クラウドコンピューティング
第01回 オリエンテーション,概要

3年、秋学期、選択; 旧「オペレーティングシステム」

オリエンテーション

(脚注) もちろん、オリエンテーションは期末試験の範囲外ですが、スライドの後半は試験範囲です

対象者、大前提、履修登録について 【とても大事】

- おもに情報システム工学科3年生が対象者ですが、他学科の方も履修できます
- ただし、大前提として「情報システム開発基礎演習」の単位取得者だけが受講可です
  - 具体的には、 「情報システム開発基礎演習」インフラ編(#01-06)のシステム構築が問題なくできる人だけ受講してください。
  - 本科目の授業時間内では、春学期の範囲のサポートは行いません。 メディアコンサルタントさんに聞いて各自、復習してください
  - **上記科目を未履修でもAWS ACFなどの資格保有者は受講可**です
- 上述の前提を満たさない方は単位取得が困難です。**履修登録の変更をおすすめ**します

(脚注) 春学期のシステム構築支援システムColitasは今回も利用可能です。第2回から登場予定

授業の運用形態

- コンピュータネットワークと同様のダンドリで授業を進めていきます
  - 予習、ELでの確認テスト、解説、そのあとは各回、演習を行います
  - 第2回以降では、予習が必須です
- 期末試験以外は、すべてオンラインの予定
- 知識試験(EL)は定期試験(期末試験)の枠で、場所は大学で、行います

目的,目標

- システム構築の実務/実作業をやりますが、技術的には、大半が春学期の復習です
- 春学期の演習内容をなぞりながら、 システムの状態の調査をしつつ、 その**コマンド操作の背景にある概念を考えていきます**
  - 春学期の演習内容の裏側にある**概念**をキチンと見ていくことが主目的
- 科目名がクラウドなので、 最後は **クラウドらしいシステム**を作ることを課題とします
  - **クラウド(AWS)らしいシステムとは?** (詳細は第10回あたり〜)
    1. スケールアウト/スケールインできる
    2. ステートレスなアプリケーションサーバの作成
    3. むずかしい分散システムのところはAWSのサービスに頼る

(脚注) いわゆるOSの動作原理をあつかう科目は大学院の授業へ移しました

この授業は、何をやり、何をやらないのか

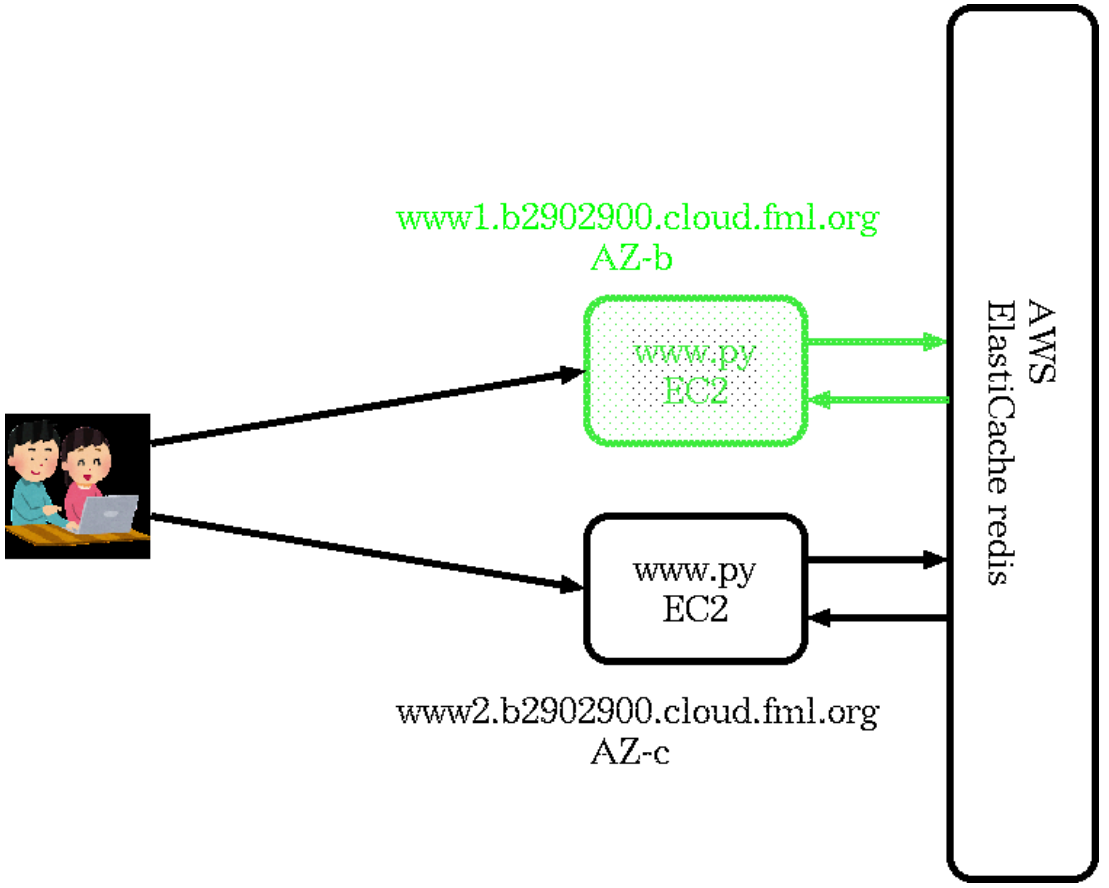
- 何をやるのか
  - 春学期のコマンド操作の復習
  - そのコマンド操作の背景にあるOS概念の説明。 それにともなうOSの内部構造の表面的な説明
  - クラウドらしいシステムの構築 (秋学期NEW)
- 何をやらないのか
  - 資格試験の勉強
  - OSの内部構造(カーネルが提供する機能)の説明(脚注1)
  - (ミニチュアUnix)カーネルのソースコード(1万行)を読む(脚注2)

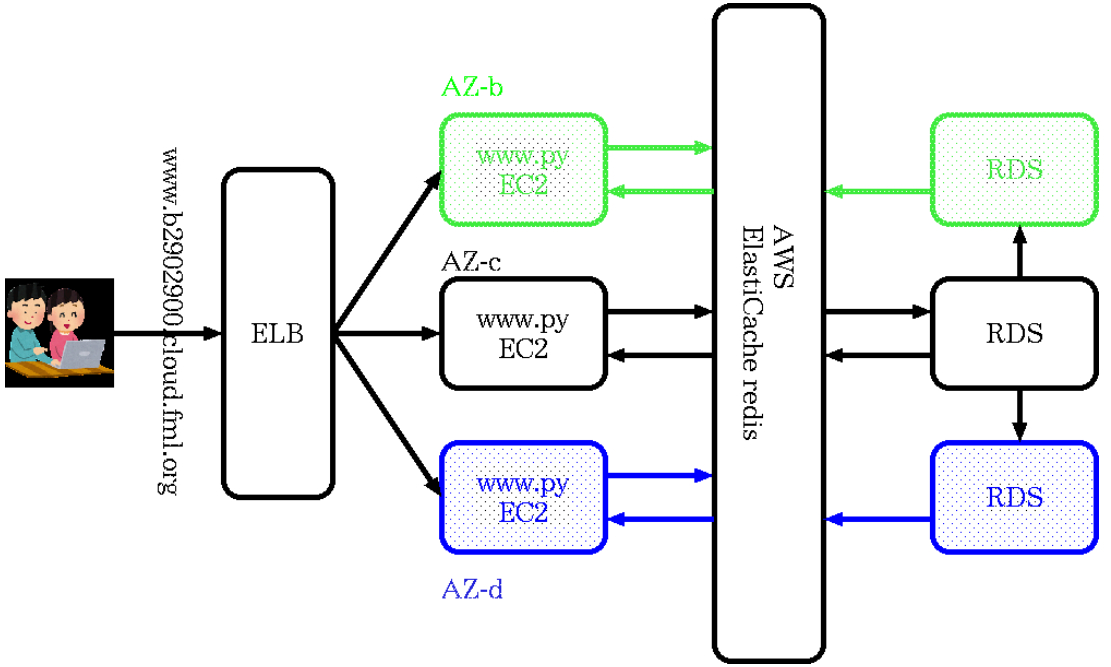
(脚注1) いわゆるOSの動作原理をあつかう科目は大学院の授業へ移しました

つまり、たいていの理工系大学学部3年にある「オペレーティングシステム」という科目は大学院で行うという意味

(脚注2) 世界最高峰の理工系大学MITの学部3年の授業「オペレーティングシステムエンジニアリング (6.828) 」は、 これをやっています。 たまに、うちの研究室の有志(大学院は大前提)でやることがありますが、 相当Unixが使えないと話についていけません

最終課題の最低ラインと理想形(予定)





(脚注) 最低ライン課題(左側)は「ショッピングカートの中身が、システムが一部壊れても消えない」ことです

OSの概要

第2回からが本番ですが、軽く基本的な概念や用語を先出しします(**期末試験の試験範囲、なお脚注は範囲外**)

デジタルコンピュータが提供するもの

- 現代では主に**ユーザが使いやすい「GUIベースで対話的に利用する」仕組み**の提供
  - デジタルコンピュータという機械を生々しく操作すれば最高効率で使えるはず
    でも、それは(一般のユーザにとっては)非常に難しい
  - ユーザが使いやすいように、コンピュータに何層か(抽象化する)システムをかぶせています
- OSの種類(代表的な2種類をあげると...)
  - **タイムシェアリングシステム(本科目の対象) ... 対話的,複数ユーザが同時に利用可**
  - リアルタイムOS ... 自動車やロケットなどミリ秒単位で制御したいデバイスで用いる

(脚注1) コンピュータというハードウエアについては、 たいてい「コンピュータアーキテクチャ」という科目があつかう内容の復習

(脚注2) 2024/09 Linuxカーネルにリアルタイムの機能が追加されたので、ちかぢか各Linuxディストリで利用可になるはず

コンピュータの生ハードウエアの操作は辛すぎる

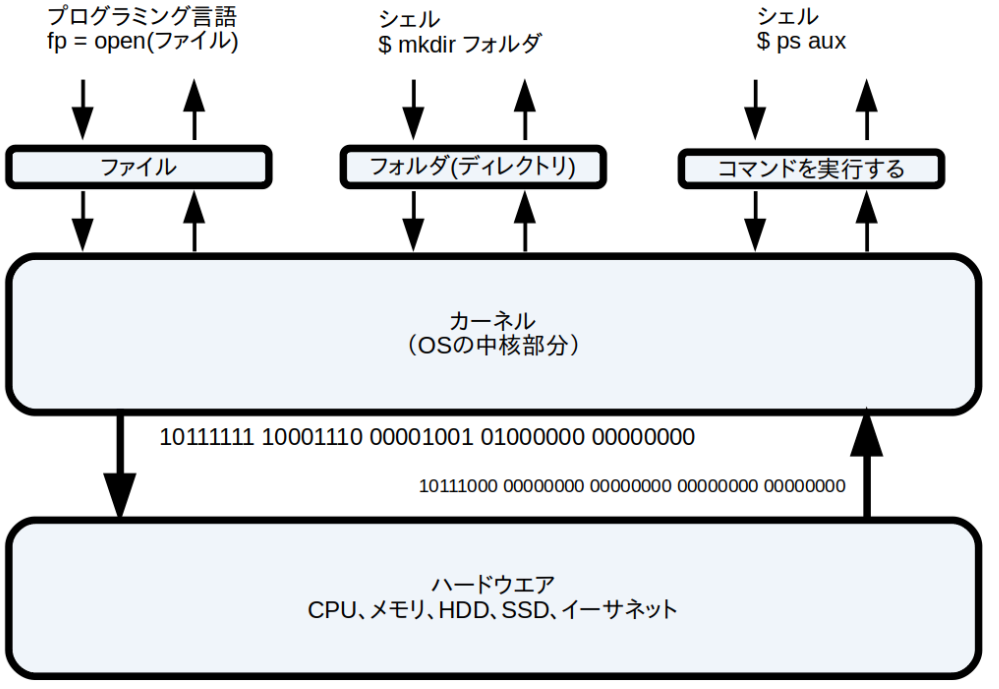
```
01010101
01001000 10001001 11100101
10111111 10001110 00001001 01000000 00000000
11101000 11111000 11111011 11111111 11111111
10111000 00000000 00000000 00000000 00000000
01011101
11000011
```

- コンピュータとは2進数の命令を処理する機械のことです

(脚注) この例ではmain関数でprintf()をするプログラムを生書き(2進数表記)しています。 HDDへの読み書きもネットワークの送受信も、こういう2進数の群れを打ち込んで指示を出すことになります。 1950年代くらいだと、これをやっていたのですが、さすがに、達人のみなさんといえども、これはツラい。 それで、プログラミング言語とかOSが開発されてきたわけです

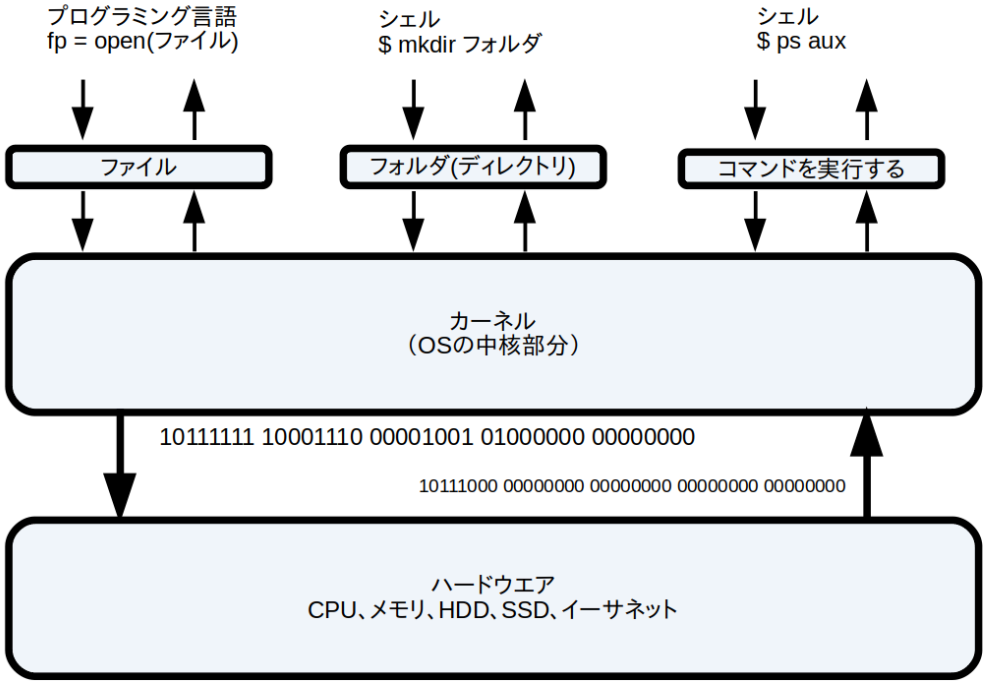
OSの役割- ハードウエアを使いやすくするシステムソフトウエア -

- ハードウエアを抽象化して扱いやすくする**システムソフトウエア**がOS
  - 0101...ではなく分かりやすい何か
  - ファイルとかフォルダとかネットワークとかアプリのアイコンとか...
  - グラフィカルユーザインターフェイス(GUI)
  - 各種デバイス(例:USBメモリ)も生操作は大変ですが、 自動的に認識したり、 読み書きをドラッグ＆ドロップ出来たりするのは、 途中に挟まっているOSが色々してくれるから



用語を少々

- **カーネル** ... OSの中核部分のプログラム
  - 例: linux
- **ユーザランド** ... ユーザに見えているOS部分の総称。コマンド群のことと考えてよい
- コマンド ... Unixで提供されているプログラム
  - 例: ps, ss, sudo, python3
- **シェル** ... (a)コマンドを打ち込んで結果を表示しているプログラム (b)**人間と対話処理をする**
  - 例: bash
  - 例: `$ ps aux`の`$`はシェルが「命令まち」を意思表示する特殊文字(**プロンプト**)




```
$ ps aux
```

(脚注1) 本科目は「春学期のユーザランドの操作の意味と、 該当するカーネルの概念の一部を解説する」ものと言えます

(脚注2) Unixではカーネルですが、OSの教科書ではスーパバイザーと呼ぶことが多いようです。 同じ意味です

本日のTODO

本日やること、次回予告

- ポータルの出席を入れる
- ポータルの「目標設定」を入力する
- アンケートに答える
  - google form -> https://docs.google.com/forms/d/e/1FAIpQLSeB0tKeK5ZeBXSsUgVHwCf3WFyMvaaBSNyI
  - ポータルに同じURLを貼っておくので、そこからたどって回答してください
- 確認事項、該当する人は残ってください
  - AWS Academyからの招待メールが来てない
  - ELのコースが見えない
- 来週(10/04)は休講なので、第2回は10/11です

クラウドコンピューティング

第02回 概念「権限」

3年、秋学期、選択; 旧「オペレーティングシステム」 ;脚注はELや定期試験の範囲ではありません

シリーズ「権限」

- 02-05
  - 02 全体像
  - 03 ユーザ
  - 04 プロセス
  - 05 ネットワーク

(脚注1)脚注はELや定期試験の範囲ではないので、読み飛ばしてかまいません

(脚注2) マルチユーザ = タイムシェアリングシステム(TSS)というわけでもないのですが、 TSSならマルチユーザな使い方をしたいはずなので、 TSSではユーザの区別やユーザ権限の問題が自動的に付随してきます

身近にあるコンピュータの特徴は？

- 現代で身近なPCやAndroidは、タイムシェアリングシステム(TSS)という種類のOS
- TSSの特徴は、**マルチユーザ**、**マルチプロセス**です
  - マルチユーザとは、一つのコンピュータを複数人で同時に利用すること
  - マルチプロセスは、同時に複数のプロセスが動いているということ
  - 現代の個人利用の場面では、TSSのメリットが分からないでしょうが(脚注2を参照)、
    サーバは、この仕組みのおかげで効率的に動作できています

(脚注1) マルチユーザ環境なら当然プロセスは複数うごいているはずだからマルチプロセスが要求されますよね？

(脚注2) 現代では、**PCが安価で高速**なので、マルチユーザな使い方をしていません。 TSSが考え出された1950年代のコンピュータは非常に高価でした。 1960年代前半に「20億円ほどのコンピュータを同時に30人で使える！」が世界最先端だった時代です。 ちなみに、そのコンピュータの速度は、みなさんのPCの数百分の一くらいの速度です。 高価なコンピュータを効率よく複数人で利用するための工夫がTSSでした。 コンピュータの値段が下がってきて、1970年代末には、頑張れば個人でもコンピュータが買えるようになりました。 つまりパーソナルコンピュータ(PC)です。 個人用途のPCなら、マルチユーザ仕様でなくとも良いのです

身近なOS(TSS)の例

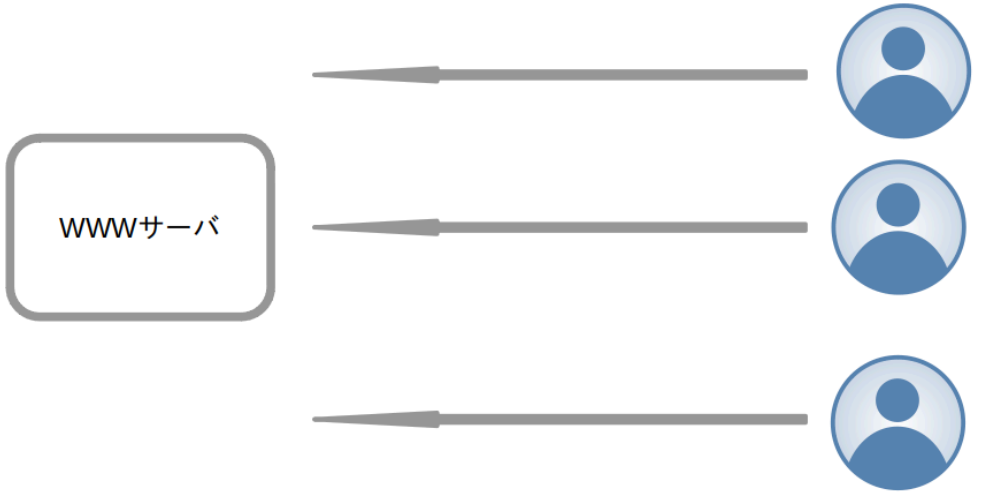
- Linux
  - オリジナルUnixそっくりの動作をするOSを新たに作った
  - こういったOSをUnixクローンと呼ぶ
  - 例: 演習で使う Debian GNU/Linux
- AndroidやChrome OS (ChromebookのOS)
  - LinuxをベースにGoogleが魔改造したOS
- 商用のMicrosoft WindowsやApple MacOSX

(脚注) この商用の二つは1980年代に流行したマイクロカーネル型分散オペレーティングシステムの系譜です。 マイクロカーネルは裏の裏の話なので見かけは普通のTSSっぽいです。 ちなみにMacOSXのターミナルがUnixらしく見えるのは、 BSD (バークレイUnix)のソースコードを流用してAppleが作成しているからです

WWWサーバには同時にたくさんのユーザがアクセスしている

(権限ではなくTSSの補足ですが)

- ユーザの手元のPC(WWWクライアント)は各自ひとりで占有できていますが
- WWWサーバは、同時に複数のHTTPアクセスを受け、複数ユーザの相手をしています。 TSSは、この同時アクセスを効率的にさばくことに役立っています



(脚注) WWWサーバが複数ユーザのコンテンツを裁いている場合、 ユーザごとのコンテンツ(そのユーザ以外にはアクセスできないコンテンツ)があり、 WWWサーバが、そのコンテンツをWWWクライアントに返しています。 このためWWWサーバ(プロセス)には他人のコンテンツを見ることが出来る特権が必要になりますが、 できれば避けたい処理です。 うちのITインフラ演習は、 AWSで、 ひとりにひとつずつVPCを割り当てる独自プライベートクラウド状態なので、このパターンをやっていませんけど

Unixの場合: ユーザの区別

- PCもコンピュータなので管理の必要があります。 管理するうえでは、ときに危ない操作も必要です。 危ない操作は管理者だけが行えるようにシステムを作りこむべきです
- Unixには、一般ユーザと管理者ユーザ(root)の区別があります
  - EC2(debian)にログインするとユーザadminです。これは一般ユーザ
  - ユーザrootとしてログインすることはありません（普通rootログインは出来ませんし、させません）
  - Linux上で管理作業をするときにはsudoコマンドを使います(詳しくは第3回で復習します)

(脚注1) 第3-5回で、もう少し詳しく見てみます

(脚注2) Windowsにも管理者ユーザはあるのですが、 意識しなくても使えてしまうのが問題です。 メニューにある「管理者として実行」を(パスワードなしで)選択・実行する時、 Linuxのsudo相当の操作をしているはずですが、そうは意識していないでしょう。 それより、なにより、「(パスワード)認証なしで管理者の操作が出来るOSって、どうなんだ？」と、ツッコムべきところでしょうや？

Unixの場合: ユーザとプロセス

- 動いているプログラムのことを**プロセス**と呼んでいます
- ユーザが実行したプロセスをOSは追跡しています
- ユーザXが実行したプロセスAには、ユーザXの権限が関連づけられます
  (業界人いうところの「紐づきます」)
  - 一般ユーザが実行したプロセスなら、一般ユーザの権限
  - 管理者ユーザrootが実行したプロセスなら、管理者権限
    - OSの危ない操作をするには管理者権限が必要です
    - つまりユーザrootだけが「危ない操作をするコマンド」を実行できます

(脚注1) 第4-5回で、もう少し詳しく見てみます

クラウドコンピューティング
第03回 ユーザの権限

3年、秋学期、選択; 旧「オペレーティングシステム」 ;脚注はELや定期試験の範囲ではありません

TSSはマルチユーザ

- TSSの特徴の一つはマルチユーザです(前回の復習)
- 当然コンピュータはユーザを区別する必要があります
- コンピュータには安易に操作してはいけない機能や部品があります
  - つまり、ユーザには「やっていいこと」「わるいこと」があります
  - 大きく分けて、一般ユーザと管理者ユーザ(root)があります
  - ようするに管理者とは「危ない操作ができる」ユーザです

ユーザの区別

- コンピュータは数字をあつかうのが得意です
- 各ユーザには登録時に一意の数字を割り当てます
  - Unixではuid(user idの略)と呼んでいます
  - ただしuid = 0は特別で、管理者ユーザです
- 設定は/etc以下にあるファイルに書かれています(脚注)

(脚注1) 演習で確認しましょう

(脚注2) ユーザの登録や削除は専用の管理コマンド(たいていuseradd/userdel)で行います。 ふだんは、サーバ管理者でない限り使うことのないコマンドですが、 Dockerfileを書こうとすると必要になる知識でしょう。 本科目の標準コースではdockerは扱いませんが、 授業設計の想定としては、 簡単なdockerコンテナの設定ファイル(Dockerfile)を読み書きできる程度の内容を考えているのです

一般ユーザadminに、できること、できないこと

- できること
  - 日常よくおこなう一般操作はデフォルトで許されています(そうでないと不便でしょ?)
    例: ファイルの作成や編集、アプリの操作、ネットワークの利用
- できないこと
  - Unixの管理(つまり危ない操作)全般: OSの設定変更、電源OFFなど

(脚注1) 本科目の演習環境の場合、ユーザadminのUIDは1000です。あとで確認してみましょう

(脚注2) ユーザの設定はOSごとに異なります。これは、あくまでもAWSのdebianイメージの場合です

管理者ユーザrootに、できること、できないこと

- できること
  - 一般ユーザが出来ることは、すべて出来ます(あくまでもユーザの一人なので、当然できます)
  - 管理コマンド群を使う各種Unixの管理: ユーザ、ファイルシステム、デバイス、ネットワーク ...
  - サーバの起動/停止/再起動
  - カーネルチューニング(カーネルのパラメータを変更することで挙動を少し変更できます)
- できないこと
  - ほぼ、ありません(脚注2)

(脚注1) ユーザ権限は (a)プロセスにも (b)プロセスが利用する機能(e.g. network)にも影響を与えます。 詳細は、(a)は第4回、(b)は第5回

(脚注2) ふつうハードウエア(PC)に干渉する操作は出来ません。 PCと協調できればUnix側から操作できることもあります。 たとえばUnix側から電源OFFが出来ますが、 これはPCが提供する仕組みを使っています。 実行中に、カーネルの動作を大きく変更することも出来ません。 もちろん、Unixのソースコードを改変してリビルドし、Unixの中身を入れ替えるなら、(Unixの変更は)なんでもできます

一時的にroot権限を取得して管理コマンドを使う

- ユーザrootは極力つかわない、必要な場面で一時的に使う**(最小権限の原則)**
- 権限を奪取するコマンド
  - su ... Unix伝統のコマンド
  - sudo ... Linux文化圏の人たちが作ったコマンド
    (演習環境はdebianなので主にsudoを利用)

(脚注1) あらゆる設計において「"Principle of Least Priviledge" (最小権限の原則)」は正しいので、 よく覚えておいてください。 そして、覚えるだけでなく、この原則を心がけて設計してください

クラウドコンピューティング
第04回 プロセスとファイルとユーザ権限

3年、秋学期、選択; 旧「オペレーティングシステム」; 脚注は試験に出ませんので飛ばしてOK

プロセスとファイルとユーザ権限

- 実行中のプログラムイメージを(総称して)**プロセス**と呼んでいます
- ユーザAが開始したプロセスXは「ユーザAのプロセスX」と認識されます
  - 例:一般ユーザadminが起動したpython3には、ユーザadminの権限だけが関連づけられています
  - 例:管理者ユーザrootが起動したpython3は、rootで出来ることが何でも出来ます
- 当然、このプロセスXが操作できる対象もユーザAが操作可能な対象と同じ範囲です
  - ファイル関係は、次ページ以降で説明しますが
  - ネットワーク関係の制限もあります(第5回へ続く)

(脚注1)「総称」という表現が苦しいところですが、 メモリ上のイメージ、カーネル内のデータ構造などなど全部をまとめてと言いたいため、 こういう表現になります。 ソースコードのレベルで理解しないと何を言っているのか具体的には分からないでしょう

ファイルには属性がある

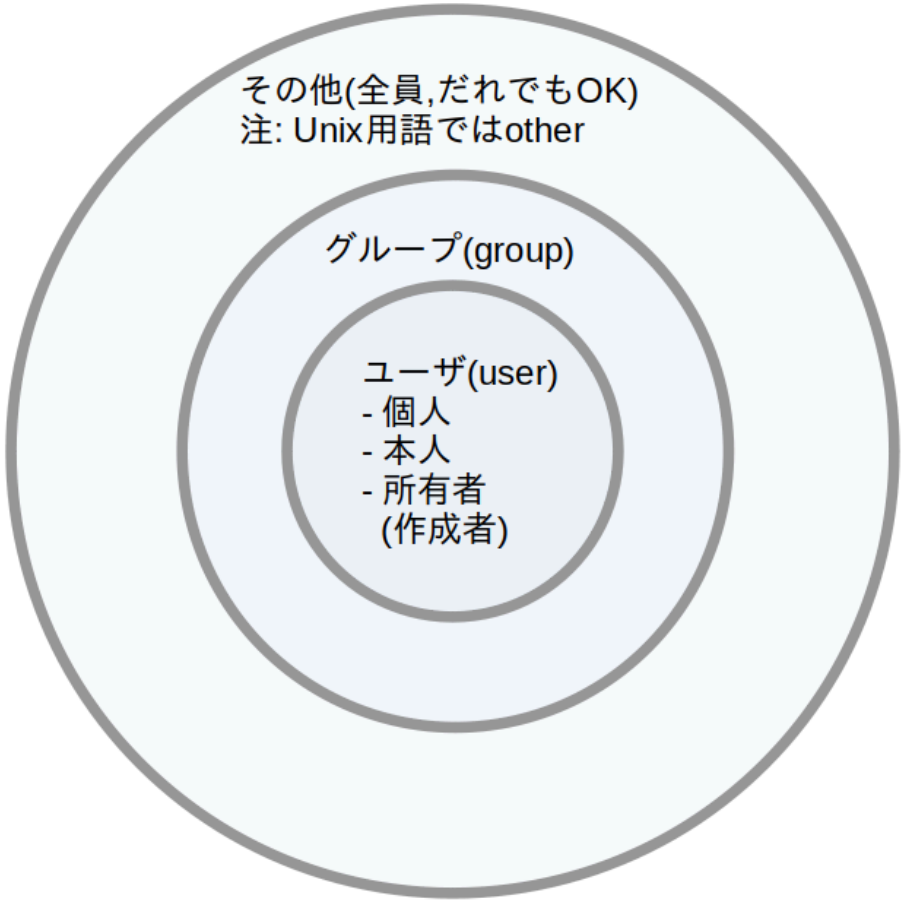
- プログラムの大元はUnix上のファイルです。典型例はsudoやwww.pyですね
- ユーザの権限で「(自分が所有者)のファイルの属性」を変更できます
- ところで、Unixでいう(権限関連の)「ファイルの属性」とは何でしょうか？
    - 「できること」という観点で考えると、大きく分けて3種類あります
        - だれが**「読み」「書き」**できるのか？
        - だれが、そのファイルを、プログラムとして**「実行」**できるのか？

```
[例] (おいおい復習していくけれど)一番下のls -lの出力例 rw-r--r-- 部分が権限の属性を意味しています
$ curl -o www.py http://api.fml.org/dist/www.py
$ ls -l www.py
-rw-r--r-- 1 admin admin 6370 Oct 01 08:16 www.py
```

(脚注1) lsはファイルの情報を表示するコマンド(1年生の復習)です。 (脚注2) ふだん、いい加減なWindowsの運用をしているだけでWindowsのファイルにも内部的には似たような概念/区別があります。 たんに、ほぼ人類の誰も、その機能を使って無いだけ:-)

Unixセキュリティモデルとファイル属性

- ユーザ(個人)、(開発)グループ、全体の３とおりの区別だけあります
- 各区分の中では一律の制限になります
  - グループに対する属性を例に取ると、「グループメンバーの2人(ユーザBとC)にだけは読み書きの権限アリ、その他のユーザは読めるだけ(書きこめない=編集できない)」といった設定は出来ません。 基本は、メンバー全員が「読めるだけ」「読み書きできる」です
- この区分方法がベストというわけでも、OSの正解でもありません、というより正解はありません。 「**研究開発現場が自分たちの欲しいと考えた**」OSがUnixです。 この程度の区別で十分だとUnix開発者たちは考えていたということです

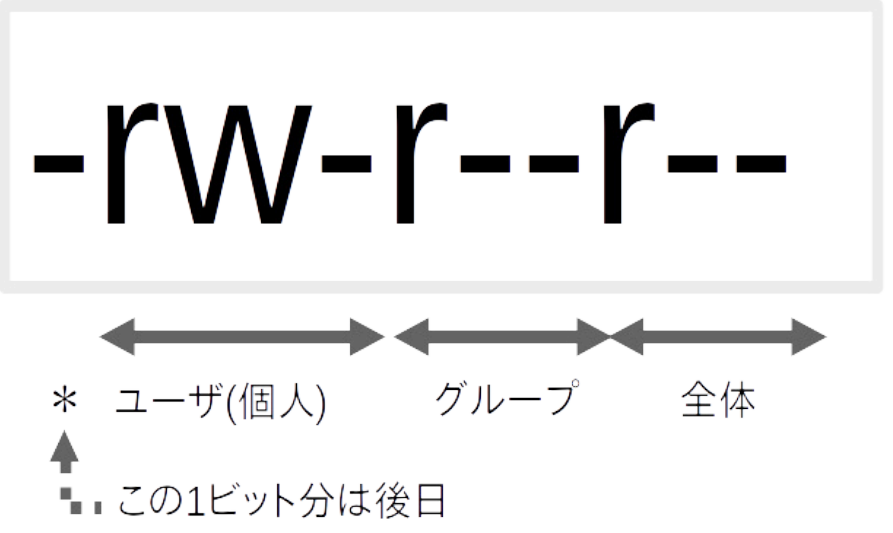


(脚注1) 実際IBMのコンピュータでは、はるかに細かい設定が出来ます

(脚注2) オリジナルのUnixではスライドのとおりですが、 より細かな設定が出来る機能もあります

Unixのファイル属性(ls -lの表示)の読み方

- Unixのファイル属性は合計9とおりあります
  - だれ？という区分が3種類
    - ユーザ(本人)
    - グループ(に所属しているユーザ)
    - 全員(だれでも)
  - それぞれに「読める」「書ける」「実行できる」属性が設定できます。つまり3x3=9とおり。できる or できないは1ビットで表現できるので合計9ビット分
- `ls -l`の表示の左端にある rw-r--r-- 部分が、この9ビットを表現しています
  - よくみると10文字分あって、左端は-かdとなっているはずです。 この1ビット分については、後日、ファイルシステムの回で取り上げます



(脚注) 各区分ごとに「読み(read)」「書き(write)」「実行(execution)」の3つの属性が関連づけられます。 それぞれの属性に1ビット分の情報(できる/できない)が必要です。 それを、単なる1 or 0 (yes or no)ではなく、 それぞれのyesの意味を"r","w","x"で表現しています。 つまり、それぞれの選択肢は"r"か-"、"w"か"-"、"x"か"-"です。 たとえば、111(yes yes yes)はrwx、110(yes yes no)はrw-、100はr--

事例: vi /etc/passwdとvi /etc/shadow (第03回の演習の解説)

```
$ ls -l /etc/passwd /etc/shadow
-rw-r--r-- 1 root root   3272 Aug 30 21:13 /etc/passwd
-rw-r----- 1 root shadow 2077 Aug 30 21:13 /etc/shadow
```

- /etc/passwd ファイルは誰でも読めます
- /etc/shadow ファイルを読めるのは管理者ユーザrootとshadowグループのメンバーのみ、
  その他のユーザ(adminも、この枠のユーザ)はファイルを読めません(右端が"- - -"であることに注目)
- より正確に言えば、 **ユーザadminが起動したviプロセスは/etc/shadowを読む権限がない**ので何も表示されないし、 下にpermission denied(権限がありません)と表示しているわけです

事例: 社内での(グループ)ファイル属性をどうするべきか？

- デフォルトは、たいてい'rw-r--r--'です。全員が読めます。以下、２、３の設定変更事例:
- 'rw-r----'の場合？
  - ユーザ(ファイルを作成した本人)とグループメンバーはファイルの中身が読めます
  - 中身を変更できるのはユーザ(ファイルを作成した本人)だけです
  - コミットするユーザは一人だけという運用ならOK(コミッター2人はダメ)
- 'rw-rw----'の場合？
  - グループメンバー全員がファイルを読み書きできます
  - メンバーが同格の立場で一つのプロジェクトに取り組める開発現場ならOK (Unix開発陣の発想)
  - 現実には、もしくは巨大な現場では、そうはならない(-> Unixの機能では対応が難しい)
    - 例: 自社の開発担当者は読み書きできるが、 下請けの別会社の人には(参考のため)ソースコードを読むことだけを許可したい。 これは、なかなか難しい(オリジナルUnixでは、たぶん無理)

(脚注) そもそも商用インターネット時代ではPCを一人で占有しているので、 Unixのパーミッションで頑張るという発想は前時代的。 ふつうは、 githubなどのソースコード管理の仕組みを使って権限を管理するのが、いまどきの運用です

事例: 「x = 実行できる」属性(1)

- ダウンロードした/home/admin/www.pyファイルを実行しようとすると実行できません(怒られます)
- **python3コマンドにwww.pyファイルを解釈させる**なら動きます。これがスクリプト言語というものです

```
$ curl -O http://api.fml.org/dist/www.py

// ダウンロードしたファイルを、そのまま実行してみますと、あれ？
$ sudo /home/admin/www.py
sudo: /home/admin/www.py: コマンドが見つかりません

// 春学期のテキストにならって実行してみると、うまくいきますね？
$ sudo python3 /home/admin/www.py
  ... snip ...
(debug) serving at port 80
```

(脚注) ダウンロードしたファイルの属性は、たいてい(元のファイルとは異なる)rw-r--r--あたりになります。 この属性は(ファイルを新規作成する)openシステムコールがシェルの設定を見ながらセットした値です。 詳細はUnixマニュアルのopen(2)とumask(2)を参照

事例: 「x = 実行できる」属性(2)

- Q: www.pyを普通のプログラムのように実行できますか？
  A: 「x = 実行できる」属性を変更すれば実行可です
- コマンド: chmod ... 属性(モード)を変更するコマンド(change modeの略)
  - 引数は２つ。「属性の変更の仕方」と「対象のファイル(群)」
    - “だれ+属性” ... 属性を追加したいとき
    - “だれ-属性” ... 属性を削除したいとき
    - 「だれ」部分はugoの組み合わせ(a = allでugoと同義)、「属性」部分はrwxの組み合わせが可能です。
      以下の”a+x”は全員(ugo)にx(実行許可)を追加する例です

```
$ ls -l /home/admin/www.py
-rw-r--r-- 1 admin admin 6370 Oct 18 16:02 /home/admin/www.py
$ chmod a+x /home/admin/www.py
$ ls -l /home/admin/www.py
-rwxr-xr-x 1 admin admin 6370 Oct 18 16:02 /home/admin/www.py
$ sudo /home/admin/www.py
  ... snip ...
(debug) serving at port 80
```

クラウドコンピューティング
第05回 ネットワークを制御する権限

3年、秋学期、選択; 旧「オペレーティングシステム」 ;脚注はELや定期試験の範囲ではありません

前回までのあらすじ++

- マルチユーザなので、ユーザを区別する必要があります
- ユーザには権限の違いがあります
  - 特に一般ユーザと管理者ユーザ(root)の相違は重要です
- ユーザの権限は、ユーザが起動した
  - (a)プロセスにも
  - **(b)プロセスが利用する機能(例:ネットワーク)にも影響を与えます**
    つまり、ユーザの権限がプロセスに紐づいているわけです

ユーザとネットワークの制限

- 一般ユーザ admin
  - 普通に利用する(例:ブラウザを使うなどの)場合、制限はありません
    正確には、**rootが設定した制限の中で無制限**と言うべきですね
  - お試しのサーバ(ポート番号が1024以上)を起動することは可能(例: 8080/tcpで試験)
- 管理者ユーザ root
  - 主要サービスのサーバ(ポート番号が1024未満)の起動ができます 【演習に関わる重要ポイント】
    WWWサーバ、DNSサーバ、メールサーバなど重要なサーバは全て、このカテゴリです
  - OSの各種設定変更や制限の追加・削除ができます
    - IPアドレスやホスト名などの基本設定
    - フィルタ(利用方法の制限)や流量の制限が出来ます

(脚注) 【復習】 ポート番号とは、春学期のネットワークで取り上げたTCPやUDPのポート番号のことです。 アプリケーション(正確にはプロトコル)を区別する識別番号。 例: WWWサーバ(HTTP)のポート番号は80、DNSは53、SSHは22

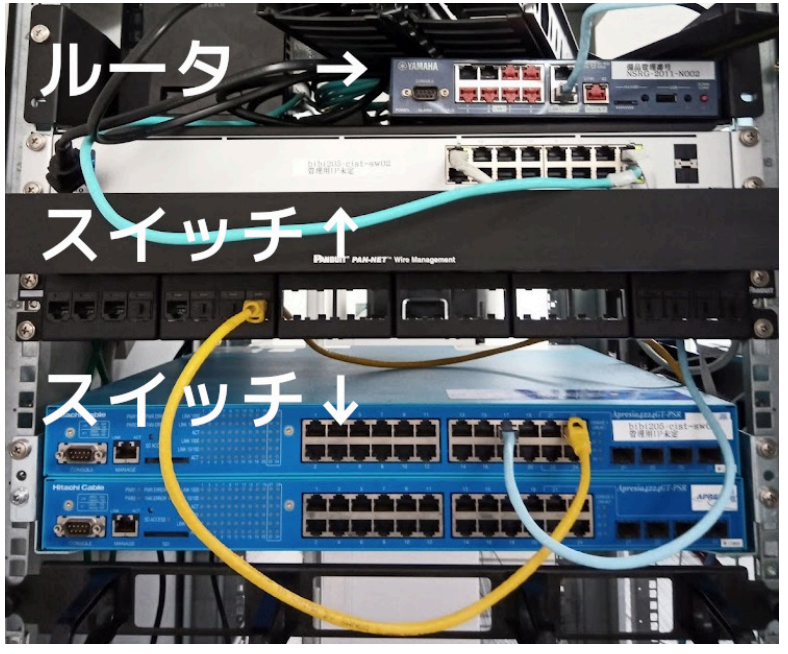
事例: www.py

- www.pyはWWWサーバなので、80/tcp(TCPのポート番号80)でリクエストを待ち受けようとします
- root権限が無いと、www.pyのプロセスは80/tcpを利用することができません
  - 一般に、業界人は、これを「www.pyが80/tcpを開けない」などと表現しています
- そういうわけでsudoをつけてwww.pyを起動する必要があるわけです

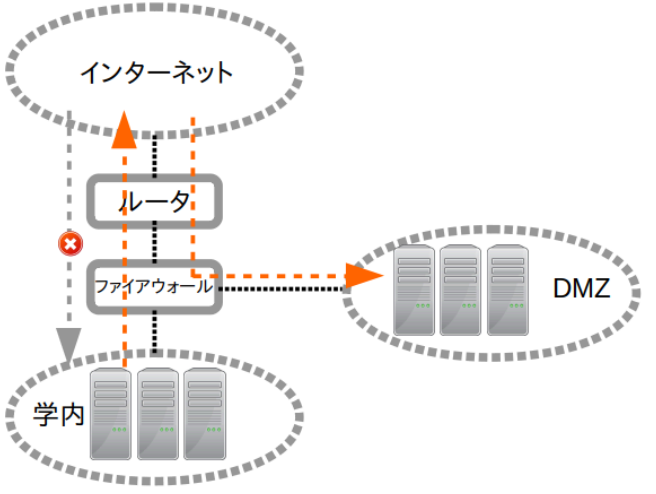
```
$ curl -O http://api.fml.org/dist/www.py
$ sudo python3 /home/admin/www.py

// chmod a+x すればpython3は不要(第04回の内容を踏まえた例)
$ chmod a+x /home/admin/www.py
$ sudo /home/admin/www.py
```

(脚注) Unixでは全デバイスをファイルと抽象化して考えています。 だから80/tcpというネットワークを待ち受ける何か(専門用語ではソケット)もファイルの一種なので、 「80/tcpを開く」という慣用表現があるのだと思いますね

【ネットワークの復習】ネットワーク機材とフィルタ



ポート(ケーブルを挿す口)ごとにフィルタが設定できますが、 そうそう、そんな細かい設定はしません



実際のネットワーク図は、こんな感じです。 通信の許可・不許可は、矢印のように大味な制御だけをしています

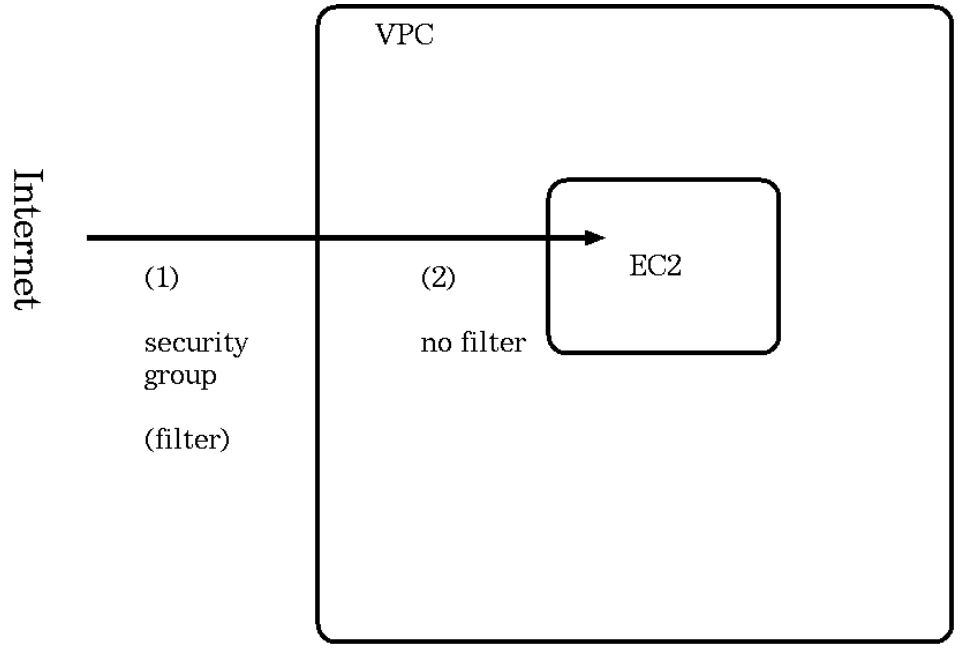
(脚注) いずれの図も春学期の科目「コンピュータネットワーク」より

AWSとUnixのフィルタの区別(1)

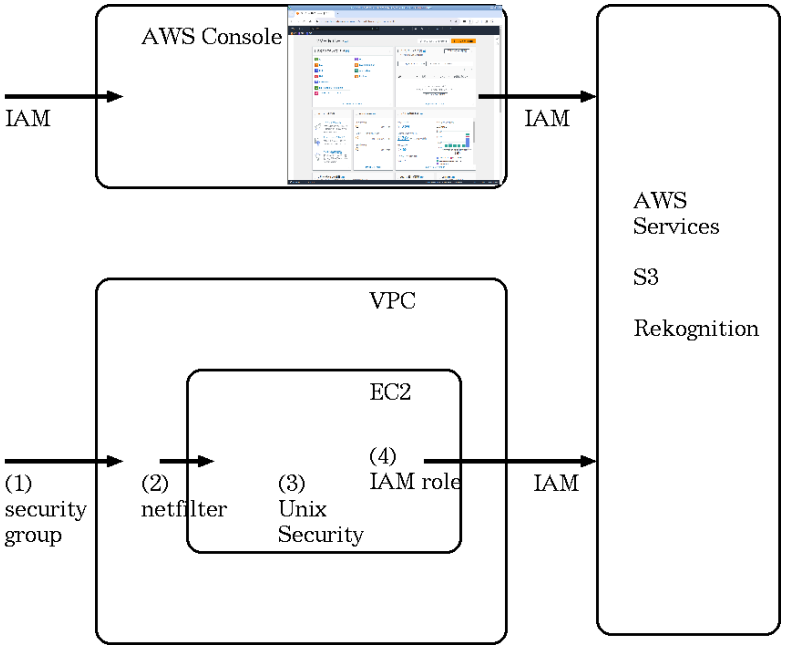
- **EC2**とは、AWSが提供する(仮想の)PCハードウエア
  - EC2(という仮想PC)にOSをインストールすることが出来ます
    つまりEC2上でOSを動かすことが出来ます
  - たいていはAWS提供のインストール済みイメージ(AMI = AWS Machine Image)を利用してシステムを構築します。 本科目で使っているイメージはAWS提供のDebian GNU/Linuxイメージ
- EC2構築時にsecurity groupを選択しますが、これはAWSの機能(!= Unixの機能)
  - UnixとAWSの機能を区別することは大事です
    - この科目で解説しているのは(EC2の上で動かす)Unixの機能です
    - security groupはEC2外のどこかで動作しているフィルタです
      つまりsecurity groupはUnixと無関係です

(脚注) VPSをレンタルするなどして自分でサーバを立てる場合は、 Unixのフィルタを設定してください

AWSとUnixのフィルタの区別(2)



(2)がUnixのフィルタです。 フィルタは未設定(つまり全通信は素どおしというザル設定)がEC2 default



全体構成図: 本科目でも、そのうちEC2からAWSのサービスを使いますが、 ここ(図の(4))には初めから制限がかかっています。 IAMはAWSの権限設定です

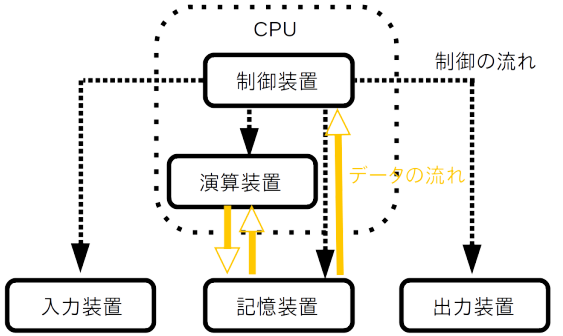
(脚注) この図(春学期の再掲)はオンプレミスぽい書き方をしています。 security groupの説明(場所)が正しいのか？は不明です。 AWSは正確な技術的説明をしてくれないので真偽が判断出来ません。 まぁ、こんな感じだろうという図(イメージ)になってます

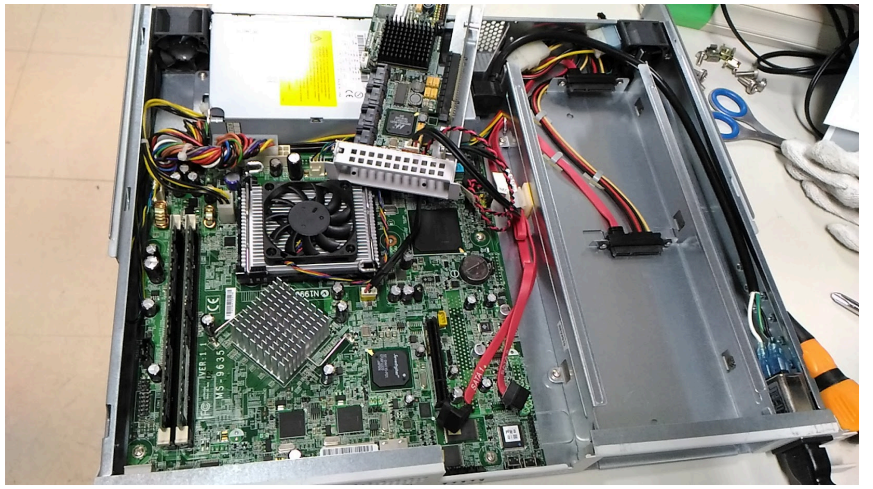
クラウドコンピューティング
第06回 コンピュータの構成要素と階層概念

3年、秋学期、選択; 旧「オペレーティングシステム」; ;脚注はELや定期試験の範囲ではありません

コンピュータの構成要素【ほぼ復習】

- コンピュータは多くの要素(部品)が組み合わさって動いています
  - CPU、メモリ、ストレージ(HDD,SSD)、ネットワーク、時計、...
- 部品を共用しているとも言えます
  - たとえば、一つのOSを複数のユーザで利用しているので、メモリも共有しています
- 右図も参照
  - 右図(上)は「五大装置」(基本情報処理試験の出題範囲、基本中の基本)
  - 右図(下)はサーバ機の蓋を開けた様子

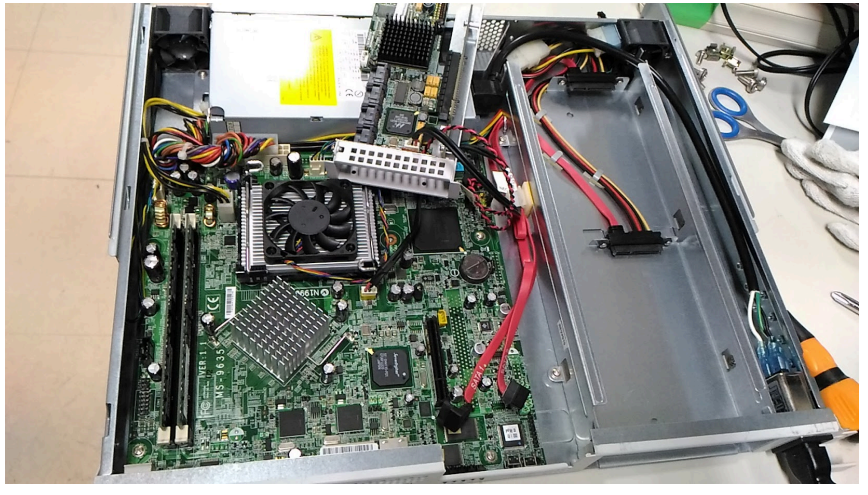




(脚注) 復習 <- たいていは「情報〜概論」とか「コンピュータアーキテクチャ」といったタイトルの科目で取り上げているはず

デバイスドライバ

- 部品を挿せば動くわけではありません
- 部品ひとつひとつが小さなコンピュータで、OSから、その部品を読み書きする必要があります
- そのため、OSの中には各デバイスごとの専用プログラムが搭載されています。これをデバイスドライバと呼んでいます



(脚注1) すぐに4文字略称を考える日本人なので、業界人は「デバドラ」と発音しています

(脚注2) モノによってはOSが起動した後にデバイスドライバを追加することも出来ますが、 基本は起動前に、つまり最初からOSの(配布物の)中にデバイスドライバが含まれている必要があります。 対応するストレージやネットワークなどのデバドラがないとOSが起動すらできません。 なお、新しいハードウエアが登場したら、当然そのデバイスドライバを開発する必要があります

仮想記憶システム(Virtual Memory)の必要性

- TSSの特徴はマルチユーザつまり一つのOSを同時に複数のユーザが使えることです。 とうぜん複数のプロセスが同時に動いています
- 同時に複数のプロセスが動くので、メモリ上にも同時に複数のプロセスがロードされています (右図はイメージ)

Physical memory (real memory)

1	Process B
2	Process A
3 4	Process Z
5	Process B
6 7 8	Process D
9	Process A
10	Process C
11 12 13 14	
15	Process E

(脚注) 本科目で「仮想記憶の動作原理」は扱いません。それは大学院の授業いき

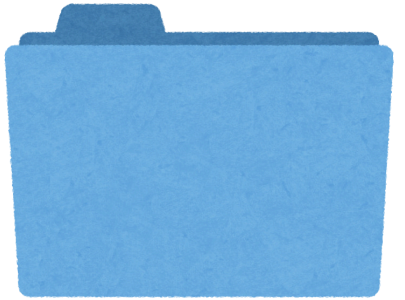
仮想記憶システム(Virtual Memory)の目的

- VMの目的はおもに2つあります
    1. 同時に複数のプロセスがメモリを利用できること
    2. 物理メモリよりも巨大な仮想メモリ空間が利用できること

(脚注1) 本科目で「仮想記憶の動作原理」は扱いません。それは大学院の授業いき
(脚注2) もともと必要だったのは1.で、これは今でも重要な機能です。 一方2.は、(むりやりにでも)巨大なメモリを利用したいという需要が生んだ機能です (例:人工知能(シンボリックAI)のためのLISPマシンとして使いたい)。 ただし、2.には無理があるので猛烈にプロセスの動作が遅くなります。 この使い方は推奨できません。 昔はメモリが小さかったので仕方ないのですが、 昨今のメモリは安価なので、 無理なことは考えず、金で解決(札束を用意してメモリを購入:-)するべきです
(脚注3) もうすこし詳細な説明をすると、 1.の機能は「プログラマがメモリ空間の使い方を気にせずプログラムを書ける」ために必要です。 本来は(物理)メモリのどこに何を配置してといったことを詳細に考えなければプログラムは動作しません。 そして、OSで動くプロセスが一つであれば、うまく動くかもしれませんが、 複数プロセスを動かしたら互いに衝突してしまいます。 だからといって、いちいちプログラマ間でメモリの使い方を相談しあうわけにもいきません。 VMの機能1.は、これを解決してくれます

階層という概念

- 分類 = 階層
- 身のまわりでは本棚が代表例ですが、階層は1階層です
  - 「情報関係の書物」「数学」の本棚のように内容の種類で置いてある場所が異なります
  - 本棚は複数階層ではなく「1階層でグループ分けされている」と考えられます
- コンピュータの部品(物理)も概念(論理)も細かく分類して考えていけます



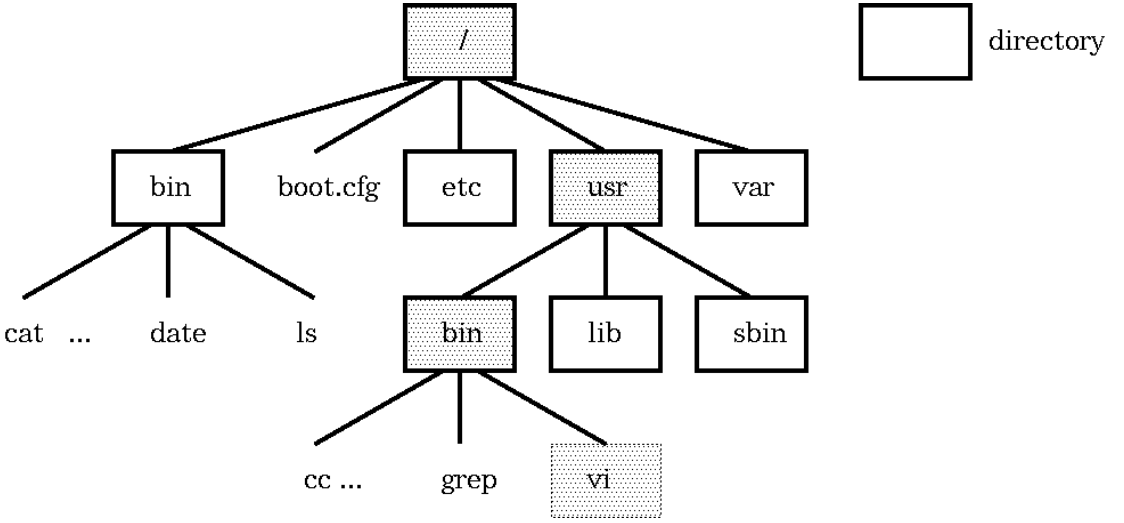
(脚注1) 分類に関係する表現 ... クラス分け(classify)、カテゴライズ(categorize)、ソートする(sort)、グループ分け(grouping)

(脚注2) たとえば、Unixの場合、ファイルもプロセス群も階層構造になっています。このあと演習で確認します

(脚注3) そもそも、分類することが学問とも言えます。 かつては学問=哲学でした。 いまの学問の分類はアリストテレスに由来しており、 極論「学問する」は「アリストテレスのように考えること」です。 ただし「論理的定義=階層型分類」は師匠のプラトン由来

階層型ファイルシステム

- ファイルは本、ファイルを格納するフォルダ(ディレクトリ)は一つの本棚と考えられます
- コンピュータは論理空間なので、この本棚も階層構造に出来ます
- 本棚(/)の中にはファイルも置けるし、より小さな本棚(usr)も作成できます。 さらに本棚(usr)の中に本棚(bin)を作り、その中に本(vi)を置けます。 これがviの実体(/usr/bin/vi)。 この仕組みを(階層型)ファイルシステムと呼んでいます(図上)
  - 本棚は1階層しかない例(図の下左)
  - マトリョーシカのイメージ(図の下右)



(脚注1) 次回以降、ファイルシステムをもう少し詳しく取り上げます (脚注2) 論理的にはファイルシステムの無限の階層化が可能です

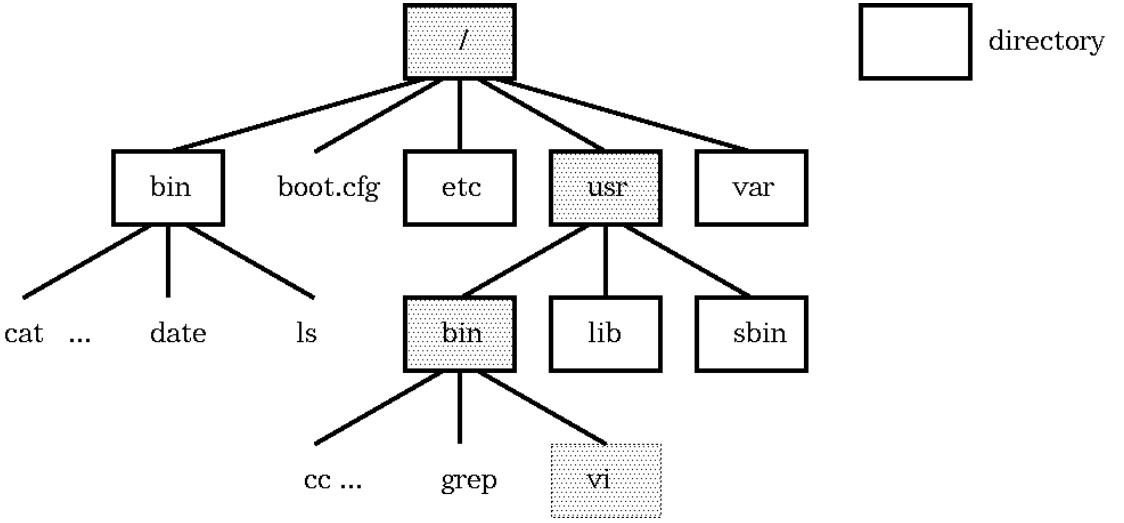
クラウドコンピューティング
第07回 ファイルシステム(ローカル)

3年、秋学期、選択; 旧「オペレーティングシステム」; ;脚注はELや定期試験の範囲ではありません

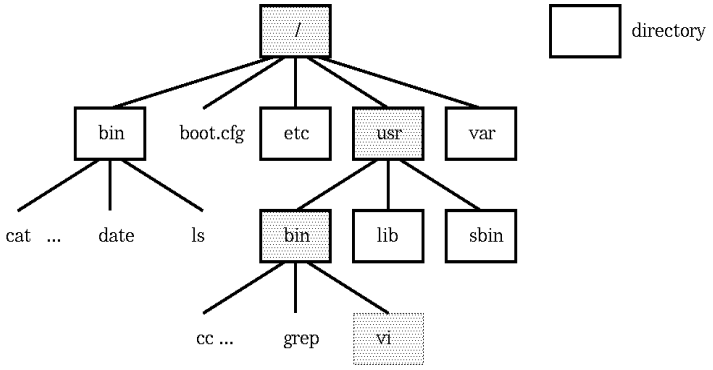
【復習】階層型ファイルシステム

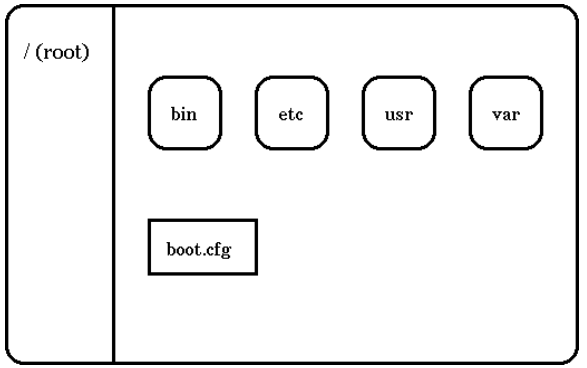
- Unixファイルシステムは右図(上)のような木構造
  - 四角がディレクトリ(フォルダ)で本棚に相当
  - ただし本棚の中には、 本も置けるし、さらに本棚も作れるところが論理空間の利点(現実の物理空間とは異なるところ)

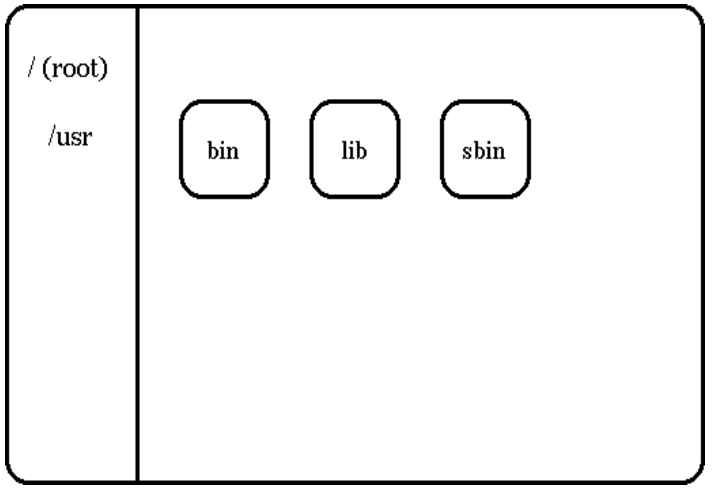
- Windowsのフォルダっぽく表現すると？
  -> 次ページ



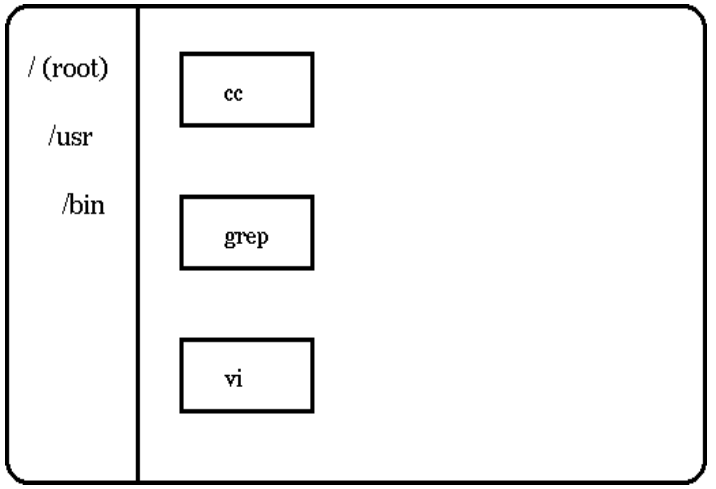
各階層をWindowsのフォルダ表示に似た図解をしてみる







左図のusrフォルダをクリックして/usrの階層が表示された様子。bin (プログラム)、 lib (ライブラリ)、 sbin (管理用プログラム) といった役割ごとに分かれています。これは左の/も同様



さらにbinフォルダをクリックすると/usr/binの階層が表示されます。 ここにはプログラム(コンパイラのcc, エディタのviなど)群が並んでいます

(脚注) binはbinary、varはvarietyの略と言われていたはず。 etcはエトセトラの略で、うまく分類できないモノすべてという意味。 etcとvarは、どちらも雑多な意味ですが、 (あまり変わらない)設定ファイルはetc、ログのような可変のものはvar以下に置きます

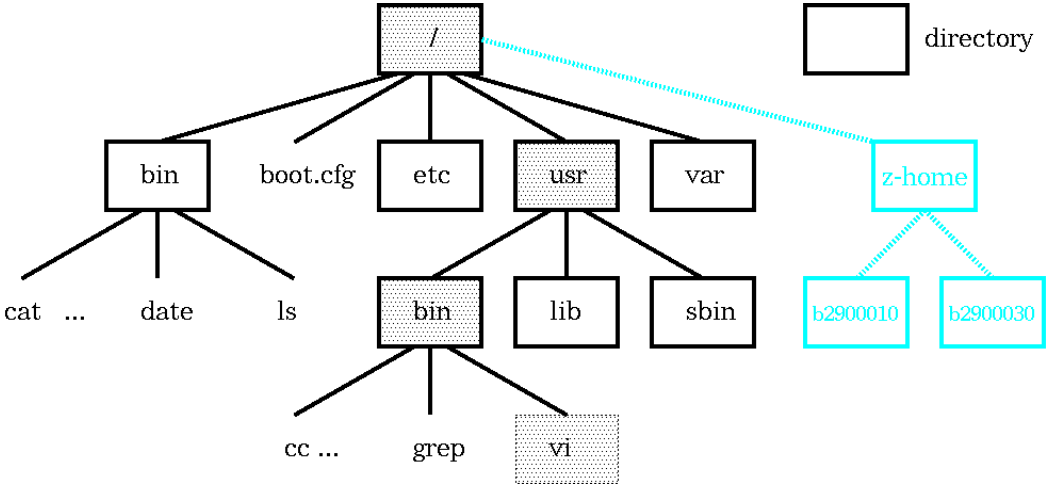
Unixファイルシステム第一階層の分類

名前	用途	由来	備考
bin	一般ユーザが利用するプログラム	binary	
dev	デバイスファイル群	device	デバイスも特殊ファイルとして実装
			(Unix特有の話)
etc	さまざまな(設定)ファイル	et cetera	
home	ホームディレクトリ群	home	
sbin	管理者が利用するプログラム	system binary	
var	さまざまな(ログ)ファイル	variety	
usr	ユーザ用プログラム	usr	(脚注)

(脚注1) 図には、たくさん書けないので、表示内容を厳選しているだけです (脚注2) /binと/sbinがあり、別途/usr/binと/usr/sbinという似たようなディレクトリがあり、 それらを使い分けていることには歴史的事情があるのですが、 今となっては分かりにくい例です。 最近の使い方では、分けることに意味が無いので、 最新のDebianでは/binと/usr/bin、/sbinと/usr/sbinは同じものとなっています (シンボリックリンクという仕組みで同じものにしています、 シンボリックリンクはWindowsの.lnkと同じと考えてOKです)

Unixファイルシステムの特徴として「ディスクを接ぎ木」できる

- あらたにz-homeというディレクトリを作成し、巨大なストレージを接ぎ木した様子
    - z-homeの下の階層には学籍番号のディレクトリ群が並んでいるという想定
- 接ぎ木するストレージは、ローカル接続(PCの中に装填し、専用の線でつなぐ場合)もあるし、 ネットワーク的に離れた場所のストレージを接ぎ木する場合もあります。 どちらも可能

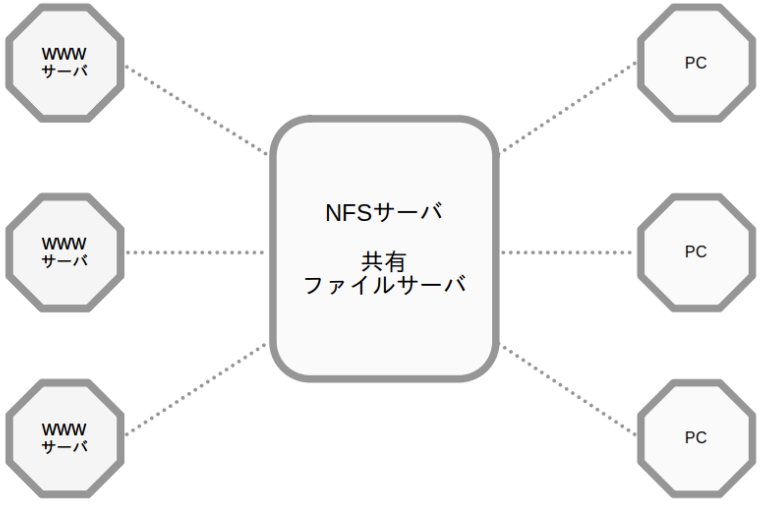


(脚注1) 本当は任意の場所に接ぎ木できます。 たとえば、adminユーザ専用の巨大な外部ストレージを用意し、 それを/home/admin/dataに接ぎ木(Unix用語ではmount)することが出来ます。 ただPCもSSDもHDDも安価な昨今では、そういう使い方をしないので、あまりピンと来ないでしょう

(脚注2) きっとUnixカーネルの内部実装の話をしないと、 ローカルな接続と、ネットワーク的な接続の相違については、よくわからないでしょう。 ただ同じように扱えるように作り込んであるということだけ「へ〜」と思ってくれればよいです

例: AWS EFS (NFSサーバ)を使ったファイル共有

- WWWサーバ群が共通で使うファイル(たとえば商品画像)を共有する
- PC教室: どのPCからでもホームディレクトリにアクセスできる
  - PC教室のように、使うたびに異なるPCを使う場合でも、 ホームディレクトリに保存したファイルは使いたい。 だからPCローカルに保存してはダメ。 ホームディレクトリは共用ファイルサーバに保存する
  - これがPC教室のZドライブのしくみ

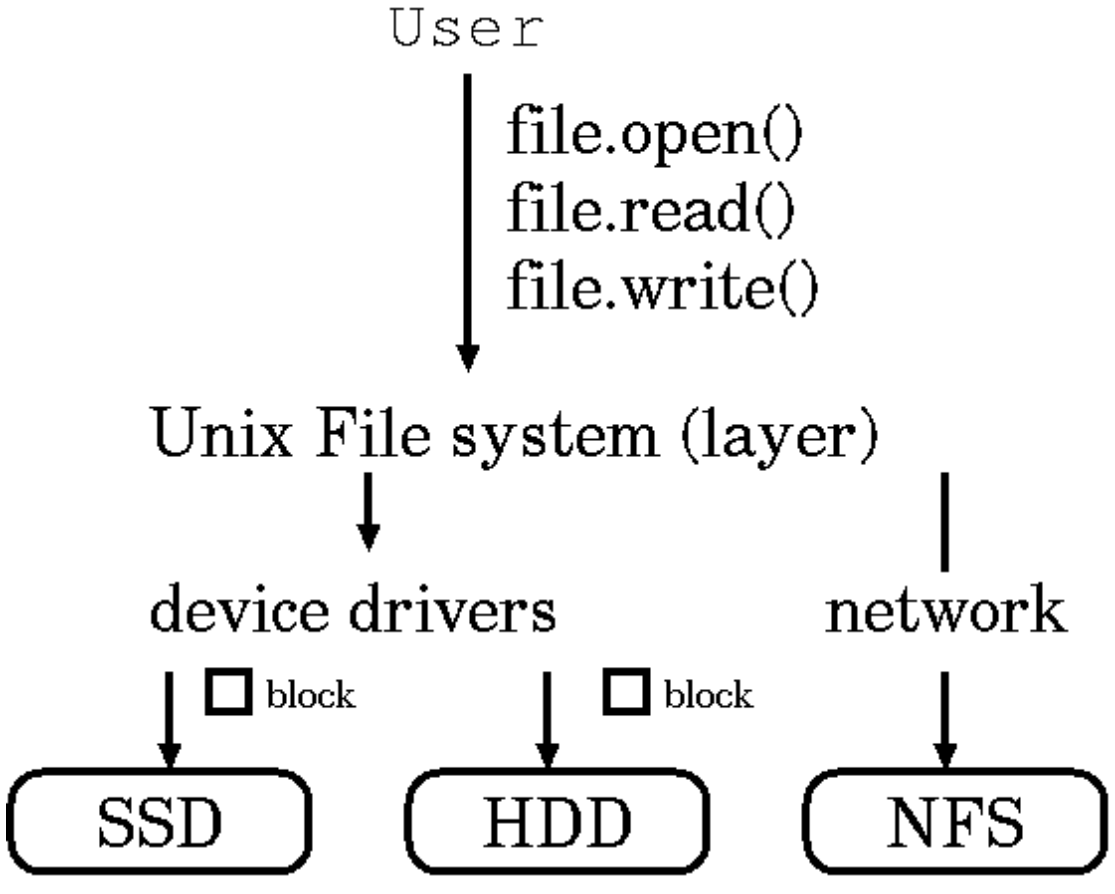


クラウドコンピューティング
第08回 ファイルシステム(クラウドストレージ)

3年、秋学期、選択; 旧「オペレーティングシステム」; ;脚注はELや定期試験の範囲ではありません

Unixファイルシステム 【(半分は)復習】

- Unix ファイルシステム
  - [復習] ディスク(ストレージ)を接ぎ木して階層構造を作ることができます。 接ぎ木できるディスクとは、SSDやHDD、NFSごしにアクセスできるストレージ
  - **ファイルとして操作できるように見せかけるのはファイルシステムという階層(layer)**のお仕事
- ファイルシステムの裏側、つまりSSDやHDDへの読み書きは
  - **ブロック(512バイト)単位で行います**(最近のディスクは、もう少し単位が大きいのが普通)
  - **ブロックストレージ**と呼んでいます
  - [復習] 実際に読み書きする担当がデバイスドライバ



(脚注1) 細かい話はAWS の「ストレージ」のドキュメントを参照

https://aws.amazon.com/jp/products/storage/

(脚注2) ファイルの中身の切れ目などは考えず、機械的に512バイトずつに切って読み書きしています。 つまり1GBファイルなら約200万ブロックに分解して読み書きするわけです。 ちなみにNFSは途中(ネットワーク部分)が異なるのでブロックとは別枠の扱いです

クラウドストレージサービス(オブジェクトストレージ)の特徴

- ファイル(=オブジェクト)単位の読み書きを行う
- たいていはHTTPSでアクセスしてファイルを読み書きする

  1. 各アプリケーションから
  2. HTTPSを使い (<- ファイルシステムへの読み書きではないことに注意)
  3. アップロード(書きこみ)とダウンロード(読みこみ)をします

- とにかく、この2つは既存のファイルシステムとは大きく異なります(その他の特徴は省略)

(脚注) その他の細かい話はAWS の「ストレージ」のドキュメントを参照 -> https://aws.amazon.com/jp/products/storage/

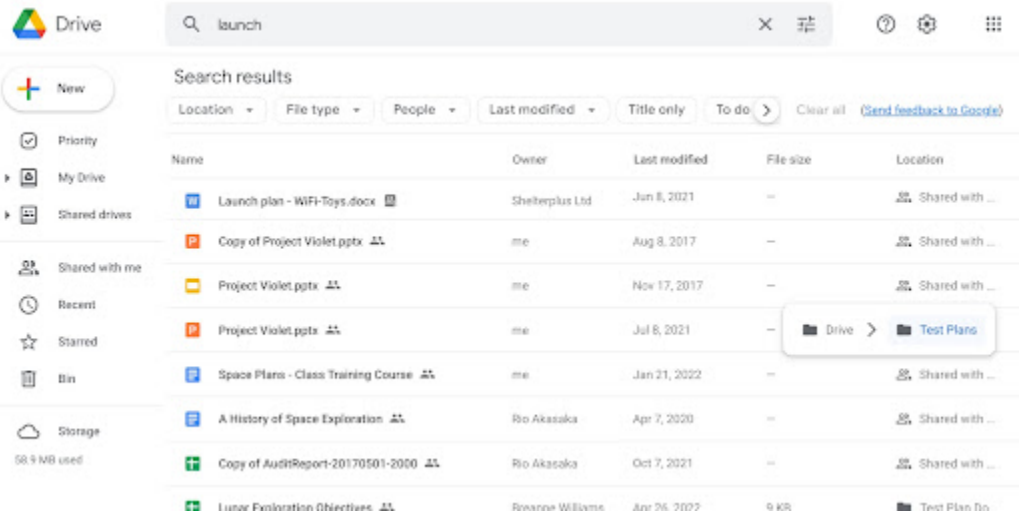
クラウドストレージサービスの例

- Google Drive
- Microsoft Azure OneDrive
- Apple iCloud
- AWS S3

(脚注) 紹介だけです。 実際に使っているものも多いので大丈夫でしょう。 AWS S3は、このあと演習で扱います

Google Drive

- Google Driveをブラウザで開くとフォルダのような画面が現れますが、 実際にそういう構造をしているわけではありません
    - 裏側がどうであれ、それ(フォルダ)らしいユーザインターフェイスを作りこんだ良い例
- 実際に読み書きする際に利用するプロトコルはHTTPSです



https://workspaceupdates-ja.googleblog.com/2022/06/google_10.html

AWS S3

- HTTPS プロトコルで読み書きするオブジェクトストレージサービス
- スケーラビリティ
  - 1オブジェクト(ファイル)あたりの上限は存在していて、5TB
  - 格納可能な総量は無限大(オブジェクトの総数は無制限)
- 99.999999999%(eleven 9)のデータ耐久性
- セキュリティ
  - IAMを適切に設定することで柔軟なアクセス制御が可能
- 参考文献
  - (公式) Amazon S3 の特徴

(脚注) https://aws.amazon.com/jp/s3/storage-classes/

「S3 はデフォルトで最低 3 つのアベイラビリティーゾーンにデータを冗長的に保存」しています。

AWS S3 応用編

- 社内の書類もAWSにバックアップしましょう
  - あんがい低価格、ストレージの容量に上限なし
- 災害対策に使いましょう
  - 高可用性、世界の各地にあるAWSリージョンをうまく使うのがポイント
  - 例: 東京本社の重要データを、大阪やアメリカのS3にバックアップする
- 簡易(静的)ウエブサーバとして”も”使えます
  - これはオブジェクトストレージの特徴ではなくAWS S3固有の機能です

(脚注1) Q: 社内の設備のデメリットとは? A: 巨大なデータは磁気テープに保存しますが書くのも読むのも色々めんどうなのです。 (情シスとしては)そういったチマチマした運用サポートから開放されて嬉しいですね。 もちん運用上のセキュリティを事前に考えておくことは話の大前提です。 なお、そこまでデータが大きくないとS3のメリットは微妙かもしれません

(脚注2) 最近は、災害対策のことBCP(Business Continuity Plan;事業継続計画)と呼んでいます

(脚注3) 静的＝コンテンツを一方的に配信するだけですが、 いまどきはJavascriptを配信して、ブラウザだけで色々できるため、 ショッピングカートの購入ボタン(= DBMS への書きこみが発生する時)以外は、 静的コンテンツ配信でも大抵のサービスが作れそうですよ?