

円周率の計算

$\pi = 3.141592653589793.....$

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + (-1)^n \frac{1}{2n+1} \cdots$$

ライプニッツの式
(1400年頃)

$$\pi = \sqrt{12} \left(1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} - \frac{1}{7 \cdot 3^3} + \cdots \right)$$

マドハヴァの式
(1400年頃)

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \cdots$$

ニーラカンタの式 (1500年頃)

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots$$

オイラーの式 (1735)

桁落ち誤差 (丸め誤差、情報落ち)
有限の桁数のために生じる誤差

打切り誤差
無限の和を途中で打ち切ること
で生じる誤差

倍精度

real*8 :: pi=0.d0

integer :: i

ライプニッツ

```
do i=3999999, 1, -2
  pi=pi+2.d0/(2*i-1)/(2*i+1)
end do
write(*,*) pi*4
```

Fortran pi.f95

単精度

real :: pi=0.0

integer :: i

ライプニッツ

```
do i=1, 3999999, 2
  pi=pi+1.0/(2*i-1)-1.0/(2*i+1)
end do
write(*,*) pi*4
```

単精度

real :: pi=0.0

integer :: i

ライプニッツ

```
do i= 3999999, 1, -2
  pi=pi+1.0/(2*i-1)-1.0/(2*i+1)
end do
write(*,*) pi*4
```

倍精度

real*8 :: pi=0.d0

integer :: i

マドハヴァ

```
do i=24, 1, -1
  pi=pi+(-1)**(i-1)/(2.d0*i-1)&
    /3.d0**(i-1)
end do
write(*,*) sqrt(12.d0)*pi
```

C 言語 と Fortran の構文比較

入出力

C 言語

必要なヘッダファイル `stdio.h`

```
printf(“書式”, 出力変数)
scanf(“書式”, 入力変数)
```

出力 : モニタ
入力 : キーボード

書式指定子

`%n s` : 文字列 n 桁
`%n d` : 整数型 n 桁
`%n ld` : 倍精度整数型 n 桁

`%n .m f` : 固定小数点型、幅 n 桁、小数点以下 m 桁
`%n .m lf` : 倍精度固定小数点型、幅 n 桁、小数点以下 m 桁

`%n .m e` : 指数表示、幅 n 桁、小数点以下 m 桁
`%n .m le` : 倍精度指数表示、幅 n 桁、小数点以下 m 桁

エスケープシーケンス

<code>¥n</code> : 改行コード	<code>¥t</code> : tab コード	<code>¥'</code> : 文字 '
<code>¥a</code> : ベル	<code>¥¥</code> : 文字 ¥	<code>¥"</code> : 文字 "

Fortran

```
write(装置番号, '(書式)') 出力変数
read(装置番号, '(書式)') 入力変数
```

出力 : モニタ (装置番号は 6 あるいは *)
入力 : キーボード (装置番号は 5 あるいは *)

書式指定子

`An` : 文字型 n 桁
`In` : 整数型 n 桁
`F n.m` : 固定小数点型、幅 n 桁、小数点以下 m 桁
`E n.m` : 指数表示、幅 n 桁、小数点以下 m 桁

書式指定子が複数並ぶときはカンマ(,)で区切る

例 `write(*, '(F10.6,F8.4)') pi, r`

オペレーティングシステム

OS (Operating System) の役割：ハードウェアの抽象化、リソースの管理

UNIX, Linux, Windows 等

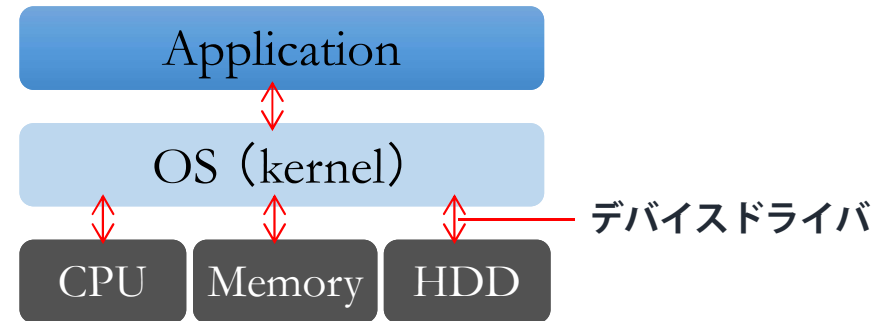
OS カーネルは全プログラム実行
と
ハードウェアを制御

- プロセス（プログラム実行）管理
プロセスの生成・実行・消滅を管理

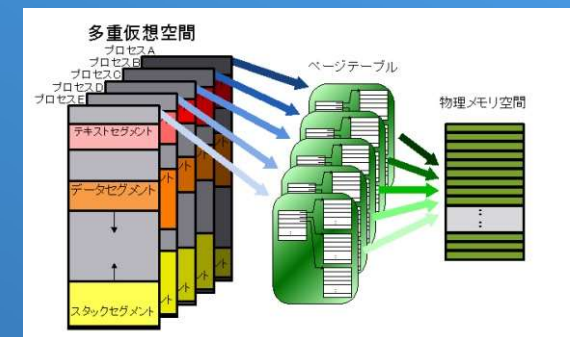
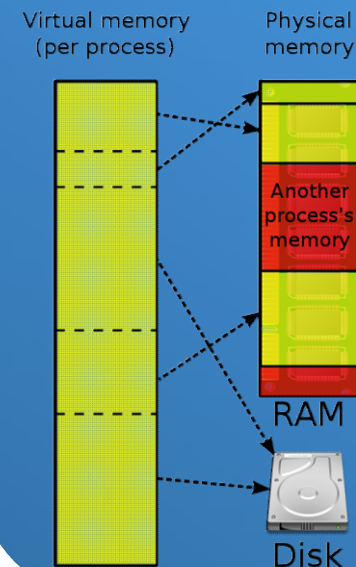
• メモリ管理

キャッシュー DRAM – HDD

に対する割り当て、解放、ス
ワップ等を行う



メモリ管理（仮想記憶）



主なオペレーティングシステム

Unix 1960年代～

マルチタスク、マルチユーザに対応

C言語と共にAT&T(ベル研)で開発
政府や教育機関に配布

カーネル(OS)のソースコード、
開発環境を含む

cc - C言語コンパイラ
as - アセンブラ
ld - リンカ
lib - ライブラリ

Linux 1991年～

UNIX互換OS

無料、オープンソース
(改良は自由だが公開)

MS-DOS 1981年～1985年

16bit PC の Disk Operating System
シングルタスク、シングルユーザ

Microsoftが開発・販売

Windows 1985年～

MS-DOSのGUI(グラフィカル
ユーザ インタフェイス)

マルチタスクに対応

Windows 3.x 1990年～

Windows 95/98/Me
個人用 32bit 対応

Windows NT系 1993年～

ビジネス用

Windows 2000 2000年

Windows XP 2001年

Windows Vista 2006年 64bit有

Windows 7 2009年 64bit有

Windows 8 2012年 64bit有

Mac OS 1984年～2006年

GUIを基本としたOS

ひとつのソフトが使えるば、
他のソフトも使える

アップルが開発・販売

対応CPU: RISCチップ PowerPC

アップルとIBM、モトローラが提携

Intel系CPU, Windowsと互換性なし



Mac OS X 2006年～

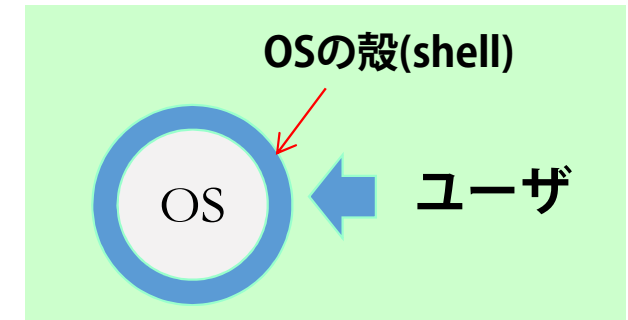
UNIX ベース

対応CPU: Intel x86-64 系

OS X以降アップルのPCではブートロー
ダをEFI(次世代BIOS)に対応させること
でLinux、Windowsをブート可能

シェル

シェル (shell) は OS のユーザーインタフェース
カーネルへのアクセスを提供するコマンドインタプリタ



UNIX系 : sh, bash, csh, tcsh

Windows : cmd.exe

- プログラムの実行 `$ コマンド名 [引数1 引数2 ... 引数n]`
- ファイル名の置換 `$ ls a*` → a から始まるファイルの表示
- 入出力の切り換え `$ ls a* > file` → file に出力
- パイプ機能 `$ ls | sort -r` → ファイル名のリストを逆にソートして表示
- ユーザの環境設定 `$ set PATH= "/bin:/usr/bin"` → 環境変数PATHの内容を /bin:/usr/bin に設定
- インタプリタ型のプログラミング言語

特殊文字	意 味
*	0個以上の任意文字と一致
?	任意の1文字と一致

特殊文字	意 味
<	後のfileを入力用に使用
>	後のfileを出力用に使用
>>	後のfileを追記出力用に使用

```
#!/bin/csh
set n=1
while ( $n <= 10 )
    echo $n
    @ n = $n + 1
end
```

```
#!/bin/csh
set save_file = "result"
echo "PATH を入力して下さい: "
set dir = $<
echo "ファイル名を入力して下さい:"
set name = $<
echo "結果"
echo ""
find $dir -name "$name" -print >! $save_file
echo "Date = 'date'" >>! $save_file
cat $save_file
```

UNIX コマンド

表示	ls	: ファイルの一覧表示
	cat <i>file_name</i>	: ファイルの中身の表示
	sort	: 入力された文字あるいは数のリストの行を並び替える
ファイル操作	rm <i>file_name</i>	: ファイルの削除
	cp <i>file_name1 file_name2</i>	: <i>file_name1</i> を <i>file_name2</i> にコピー
	mv <i>file_name1 file_name2</i>	: <i>file_name1</i> のファイル名を <i>file_name2</i> に変更
ディレクトリ操作	mkdir <i>directory_name</i>	: ディレクトリ（ファイルを入れるフォルダ）の作成
	cd <i>directory_name</i>	: ディレクトリの移動
	pwd	: 現在のディレクトリ名の表示
プロセス管理	ps	: プロセス（プログラム）の実行状況を表示
	kill	: プロセスの強制終了
	top	: 実行プロセスの負荷状況を表示（外部プログラム）
その他	date	: 現在の日時を表示
	vi	: テキストファイルの編集
	emacs	: テキストファイルの編集
	gnuplot	: グラフの作成

実行ファイルの作成手順



機械語を実行手順に従って並べたもの

```
0000 33 C0 8E D0 BC 00 7C 8E C0 8E D8 BE 00 7C BF 00
0010 06 B9 00 02 FC F3 A4 50 68 1C 06 CB FB B9 04 00
0020 BD BE 07 80 7E 00 00 7C 0B 0F 85 0E 01 83 C5 10
0030 E2 F1 CD 18 88 56 00 55 C6 46 11 05 C6 46 10 00
0040 B4 41 BB AA 55 CD 13 5D 72 0F 81 FB 55 AA 75 09
0050 ...
```

1. プログラム言語で書かれたソースファイルを作成

```
int main(void){
    return(0);
}
```

main.c

2. 関数（モジュール）の利用に必要なヘッダファイルを用意（C言語系の場合）

stdio.h

3. プリプロセッサにより、ソースファイルにヘッダファイルを挿入、マクロの展開（文字の置き換え）を行う

```
#include <stdio.h>
#define PI 3.1415926
```

4. 前処理済みのソースファイルをコンパイルしてオブジェクトファイルに変換（変数やラベルは、まだ名前の状態）

main.o

5. 利用するライブラリファイルとオブジェクトファイルを結合させ
変数やラベルのアドレスを確定して実行ファイルを生成

```
main.o
libc.so
libm.so } a.out
```

実行ファイル作成方法

gcc はC言語のソースファイルから

gfortran はFortran言語のソースファイルから 機械語の実行ファイルを作成する (コンパイラ)

gcc, gfortran コンパイラの使用方法和オプションの説明

-c : コンパイルのみ行う (オブジェクトファイル *.o を生成する)

\$ **gcc -c test.c** → ソースコード test.c からオブジェクトファイル test.o を生成

-o file_name : コンパイル,リンクして実行ファイルを file_name として生成する
-o がついていなければ a.out という実行ファイルが生成される

\$ **gcc test.c -o test** → test という実行ファイルを生成

-lm : -lはライブラリをリンクすることを表し, mはライブラリの名前(libm.a)を表す

\$ **gcc test.c -lm -lgw -o test** → libm.a, libgw.a というライブラリをリンクして
コンパイルし, 実行ファイル test を生成

OS から実行ファイルへの制御の移行

実行ファイルは、返却値（出力）と引数（入力）をもつ関数 : オブジェクト

実行プログラム main に CPU の制御が移るときの流れ

1. 実行ファイルへの引数のデータが格納されたアドレスがスタック領域に追加される
2. そのスタック領域のアドレスがSPレジスタにセットされる。
3. OS へのリターンアドレスをスタック領域に追加してOS から関数 main のプログラムコードの先頭アドレスにジャンプする。（IPが移される。）
4. 関数 main はスタック領域に記録された引数のデータが格納されたアドレスをSPレジスタを使って読み取り、引数を受け取り、実行を開始する。

```
返却値の型 関数名(引数の型 引数){  
  
    return(返却値)  
}
```

```
int main(void){  
  
    return(0);  
}
```

5. mainのプログラムコードが終了したときには返却値をAXレジスタにセットし、エラーコードとしてOSに返し、リターンアドレスをスタック領域から呼び戻して、そのアドレスへジャンプする。（IPがOSのプログラムコードに戻る。）