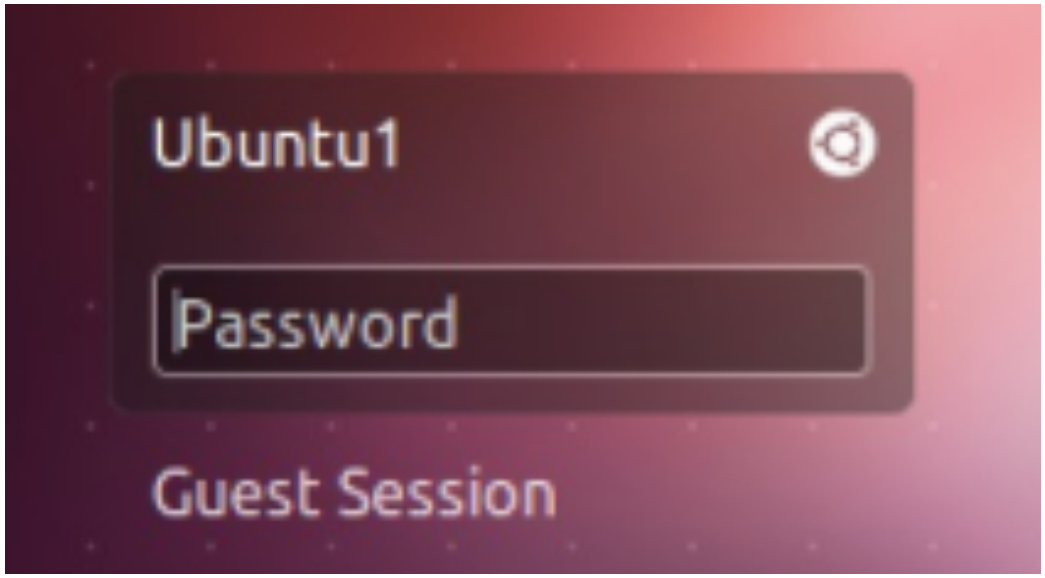
プログラミング応用

第4週

本日の講義

- OSの役割
- OSの歴史
- OSが動作する仕組み
  - カーネル
  - システムコール
  - プロセス/プロセス間通信
- C言語からカーネルを呼び出す
- 演習

例）ログイン用ソフトウェア(1/2)



ログイン用ソフトウェアの基本動作：

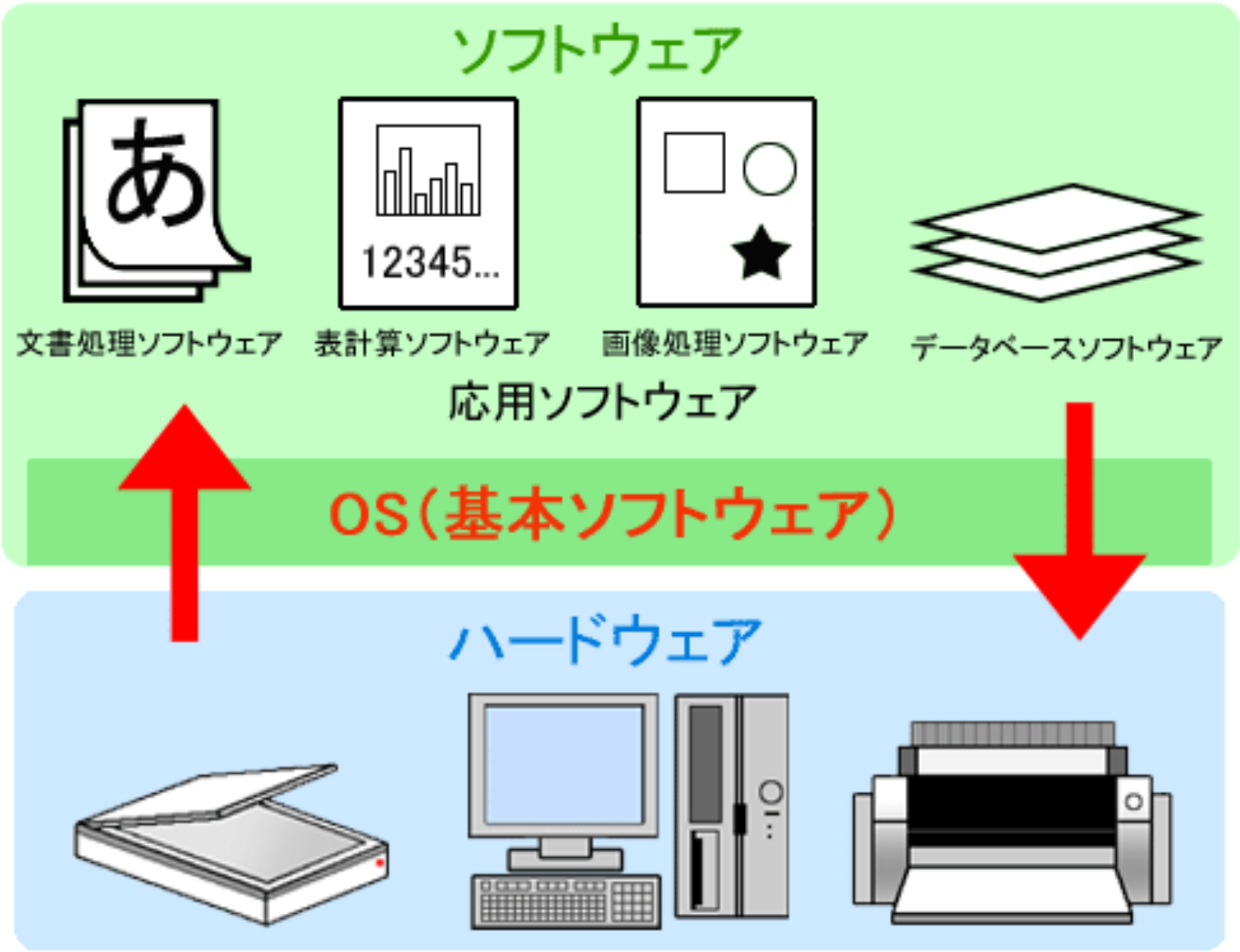
1. ログイン用IDを入力
2. パスワードを入力
3. 正しければデスクトップにジャンプさせる

では、どうやって作る？

例）ログイン用ソフトウェア(2/2)

- ログイン用ソフトウェアのプログラミング
  - キーボードから文字を受け取る機能
  - ディスプレイに受け取った文字を表示する機能
- これらの開発ではハードウェアを制御する命令を記述するは必要ない
  - キーボードの入力信号を検出したり、ディスプレイのどのドットに何色を表示するかといった具体的なハードウェア制御は考慮しなくてもソフトが作成可能
  - OSがハードウェアを制御

ハードウェアとソフトウェア



OSの代表的な役割

- ハードウェアとソフトウェアの橋渡し役
  - 計算資源の管理（＝プロセス管理）
    - 限られたCPUの計算能力を効率良く使えるように、プログラム実行の優先順位を付ける
  - メモリ資源の管理（＝空間管理）
    - 限られたメモリ資源を有効に使えるよう、プログラム実行に割り当てるメモリを調整
  - ファイルシステムの提供
    - 0と1で記録されたメモリ上の情報は人間には解読が困難
    - 人間に分かりやすいファイル/ディレクトリといった概念で表現

OSの歴史（OS誕生以前）

- 1640–50年代：
  アプリケーションソフトウェアにハードウェアを制御する命令を記述
  - 機械語を0と1で記述
  - 開発者はソフトウェアもハードウェアの制御も詳細に把握する必要があった
- 1960年中頃にIBM System/360にOSの機能が確立し、複数のプログラムが同時実行可能に


IBM

OSの歴史（OSの登場後）

- 1970年代：複数の利用者が同時利用
- 1980年以降：
  GUI（グラフィカルユーザインターフェース）の誕生
  - 個人用コンピュータとしてはWindows(1985-)/MacOS(1984-)がシェアを独占
  - 開発者はLinuxなどUNIXから派生したOSを利用することが多い
- 2000年代以降
  - モバイル端末向けの軽量化されたUNIX系OSが誕生
    iOS/Android

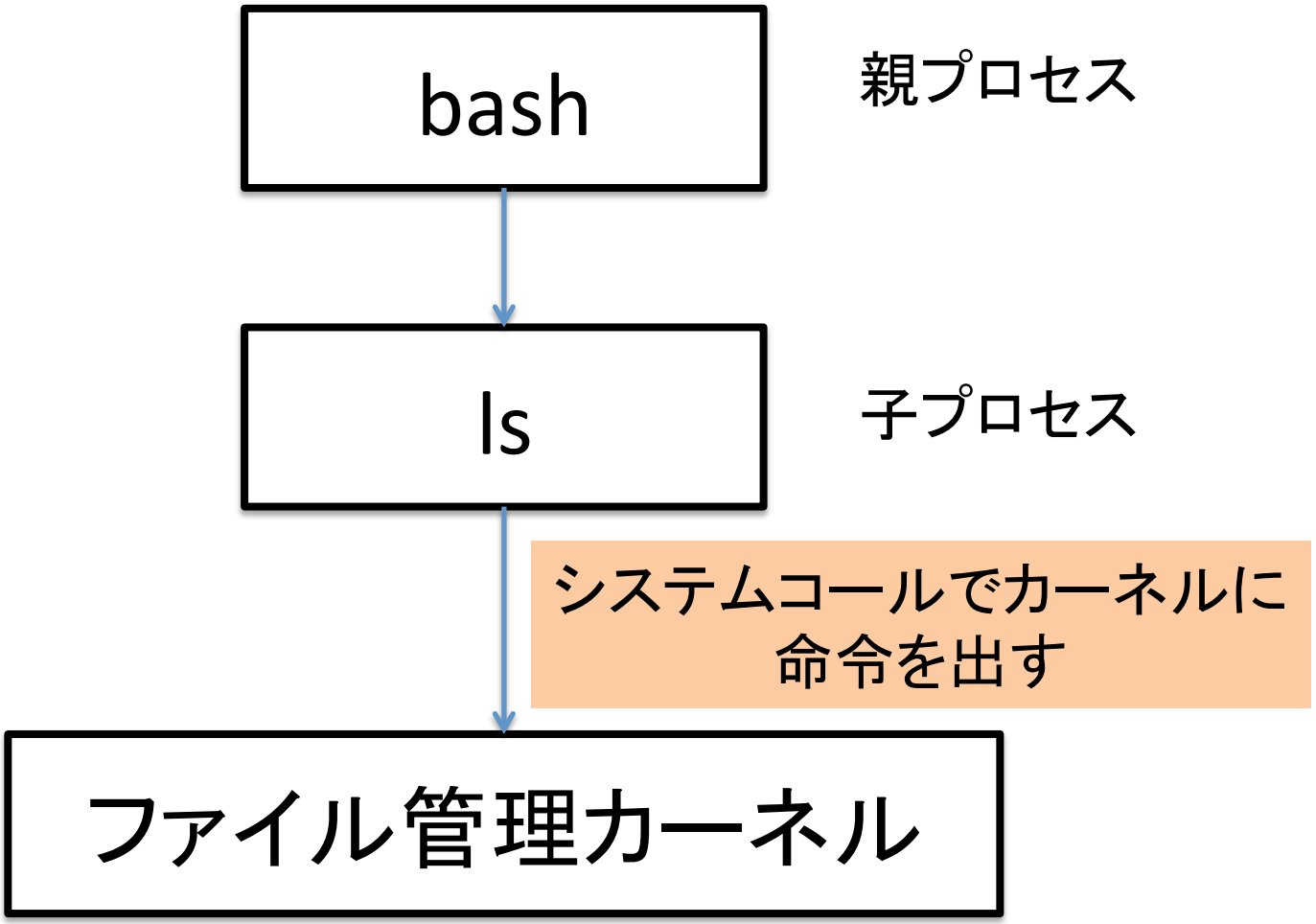
OSの内部構造

- OS　＝　カーネルの集合体
  - 各々のカーネルはOSの基本機能を提供
  - カーネルの提供する機能
    プロセス管理、空間管理、ファイル管理、
    割り込み制御、入出力制御、時間管理など
- アプリケーションソフトウェアは「システムコール」を使ってカーネルの機能を呼び出す
  - アプリケーションソフトウェアは基本的に
    ハードウェアを制御する命令を出すことは
    出来ない

プロセス

- プロセス = プログラムの実行単位
- 例）「端末」でlsコマンドを入力した場合
  1. 「端末」を起動するとbashプロセスを起動
  2. lsコマンドを入力するとlsプロセスが起動
  3. lsプロセスはファイル管理カーネルに命令を出す
  4. ファイル管理カーネルが画面に表示
  5. lsプロセスが終了

子プロセス/親プロセス



プロセス管理

- 複数のプロセスを同時に実行する仕組み
  - CPUの計算資源は有限
    （同時実行可能なプロセス数 = CPUのスレッド数）
  - OSは複数のプログラムが効率的に実行されるよう実行するプロセスの優先順位を決定
  - ユーザがアプリケーションを起動すると1つ以上のプロセスが生成

- topコマンドやpsコマンドで実行中のプロセスを確認できる

空間管理

- メモリ資源を管理する仕組み
  - メモリ資源は有限
  - どのプログラムにどの程度のメモリを割り当てるか決定する
  - 実際には”仮想メモリ(Virtual Memory)”を用意し割り当てる
  - プログラムが実際のメモリ資源よりも多くメモリを使った場合にはハードディスクなどにデータを退避させる（＝スワップメモリ）

- topコマンドやfreeコマンドでメモリの使用状況を確認できる

ファイル管理

- lsコマンドを入力するとファイル/ディレクトリが一覧される
  - ユーザはメモリのどこにファイルがあるか知らない
  - メモリに記録されている情報を“ファイル“という人間にも分かりやすい概念で表現

- mkdir, ls, cd, cp, mvなどのコマンドでOSの提供するファイル操作を行うことが出来る

OSの機能の呼び出し

1. シェル（端末）から呼び出す
   - 本日の演習（UNIXコマンドを用いたプロセス管理）
2. アプリケーション・プログラムから呼び出す
   → C言語などからシステムコールを呼ぶ
   - 次回、次々回の内容
     （複数のUNIXコマンドを組み合わせて使う、
     C言語からカーネルに命令を出す）

演習0

- 課題の作業用ディレクトリを作成し、ディレクトリ内に移動
  - $ cd c-programming-practice-(学生番号)
  - $ mkdir week4
  - $ cd week4

＊出来る人はGitでブランチを切ってから課題を始めてみましょう

演習1(OSの機能)

- 「OSの役割」と、「OSがなかったらどのような不都合が起こるか」、100文字程度でos.txtにまとめGitHubにpushしなさい。
  - $ gedit os.txt &
  - $ git add –A
  - $ git commit –m “第4週課題1提出”
  - $ git push origin master

演習2(プロセス管理1/3)

- topコマンドを実行しなさい
- topコマンドの実行画面の以下の項目の意味を調べ、top_command.txtというファイルにまとめなさい
  - PID
  - COMMAND
  - %CPU
  - MEM
  - STATE
- bashプロセスの現在の状況についてtop_command.txtに追記し、GitHubに提出

演習3（プロセス管理(2/3)）

- pstreeコマンドの使い方を調べ、bashプロセスの親子関係を表示しなさい
- pstreeコマンドの出力をpstree.txtにコピーし、どのような意味であるか考察を追記し、GitHubに提出

演習4(プロセス管理3/3)

- killコマンドの使い方を調べ、bashという名前のプロセスを終了しなさい
- killコマンドの使い方（指定したプロセスを終了する方法）をkill_command.txtにまとめGitHubに提出

演習5（ファイル管理）

- UNIXコマンド（cd, ls, mkdir, cp, mv）の使い方を改めて確認しなさい。
- 提出は不要です。

次回

- 次週は複数のUNIXコマンドをまとめるシェルスクリプトという言語や、C言語プログラムからカーネルを操作する方法を学びます。

- C言語が苦手な人はGW中に入門書を1冊やり切るつもりで復習しましょう