更するほうがいいのは当たり前である．試作を作ってみれば，何がうまくいくのか，そしてより重要なことに，何がうまくいかないのかが早期に発見することができる．

[3]すべての数値データをASCII文字として格納することを意味する

ソフトウェアというものは，実は作るものではなく，成長していくものである．新しいハードウェア，新しいアーキテクチャは頻繁に現れる．ソフトウェアはそれらの新しい環境に対応していかなければならない．つまり，移植性の高いソフトウェアでなければならない．移植できるものは生き残り，それ以外のものは，やがて時代に取り残されることとなる．

すべてのソフトウェアというものは，命令とデータとから構成されている．ソフトウェアの移植性を考える場合には，データの移植性もあわせて考えなければならない．データはASCIIフラットファイルとして保存する．ASCIIテキストは，最良の形式でないにしろ，間違いなく最も一般的な形式である．データをテキスト形式で保存することによって他システムへ移植をきわめて簡単なものとしている．

「良いプログラマはよいコードを書く．偉大なプログラマは良いコードを借りてくる」，すなわち，新しいアプリケーションを開発するには，既存のソフトウェアのコードを再利用するべきである．新しいアプリケーションを一から開発するのはいいことである．しかし，誰かがすでにやっていることを改めてやり直すのは，時間の無駄である．今日，世界に存在するソフトウェアは偉大な共有資産である．

ソフトウェアの梃子の効果をいっそう大きくするために，できるだけシェルスクリプトを使う．シェルスクリプトは他人の努力の結果を自分の目標の達成に向けて組み込むことができる．シェルスクリプトで使うコードの大部分は自分で書くのではない．すでに他人が書いてくれているものを利用するだけでよいのである．シェルスクリプトを使えば，世界中の人々の仕事を合成することができ，小さな努力で大きな成果をあげられる．

小さなプログラムのコレクションが手元にあれば，シェルスクリプトを簡単に作れる．しかし，ユーザにいちいち応答を要求するようなプログラムは，あまり役に立たない．拘束的ユーザインターフェイスは避けるべきである．拘束的プログラムは，どこかで人間と対話することを想定している．そのため，これをシェルスクリプトに組み込むことは非常に難しい．

すべてのプログラムは，簡単なものでも複雑なものでも，何らかの形式のデータを入力として受け入れ，何らかの形式のデータを出力として生成する．すなわち，すべてのプログラムはフィルタである．プログラムはデータを作らず，データを作るのは人間である．プログラムはデータを一つの形式から別の形式へと変換するだけである．

3 ファイル・ディレクトリについて

Windowsを使用していても，ファイルを格納する場合に一つの場所だけにすべてのファイルを格納しようと

すると，それぞれのファイルを識別するためにファイル名をすべて別のものにしないといけなくなってしまい，ファイルの管理が非常に大変になる．さらに，UNIX のように複数の人が使うシステムではその管理がより面倒なことになってしまう．

そこで，ディレクトリという概念が出てくる．ディレクトリとは,「京都府の京田辺市の興戸に住んでいる山田さん」といったように，あるものを指定する際に大分類から開始して階層に分けて指定するものである．こうすることで，物事の管理がしやすくなる．ディレクトリが別なら同じファイル名のファイルを作成することも可能となる．

Fig. 1 の”/”で表されているディレクトリをルートディレクトリといい，全てのディレクトリの起点となる．

現在自分がいるディレクトリをカレントディレクトリといい，あるディレクトリの一つ上のディレクトリを親ディレクトリ，一つ下を子ディレクトリという．ユーザがログインしたときに最初にいるディレクトリをそのユーザのホームディレクトリという．これは通常は”/home”ディレクトリ以下にユーザ名と同じ名前で，”/home/arakuta”のように作成されている．一般ユーザは通常このディレクトリ以下に対してのみ，ファイルを作成する制限が与えられている．

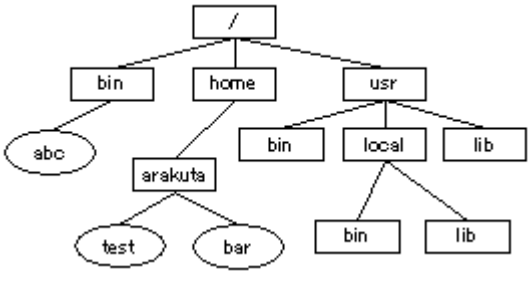

Fig. 1　ディレクトリ構造の例

3.1　絶対パスと相対パス

ファイルやディレクトリを指定するには，ファイルシステム上での絶対的な位置を指定する方法と自分が今いる位置からの相対位置を指定する方法がある．

絶対的な位置を指定する場合は，”/”を先頭において，ルートディレクトリからの位置を”/home/xyz/sample/test”のように指定する．

相対的な位置を指定するには，先頭に”/”をつけずに，現在のディレクトリを起点として，”sample/test”のようにする．こうすることで，たとえばいま，”/home/xyz”にいたとすると，全体として，”/home/xyz/sample/test”というファイルが指定されることになる．

相対的な位置指定の場合によく用いられるものとして，”.”でカレントディレクトリを指し，”..”で親ディレクトリを指すというものがある．

3.2　所有権とパーミッション

UNIX では，全てのファイルについて，そのファイルを所有しているユーザとグループを設定できる．所有者，グループ，その他の人に読み込み権，書き込み権，実行権をそれぞれ設定することができる．また，通常のファイルとディレクトリでは多少意味が異なる．通常のファイルではほぼそのままの意味であるが，ディレクトリの場合は，読み込み権とはそのディレクトリの一覧を表示する権利であり，書き込み権とはそのディスクの中にファイルを作成する権利であり，実行権とは，そのディレクトリをカレントディレクトリとする権利である．

4　プロセスとジョブ

UNIX ではあるコマンドを指定して実行すると，それは「ジョブ」と呼ばれる実行単位になる．たとえば，man cat というコマンドを実行すると，それは，man cat という仕事の単位になる．現在のジョブは jobs コマンドで調べることができる．ジョブというものは，端末ごとに見た仕事の単位であり，実際には UNIX 本体は man cat をもっと小さな仕事の単位で処理している．man コマンドを実行すると，pager コマンドが起動されるものがわかる．ユーザ man cat を入力すると，UNIX 内部では，

```
20820 pts/5     00:00:00 man
20823 pts/5     00:00:00 sh
20825 pts/5     00:00:00 pager
```

というように 3 つのプロセスが起動している．この小さな実行単位を「プロセス」と呼ぶ．

参考文献

1) 情報通信辞典
http://e-words.jp/