

システムプログラミング

1回目：イントロダクション

芝浦工業大学 情報工学科
菅谷みどり

システムプログラミング

講義目的

- OS の授業で学んだことを実践的にプログラミングを通じて学ぶ
 - IoTのシステム基盤（サーバクライアントの実装技術）
 - 通信を重視したコアとなるシステムのプログラミング

システムプログラミング

3

Next Generation Computing

基盤システム研究室

<http://www.dlab.ise.shibaura-it.ac.jp/>

\db.ac.jp-dlab.aisei.dlab.www\ddt

プレゼン 10/1 (火), 10/8 (火) 304 教室

2019/9/26
2

2018
プレゼン



授業計画

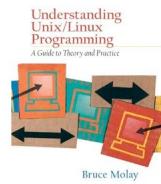
- 9/27 イントロダクション, 歴史, シェルスクリプト
- 10/4 システム管理, 開発環境プログラムのコンパイル, リンク
- 10/11 ファイル（高水準なファイル操作）入出力ハードウェアと制御, ファイルシステム, ファイル構造
- 10/18 プロセス
- 10/25 スレッド
- 11/8 プロセス2, デッドロック
- 11/15 研究紹介（講演会）
- 11/22 時刻, 割込み, シグナル
- 11/29 ネットワークプログラミングI クライアントプログラム
- 12/1 ネットワークプログラミングII サーバプログラム
- 12/13 ネットワークプログラミングIII 高度な通信
- 12/20 グループワーク
- 13.1/10 グループワーク（中間報告）
- 14.1/17 グループワーク 発表会, 提出

システムプログラミング

4

参考書

- Understanding UNIX/LINUX Programming: A Guide to Theory and Practice, Bruce Molay, Prentice Hall; 1版 (2002/11/25), 11,200
- Unix/Linux プログラミング理論と実践, Bruce Molay、長尾 高弘
- 河野 清尊, C言語によるUNIXシステムプログラミング入門



システムプログラミング

5



推薦図書

魔法のCプログラミング演習書

- 入門から実践まで -

発行年月日 : 2017/01/06

内容

- C言語の基礎から応用まで
- 語り口調でわかりやすい
- 無理なくステップアップできる



6



成績評価

- レポート, プログラミング課題 (70%)
- サーバクライアントアプリケーション製作 (30%)

システムプログラミング

7

授業の形態

- 資料の配布/プログラムの提出
 - シェアフォルダ
 - Scomb
- アナウンス
 - Scomb

システムプログラミング

8

なぜシステムプログラミングを学ぶのか

- ・ システムプログラミング言語とは、システムプログラミングでよく使用されるプログラミング言語のことである。Wikipedia引用
- ・ システムの基礎を身につけることができる
 - ・ データのIN/OUTの基礎
 - ・ 様々な上位言語の低レベルの基礎
 - ・ ハードウエアの制御方法
 - ・ ソフトウエアの基礎の考え方
 - ・ マルチタスク, マルチスレッド
 - ・ IT技術の基礎

システムプログラミング

9



講義の前提

- ・ 講義の前提
 - ・ オペレーティングシステムの講義内容の基本的な部分を理解している
 - ・ C言語によるプログラミングに慣れている
 - ・ 関連科目
 - ・ オペレーティングシステム、プログラミング入門1,2

システムプログラミング

10

本講義の概要

- ・ OSについて、Linuxを使って**体験的に学習する**
 - ・ Linux上で、シェル、C言語によるシステムプログラミングを行う
- ・ Linux
 - ・ ソースコードが公開されている
- ・ Windows, MacOS
 - ・ ソースコードが非公開
- ・ 情報工学科では、LinuxなどのUnix系OSに触れておくのは必須

システムプログラミング

11



イントロダクション

システムプログラミング

12

これからの情報システム

Internet of Things

- ・さまざまなもののがインターネットに接続する
 - ・遠隔にある計算機資源を利用しつつ、暮らしを快適/安全にする仕組

基礎的な工学分野

- ・Internet : 情報をつなぐ：ネットワーク, 通信, 分散システム
- ・Things : 物理環境の把握と利用：センサ, 組込みシステム
 - ・ロボット (制御)
- ・Of (Information Processing)
 - ・情報を処理, 分析, 判断, 学習：サーバ, データベース, 人工知能

技術・通信技術の統合, 新しいサービスの創出

- ・サービスを継続的に維持, 発展させる

2019/9/24
ネットワークを利用したロボットサービス研究専門委員会

13

IoTの技術とは



44

授業計画

1. 9/27 イントロダクション, 歴史, シェルスクリプト
2. 10/4 システム管理, 開発環境プログラムのコンパイル, リンク
3. 10/11 ファイル (高水準なファイル操作) 入出力ハードウェアと制御, ファイルシステム, ファイル構造
4. 10/18 プロセス
5. 10/25 スレッド
6. 11/8 プロセス2, デッドロック
7. 11/15 研究紹介 (講演会)
8. 11/22 時刻, 割込み, シグナル
9. 11/29 ネットワークプログラミングI クライアントプログラム
10. 12/1 ネットワークプログラミングII サーバプログラム
11. 12/13 ネットワークプログラミングIII 高度な通信
12. 12/20 グループワーク
13. 1/10 グループワーク (中間報告)
14. 1/17 グループワーク 発表会, 提出

システムプログラミング

14

連携による新しいサービスが生まれる



45

IoTの本質

IoT : さまざまなもののがインターネットに接続する

基礎的な工学分野

- Internet : 情報システム, OS, 通信, 分散System
- Things : 物理環境の把握と利用 : 組込みシステム
- of (Information Processing) : 解析, 人工知能

• + aが必要

技術をネットワークに統合し新しいサービスを創出する。最後の統合技術の鍵は、ネットワーク設計技術



デバイス, 技術中心 → 目的型への変更

ソフトウェア(人へのサービス)を中心とした統合的なアイデア, イノベーション時代へ

2019/9/24
2018年 OB/OG会資料

46

高度なIoTシステム開発において必要となる技術

サーバ

- オペレーティングシステム (マルチタスク(多-to-多), 分散処理)
- クラウドシステム (仮想マシン)

通信

- ネットワーク, ミドルウェア, ゲートウェイ
- システムプログラミング

エッジデバイス (ロボット)

- 組込みシステム
- センシング技術, 多様なセンシング技術

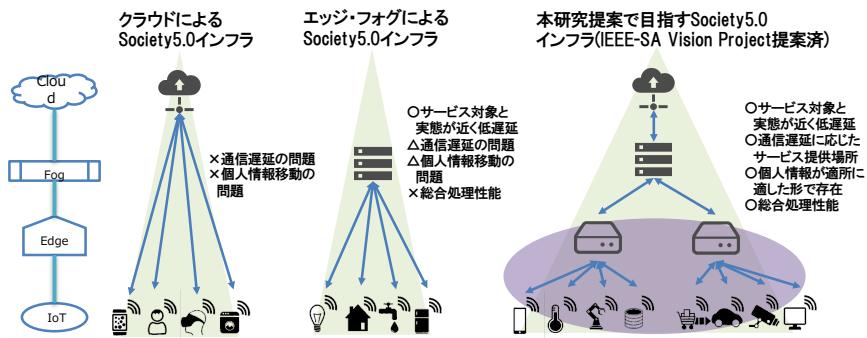
制御

- 解析技術 (データ解析技術, 人工知能/機械学習)

2019/9/26
2018年 OB/OG会資料

47

背景: エッジの低成本・高機能化が Society5.0実現の鍵



Society5.0の実現には、エッジ領域の高機能化による、個人情報のカプセル化、IoTのローカルな管理、クラウドへの情報転送を防ぐ事前処理などによる、フォグ・クラウドとのシームレスな連携や、情報管理が不可欠

Lets' Start Programming

Virtual Box の導入

- 学術情報センター：<http://www.sic.shibaura-it.ac.jp/>
 - 仮想環境(Virtualbox,VMware)におけるLinux利用方法
 - Virtual Box による Ubuntu 12.04 LTS の利用例
 - イメージダウンロード, 設定
 - ユーザの追加(adduser), emacs のインストール
- ログイン
 - ユーザーを sudouser に追加する
 - はじめに, user, Shi8UR@ で再ログインする
 - \$ sudo gpasswd -a ユーザー名 sudo で, ユーザーを追加する
 - パスワードを聞かれたら, user のパスワードを入力する
 - 自分のディレクトリへのアクセス方法の確認
 - ssh コマンドの利用
 - ssh [user名]@yli.sic.shibaura-it.ac.jp
 - → データの提出などは, 本サーバから従来通りに行う

システムプログラミング

73



tmuxのインストール

操作になれる



UNIXの歴史

UNIX の歴史

- UNIX の誕生
- C言語の登場
- UNIX の普及
- インターネット時代の OS

システムプログラミング

76



OS の歴史概略

- 1956年 GM-NAA I/O (IBMメインフレーム), OS/360 IBM
- 1965年 Multics の登場
- **1969年 UNIX 誕生/AT&T Bell (Ken Thompson, Dennis Richie)**
- 1977年 BSD (Berkeley Software Distribution) 初リリース
- 1983年 GNU プロジェクト発足 (Richard Stallman)
- 1984年 Mach OS (Carnegie Mellon University)
TRON (坂村健), MS-DOS (Bill Gates), DR DOS
- 1986年 MINIX (Andrew Tanenbaum)
NeXT (Steve Jobs)
- 1990年 MS Windows 3.0 (Bill Gates)
- **1991年 Linux (Linus Torvalds)**
- 1994年 Internet 普及はじめる
- 1995年 Windows 95

システムプログラミング

77



新しい OS (Multics)

- マサチューセッツ工科大学, Project MAC
 - GE およびベル研究所と共同で開発
- TSS のシステムとして開発
 - 現在でもなおメインフレーム OS で利用されている (1965-)
- **斬新なアイデア**
 - セグメント方式
 - 単一レベル記憶と呼ばれるデータアクセス法の実現
 - プログラムを仮想空間で実行！！
 - **マルチユーザー, マルチプロセス**
 - Multics は PL/I により記述 (30万行に及ぶ膨大な OS)

システムプログラミング

78

UNIX の誕生

- **1969年, ベル研究所**
 - シングルタスクの軽量なOS 作成 (Ken Thompson)
 - Multics の Multi を Uni に変えて "Unics"
 - 直接 "UNIX" と書かれて, UNIX と呼ばれるようになる
 - (命名者は Brian W.Kernighan)
- **当時のハードウェア**
 - PDP-11 :0.07MIPS (16bit bus, CPU: 6.667 MHz)
 - IBM370: 9.1MPIS (1974)
- **近年のハードウェア**
 - PCサーバ：
 - Pentium 4 Extreme Edition : 9,726 MIPS at 3.2 GHz (1コア 2スレッド)
 - Core i7 Extreme Edition 990x : 159,000 MIPS at 3.46 GHz (6コア 12スレッド)

システムプログラミング

79



C言語

- **1970年, B言語 (Ken Thompson) 開発**
 - BCPL言語を参考にする
- **1971年, C言語(Ken , Dennis M.Ritchie) 開発**
 - UNIX を DEC のミニコン PDP-11 に移植
 - 移植：特定の環境から、他の環境にソフトウェアを修正しつつ 移行させること（ポーティング）
- **1972年, UNIX を アセンブラーからC言語で書き直し**
 - → C言語は UNIX の副産物
 - C言語で書き直されたことの意義
 - UNIX は PDP-11 以外でも、多くのマシンに移植可能となる
- **1975年頃**
 - Bell 研究所から、OS のソースを世界中の大学や研究機関に安価に配布
 - → 当時、画期的な出来事。世界中の研究機関に急速に普及



"The C Programming Language", Prentice-Hall, 1978

システムプログラミング

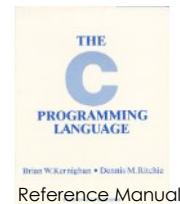
80



UNIX の普及と C プログラマの増加

- 1971年, C言語(Ken , Dennis M.Ritchie) 開発
 - UNIX を DEC のミニコン PDP-11 に移植
- 1972年, UNIX を アセンブラーからC言語で書き直し
 - C言語で書き直されたことの意義
 - UNIX は PDP-11 以外でも、多くのマシンに移植可能となる
- 1975年頃
 - Bell 研究所から、OS のソースを世界中の大学や研究機関に安価に配布

UNIX の爆発的な普及
それとともに、Cプログラマも飛躍的に増加



81

システムプログラミング



UNIX の発展

- BSD (Berkeley Software Distribution) 版
 - Ken Thompson が California 大学 Berkeley 校に赴任
- 1978年, Bill Joy (当時大学院生)
 - first Berkeley Software Distribution (BSD) リリース
 - Berkely Socket 開発 (TCP/IP)
- 1989年, System V Release 4
 - 商用UNIX としてリリース (本家 AT&T で進化を続ける)
 - UNIX v7
 - System III (1980年頃)
 - System IV, System V (Release 5)
- SUN OS (1984年ごろ)
 - 4.2 BSD UNIX の TCP/IP サポート (1984年)
 - UNIX v6でスタートし、4.4 BSD に到達

システムプログラミング

82

UNIX の優れたグランドデザイン

- 優れたグランドデザイン
 - グランドデザイン：全体を長期的かつ総合的に見渡した構想
 - E.g. 秦の始皇帝
- プログラムを書く際のメモリとCPUの抽象化に成功した
 - プログラマがメモリとCPU(時間)を意識せずにプログラムを書く事を可能とした
 - 組込みシステム(66MHz, 32MB)であっても、汎用システム(4GHz, 8GB)であっても同じコーディングスタイルを維持できる

システムプログラミング



83



UNIX の優れたグランドデザイン

- メモリ, CPUの抽象化
 - プロセスが利用
- シンメトリックなAPI
 - Data: In/out
 - File : open/close
- データのIn/Outで統一
 - 関数の設計、型とデータの名前でのやり取り
- 全てのファイルとして扱う
 - ファイルのopen/close
 - Openしたファイルポインタ(メモリの先頭番地)からデータをread/writeしてデータを読み書きする
 - ネットワークの設計(Socket)

システムプログラミング

84

マイクロカーネル

- 1980年代
 - オブジェクト指向
 - 1990前半, Java (Bill Joy)
 - 1987年 : MINIX, オランダ, タンネンバウム(4000)
- 1986年, Mach OS : CMU (カーネギーメロン) 大学
 - カーネルとそれ以外の機能を目的別に分ける
 - → マイクロカーネル
- NeXT
 - Apple 創業者スティーブジョブス
 - GUI ベース

システムプログラミング

85

インターネット時代

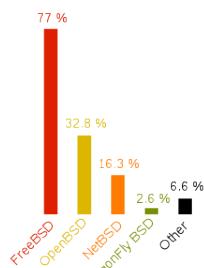
- 1969年
 - 米軍 ARPANET から始まる
 - DOD (Department of Defense、米国防総省) 内に、軍および民間の研究者を集めた A R P A (Advanced Research Projects Agency、国防総省高等研究計画局 – 1972年に DARPA(防衛高等研究計画局)に改称) を設立した。

システムプログラミング

86

Linux 登場

- FreeBSD
 - 4.3BSD → 386BSD → FreeBSD
- Minix
 - Andrew Tanenbaum
 - ライセンスの問題を回避するために自作した UNIX ライクな仕組みを持つOS
- Linux
 - Linus Torvalds は Linux を GPL のもとに公開
→ 爆発的に普及
 - Linux モノリシックカーネル vs.
 - Windows NT/Mach マイクロカーネル



システムプログラミング

87

VMS, Windows NT

- VMS
 - PDP11, VAX
 - UNIX 文化を生み出した最初のプラットフォーム, DEC のスーパーミニコン(VAX シリーズ) 上でNIX ライクOS
- 1993 年 Windows NT
 - VMS の後継OS をマイクロカーネル技術を取り入れて開発途中プロジェクト解散
 - このチームがMicrosoft 社に移り, OS2, Windows チームと合流し, 開発が進む
 - LAN の世界で当時王者の一にあったノベル, UNIX ベンダー達に大きな衝撃を与える

システムプログラミング

88

Linux の特徴

- Linux とは
 - Linux はカーネルを含む周辺技術を集約したものを指し, Linux システムと呼ぶ

システムプログラミング

89



Linux のプログラムの行数を数える

<https://www.kernel.org/>

5.3.1 (latest Stable Kernel)

```
find ./ -name *.c -type f | xargs cat | wc
```

行, 単語, 文字

```
18,740,942 56666488 506602935
```



システムプログラミング

90

Linux カーネルと Linux システム

- Linus Torvalds らによるカーネル (=OS コア部分) の公開
 - 現在の最新版 (www.kernel.org)
 - 安定版 : Kernel 3.7.10 (2013/02/27)
 - 2.6 バージョン : Kernel 2.6.33.20 (2011/11/07)
 - 2.4 バージョン : Kernel 2.4.31 (2005/06/01)
- オープンソースソフトウェア(OSS)
 - GPL のもとに公開
 - Linux kernel のソースコードは誰でも入手可能
- 活発な開発とアップデートが継続
 - 1000万行を超えるカーネルソース, 数千人規模の開発者
 - 1,500 lines/day の更新 (3,000行/day の追加, 1,500行/day の削除)
 - 最新のデバイス, プロトコルに対応

システムプログラミング

91



Linux システム

- Linux カーネルの周辺に, 様々な機能を盛り込んだシステムを構築し, 配布 (ディストリビューション)
- 数多くのディストリビューションがある
 - RedHat, SuSE, Slackware, Ubuntu, Debian, ...
- Linux のビジネス形態として成長
 - 目的や用途に応じて Linux システムの提供事業
 - エンタープライズ系 Linux
 - 組込み系 Linux
 - Linux システムのサポート事業
- 以上のように, Linux は様々なシステムを持つことまでいる
 - 組込み機器の適用能力があることを示す

システムプログラミング

92



オープンソースに基づく開発形態

- ・ オープンソースとは?
 - ・ GPL, LGPLのライセンスに基づいたソースコードの公開
 - ・ ソースコードと諸権利の切り離しは、法律的に不可
- ・ コピーレフト (Copy Left)
 - ・ 諸権利とともに、複製物を得た人が、その複製物に諸権利を含め、配布できる（プログラムを自由に配布する方法）
 - ・ 使用、コピー、再配布、改変のいずれにおいても、コピーまたは派生物にコピーレフトのライセンスを適用し、これを明記しなければならない

システムプログラミング

93



(参考) Linux と GNU/Hurd

- ・ **GNU/Hurd (開発中)**
 - ・ マイクロカーネル
 - ・ カーネル内部は、必要最低限の機能に抑える
 - ・ 必要な機能をカーネル外部に実装する
 - ・ カーネル本体の巨大化、複雑化を回避
- ・ **Linux**
 - ・ モノリシックカーネル
 - ・ 機能やサービスをカーネルの中にできるだけ組込む
 - ・ ネットワーク機能、入出力機能、グラフィックス機能、各種でバイスへのサポートなどを追加
 - ・ 複数のCPUや分散処理システムのサポートが困難

システムプログラミング

94

Linux カーネルの特徴

- ・ UNIX カーネルの踏襲
- ・ 優れたモジュール機能
- ・ マルチスレッド、プログラミング技法
- ・ ノンプリエンプティブ (Kernel 2.4まで) と、フルプリエンプティブ(2.6以降)

システムプログラミング

95



UNIX カーネル方式の踏襲

- ・ **UNIX 技術の採用**
 - ・ プロセス/カーネルモデル
 - ・ マルチユーザシステム
 - ・ マルチプロセッシングシステム
 - ・ モノリシックカーネル
- ・ ファイルシステムの装備
- ・ プロトコルスタックの装備

システムプログラミング

96

モジュール機能

- 容易なデバイスドライバの追加, 削除
 - カーネルコンパイル時に選択
 - 静的に組込む方式：起動後にユーザーが追加/削除できない
 - 動的に組込む方式：起動後にユーザーが追加/削除できる
 - これにより、ホストマシンの性能や用途に応じ、柔軟なカーネル構築が可能となった
 - ホスト、マシンのハードウェア資源に特化した取捨選択
 - カーネル起動時間やサイズのチューニング

システムプログラミング

97



マルチスレッドプログラミング

- マルチスレッドプログラミング
 - 授業内ででもプログラムを作成する
- 複数のスレッドを切り替えて、同期実行
 - マルチプロセスでは、メモリ空間の切り替えのオーバヘッドが大きい

システムプログラミング

98

プリエンプティブカーネル

- プリエンプションとは
 - 割込みへの応答機能を改善することで、処理スピードを改善
 - 特権モード処理中であっても、割込みスケジューリングが可能となり、プロセスの並行動作をスムーズにする
- Linux では
 - Kernel 2.4 まではノンプリエンプション
 - Kernel 2.6 からは、改善される
 - $O(1)$ スケジューラの採用
 - キューにたまつたタスクの遅延軽減
 - 2つのRun Queue の装備と切り替え
 - カーネル内部処理中であっても、スケジューリングを行う
 - CFS スケジューラ採用 (2.6.23)

システムプログラミング

99



Lets' Start Shell Programming

システムプログラミング

101

OSによる抽象化

- ハードウェアの仮想化、抽象化(アブストラクション)
 - ハードウェアの違いを隠蔽し、ユーザがプログラムを実行するための使いやすい環境を提供する

抽象概念	ハードウェア機能
プロセス	プロセッサ、メモリ
ファイル、ディレクトリ	ストレージ
プロセス間通信	コンピュータ間の通信
シグナル	割込
アクセス制御	コンピュータ共有時の保護

- 基本的な抽象概念の考え方はめったにかわらない(30年以上かわっていない)

システムプログラミング

104



どんな仕組みか？

- ログインしているとは?
 - PCの場合、ログインしている人 = PCを使っている人
 - Unix マシンの場合、ログインしている人が非常に多い（数百人にもなる）場合がある
- ログインからの流れ
 - ユーザー名、パスワードの入力
 - システムはシェルを起動する
 - シェルを、ユーザに結びつける
 - ログインしているユーザはそれぞれ異なるシェルにむすびついている
 - ユーザとシステムの接続はカーネルがコントロールしている



システムプログラミング

108

ユーザ視点からみた Unix

- Unix は何をしているのか
 - まずはログインして確認する
 - Linux の利用方法について説明する
- ログイン～プログラム実行～
 - Linux 1.2.13 (maya) (ttyp1)
 - maya login: doly
 - Password: *****
- ログアウト
 - \$ exit

システムプログラミング

106

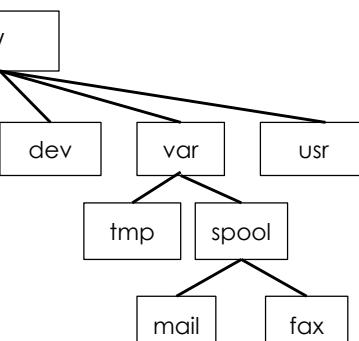


ディレクトリツリー

- ディレクトリツリー
 - Unix は、ファイルを木構造のディレクトリシステムに組織し、その内容を表示したり、位置を移動するためのコマンドを提供している
- ルートレベル
 - /etc, /home, /bin...etc
 - ユーザデータ /home/user
- 操作コマンド
 - ls list
 - cd change directory
 - pwd print working directory

システムプログラミング

109



ファイルの操作

- ディレクトリ

- ファイルの格納システムを提供する
- ユーザ
 - ホームディレクトリまたはその下のディレクトリに、個人ファイルを格納
- システム
 - システムディレクトリにファイルを格納する

- ファイル操作のためのコマンド

- ファイル名
 - ファイルは名前を持つ
 - ファイル名には "/" を除くすべての文字を使用できる
 - 大文字と小文字の両方を使える
 - 大文字と小文字は区別される

システムプログラミング

111



ファイル操作

- ディレクトリ操作

```
$ mkdir /home/doly
```

```
$ mv first_program.c mycode
```

- ディレクトリ名を指定すると、ファイルを別ディレクトリに移動することができる

- ファイルコマンド

- どのような仕組みか?
 - ユーザ：ファイルを情報のかたまり（ドキュメント）とみる
- これらは、どのようにディスクに格納されているのか？

システムプログラミング

113

ファイル操作

- ファイル操作

```
$ cat shopping-list
```

Milk

Apple

Jam

```
$ more longfile
```

- ファイルのコピー、削除

```
$ cp shopping-list last-week.list
```

- ファイルがコピーされる

```
$ rm old.data shopping.june2012
```

- ファイルが削除される

```
$ mv prog1.c first_program.c
```

- ファイルの移動（名前の変更）

システムプログラミング

112



ファイル操作

- ディレクトリ操作

```
$ mkdir /home/doly
```

```
$ mv first_program.c mycode
```

- ディレクトリ名を指定すると、ファイルを別ディレクトリに移動することができる

- ファイルコマンド

- どのような仕組みか?
 - ユーザ：ファイルを情報のかたまり（ドキュメント）とみる
- これらは、どのようにディスクに格納されているのか？

ファイル操作

- ディレクトリ操作

```
$ mkdir /home/doly
```

```
$ mv first_program.c mycode
```

- ディレクトリ名を指定すると、ファイルを別ディレクトリに移動することができる

- ファイルコマンド

- どのような仕組みか?
 - ユーザ：ファイルを情報のかたまり（ドキュメント）とみる
- これらは、どのようにディスクに格納されているのか？
- ファイルのコピーはどのように作られるのか？
- ファイルを別のディレクトリに移動するにはどうすれば良いのか？
- システムは、どのようにしてファイルの名前を変更するのか？
- そもそもファイルは、どのようにして名前を持つのか？

システムプログラミング

114

ファイル属性

・個々のファイルの属性

- ・ファイルはオーナー、パーミッションを持つ
 - ・ユーザーがそれぞれのファイルに対するアクセスをコントロールできるようにするために、Unixは個々のファイルにいくつかの属性を割り当てている

\$ ls -l file.txt

- ・rwx rwx rwx
- ・ユーザ、グループ、個人
- ・権限 r: 読み出し, w: 書込み, x: 実行

\$ touch test

\$ ls -l test

- ・-rw-r--r-- 1 doly staff 0 3 21 11:17 test

システムプログラミング



ファイル操作

・ファイルの書き込み権限の変更

- ・他人に、ファイルの実行権限を許可するには？
- ・他人に、ファイルの書き込みは認めるが、読み出し禁止とするには？

\$ chmod a+x test

- ・ls -l test
- ・-rwxr-xr-x 1 doly staff 0 3 21 11:17 test
- ・全てのユーザーにTest ファイルへの実行権限を与える

\$ chmod g+w,g-r test

- ・ls -l test
- ・-rwx-wxr-x 1 doly staff 0 3 21 11:17 test

システムプログラミング

116

システム管理コマンド

・システム管理とは

- ・システム(システムリソース)を管理すること
 - ・情報システム
- ・管理
 - ・監視：システムに異常が発生していないかを監視する
 - ・回復：異常が発生していたら、適切に回復する
 - ・保守：異常が発生しないように、常時監視、予防などの措置をとる

・システム異常の例

- ・システム(リソース)異常
 - ・CPU: CPUを使い切ってしまう
 - ・あるプロセスが暴走し、CPU 100% 使ってしまう
 - ・ディスク：
 - ・書き込んだ情報で、ディスクが一杯になって書き込めない、故障する
 - ・メモリ：メモリを使い切って、プロセスが動作しない

システムプログラミング

115

117



システムの監視

・以下のコマンドを man をみながら、どのような情報が収集できるか確認せよ

プロセス	
ps	Report a snapshot of the current processes - A : for all processes
pstree	Display a tree of process - ah : for all tree
date	Print or set the system date and time
last	Show listing of last logged in users

システムプログラミング

118

コマンドの操作

- リダイレクト
 - > ファイルに出力
 - >> 出力をファイルに追記
 - < 入力のリダイレクト
 - << 入力終端文字の指定
- 確認してみよう

```
$ echo "hogehoge" > file.txt  
$ echo "fugahuga" > file.txt
```

```
$ cat << _END_ > result.txt
```

```
> Hello!!  
> I'm Midori Sugaya  
> _END_
```

```
$ less result.txt
```

- エディタを使わずに、ファイルを作成することができる

システムプログラミング

119



コマンドの操作

- “|” パイプ
 - コマンドの出力を次のコマンドの出力として渡す
 - コマンド | コマンド
- 試してみよう

```
$ cd /etc  
$ ls -l | less  
$ find . | grep cron | less
```

- tee

- 標準入力を標準出力とファイルに出力する

- 試してみよう

Telenet のログを保存する

```
$ telnet xxx.shibaura-it.ac.jp | tee telenet.log
```

システムプログラミング

120



システムリソースの監視

- vmstat
 - Report memory resource statistics
- Check man vmstat
 - Process r/b
 - Memory swpd/free/buff/cache/inact/active
 - Swap si/so
 - IO bi/bo
 - System in/cs
 - CPU us/sy/id/wa/st
- Option
 - vmstat -d
- Usage
 - vmstat > vmstat.log

システムプログラミング

121



システムリソースの監視

- top
 - Display Linux processes
 - どのような情報が示されているかを確認しよう
- Top によるバッチモードの利用
 - \$ top -b -d 1 -n 10
 - バッチモード、1秒間隔、10回、実行
 - \$ top -b -d 1 -n 10 | tee top_batch.txt
 - tee: 標準出力と、ファイル出力への書き出し

システムプログラミング

122



システムリソースの管理

- **du**
 - Estimate file space usage
\$ du -sh /*
 - -s summary, -h human readable

\$ du /* | sort -nr

- 集計後、容量をサイズが大きい方から順に出力

\$ du /* | sort -nr | head -10

- 上から 10 行 (10位) まで表示

\$ find / -size +1024k -ls

- **df**

- Report file system disk usage

\$ df -h

- パーティション別の総容量、使用量、空き容量、利用率を調査

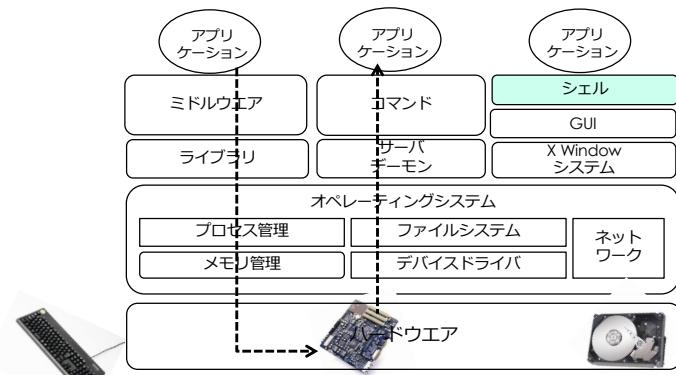
システムプログラミング

123



システムの構成要素

- 主に UNIX 系オペレーティングシステム
 - 様々なプログラムから構成される



システムプログラミング

125



ファイルの圧縮、展開コマンド

- **bz2, bunzip2**
 - Burrows-Wheeler ブロックソートテキスト圧縮アルゴリズム + Huffman コード化
\$ bzip2 file.txt
\$ bunzip2 file.txt.bz2
- **gzip, gunzip**
 - UNIX で古くから使われていた Z 形式
\$ gzip sample.txt
\$ gunzip sample.txt.gz
- **tar**
 - Gzip と組み合わせて使われる。ディレクトリごと圧縮できる
\$ tar cvfz same_dir.tar.gz same_dir/
\$ tar xvfz same_dir.tar.gz

システムプログラミング

124



シェル

- UNIXユーザのユーザインターフェースプログラム
 - OSを操作するためにOSを取り囲んでいる殻
- **具体例**
 - BSH (Bourne Shell), BASH (Bourne Again Shell), CSH (C Shell), TCSH (Tenex-like C Shell) などがある。
- **役割**
 - CLI (Command Line Interface) を通して、ユーザからの命令を受け付け、解釈、実行し、その結果を出力する。
- **コマンドの入出力**
 - リダイレクションやパイプ機能を提供する
 - 複数の比較的単純な機能を持つコマンドを組み合わせて使用できる
- **簡便さ**
 - 実行するコマンドを記述した簡易プログラムとしてシェルスクリプトを作成できる、
 - この機能と合わせて、プログラミングの素養を持つユーザには非常に強力なインターフェースを提供している。



システムプログラミング

126

シェルスクリプト

- ・スクリプト
 - ・コンパイルが不要な言語の総称
- ・シェルスクリプト
 - ・ファイルに保存しておいて、プログラムのように利用できる
- ・利点
 - ・豊富なUNIXコマンド群を有効活用
 - ・ディレクトリ内にあるファイル全てに対して、プログラムによりある処理を施す
 - ・複数のファイルから、ある特定の文字列を含むファイルのみに対して何らかの処理をする
 - ・少し複雑な処理程度であれば、C言語でプログラムを書かなくても、シェルの機能で簡単にプログラムを実行できる

システムプログラミング

127



条件文, 制御文

- ・基本の構造
 - ・if 条件文
then
 実行文
elif 条件文
 実行文
else
 実行文
fi
- ・よく使う例
 - ・if コマンドが正常に終了した
then
 通用の処理
else
 エラー処理
fi

システムプログラミング

129



シェルスクリプトを書いてみよう

・ 基本

#	コメント行
#!/bin/sh	実行スクリプトの指定
変数	英字または_で始まり、2文字以降は数字も可能
変数への代入	=“で代入、val=3 str=ABC 空白文字やメタキャラクタを含んだ文字列の場合は'または"で囲む、'This is' "This is"
変数の参照	変数名の前に\$をつける

・ 構文

- ・ if, case,
- ・ for, while, until, break
- ・ exit

システムプログラミング

128



練習1

・簡単な管理コマンドの実行

- ・コマンドを複数連続して実行するシェルの実行プログラムを作ってみよう！（下記は、pwd, ls）
- ・emacs intro.sh

```
#!/bin/sh
echo "**** Current Working Directory ****"
pwd
echo "**** Directory Listing ****"
ls
```

・実行方法

- ・sh intro.sh でも良い

システムプログラミング

130



演習1

・練習1について

- ・ファイルのパーミッションを変更し, ./intro.sh として実行できるようにしてみよう
- ・pwd, ls 以外のコマンドに変更してみよう.
- ・シェルスクリプトの利点を隣の人と相談してみよう
- ・どのようなコマンドだとよりメリットがあるか相談してみよう

システムプログラミング

131



演習3

- ・あるディレクトリ中の、拡張子が.cのファイルすべてに対して、.bakという拡張子をつけてバックアップをとる。
- ・以下の内容のファイルをbackup.sh としよう。

```
#!/bin/sh
# save files with .bak as backup

for file in *.c
do
    echo $file
    cp $file $file.bak
done
exit
```

システムプログラミング

133

演習2

・便利な管理コマンドの実行

```
#!/bin/sh
echo "**** Current Working Directory ****"
pwd
echo "**** Check Disk Usage****"
du /* | sort -nr | head -10
```

- ・利用されているディスク使用量の数字の結果が読みづらいので、読みやすくできないか相談してみよう
- ・ヒント：
 - ・man du でマニュアルがある
 - ・du のオプションを調べてみよう

システムプログラミング

132



練習

- ・シェルの for 文の使い方に慣れていない人へ
 - ・下記のプログラムを作成してみよう
 - ・for x in リスト(で、リストから x 順番にを取り出す) 部分を理解しよう

```
#!/bin/sh

for i in a b c
do
    echo $i
done
```

システムプログラミング

134



演習4

- 3秒 sleep する操作を 10 回カウントして終了する

```
#!/bin/sh
COUNT=0
while [ $COUNT -ne 10 ]
do
    COUNT=`expr ${COUNT} + 1`
    echo "${COUNT} 回目の処理"
    sleep 3
done
```

システムプログラミング

135



課題

・課題1. 以下のシェルスクリプトを作成せよ

- ディレクトリを一つ作成し、そのディレクトリ内に.cを拡張子とするファイル（中身は何でもよい）を3つ程度作る。
- 2秒おきに、バックアップを取る。5回で終了する。
- 演習内容を組み合わせて実現する。
 - デバッグ時は、\$sh -x backup.sh で出力して確認しよう。

・課題2.

- 本日行った管理コマンドを利用し、課題1で作成したシェルに1つ以上機能を追加してみよう。
 - 例) バックアップをしたファイルを圧縮する（ディスクスペースの有効活用）
 - 例) ログ機能をつける

・発展.

- Python で同じコードを書いてみよう

システムプログラミング

136

課題の提出

- Exec01
 - Exec01.sh // 課題1
 - Exec02.sh // 課題2
 - ExExec01.py // 発展1
 - README // 作成したプログラムの説明を入れる
- として、学籍番号、ローマ字名前フォルダに入れて、シェアフォルダに提出
- 課題を提出する締め切り日程は授業内で伝えます

システムプログラミング

137

