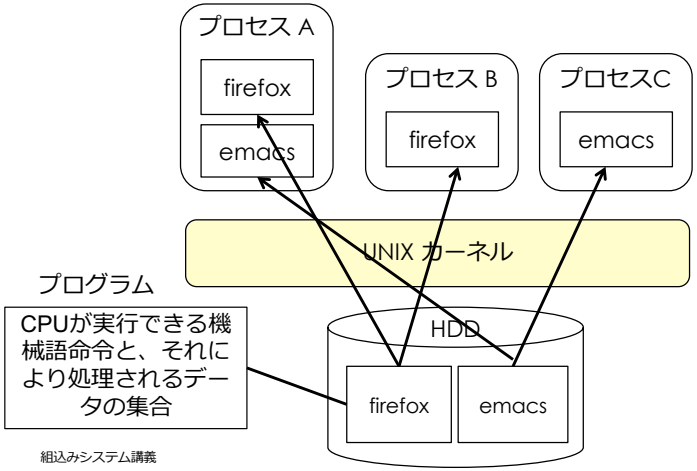
システムプログラミング
プロセス

芝浦工業大学 情報工学科
菅谷みどり

プログラムの実行環境

- **プロセスとは**
  - プログラムを実行するもの



プログラムとプロセス

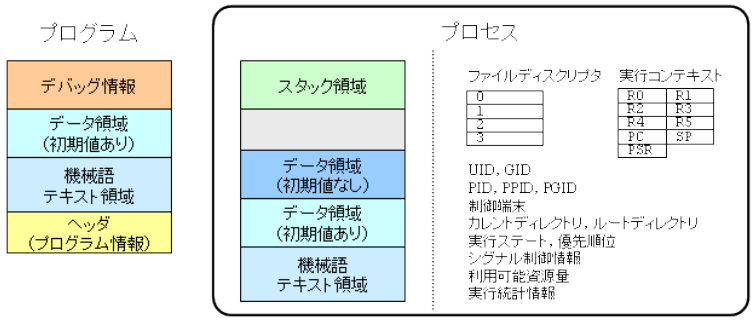
- **プログラム**
  - 永続化されたデータ
    - プログラムは実行される前の状態
    - スタティック（静的）にメモリを割り当てられている
  - CPUが実行できる機械語命令
    - それにより処理されるデータ集合（実行形式、ロードモジュール）
  - がファイルに格納されたもの
  - コンパイルされたプログラムの中身をのぞいてみよう

- $ objdump –d strncomp
- $ readelf –s –h strncomp

  - ファイルに格納されている
    - 機械語命令、データ集合（実行形式、ロードモジュール）

プログラムとプロセス

- **プロセス**
  - RAM 上に展開された実行イメージ
  - プロセスには、実行中の情報が入っている
    - 実行が進むと、書き換えられたり、追加されたりする
    - 実行中のデータを保存するデータ（スタック）



プログラムの実行

- **実行は通常、シェルから行う**

```
$ ./strncmp
ubuntu@domU-12-31-39-10-29-49:~/SysProgs$ ./strcmp
String1 : ABC
String2 : ABC
strcmp(ABC S1, ABC S2) = 0
```

- **シェル**
  - ➔ プログラムのファイル名
  - ➔ プログラムを実行するためのプロセス起動
  - ➔ 実行
  - ※プロセスは誰か（そのプロセス自身でもよい）が終了させないと，いつまでも動いている．
  - ※ 終了させるためには，そのための手順を踏む必要がある（システムコールを呼ぶか強制的に終了させる）．

プロセスの観察

- **プロセスを観察するためのツール ps の実行**

```
$ ps
$ ps xu
```

- **プロセスツリーを観察する**

```
$ pstree
```

- **CPU時間などが長いプロセスを確認する**

```
$ top –d 1
```

  - CPU, MEM を使用中のプロセスの状態を一定時間おきに表示.
  - 表示間隔は -d オプションの引数として秒単位で指定できる.

プロセスの機能

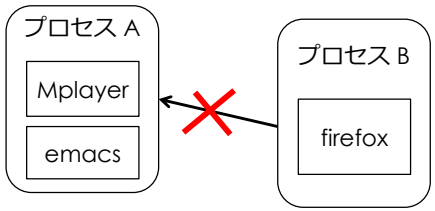
- **資源割当**
- **保護**

資源割当

- **UNIX におけるプロセス**
  - プログラム（命令）された処理を行うにあたり、必要となる計算資源の割り当ての単位
    - プロセッサ時間、メモリ、ファイル、キーボード、ディスプレィ、プリンタなどのデバイス
- **プロセスによるプログラムの実行手順**
  - ユーザによるプログラムの起動
  - ⇩
  - OS カーネルによる資源の割り当て
    - プロセッサの利用時間
    - プログラムを実行するためのメモリ
  - ⇩
  - プロセスによるプログラムの実行
    - ファイルをオープン、アクセス、キーボド、デバイスの利用
    - ファイルオープン時のファイルディスクリプタは固有

保護

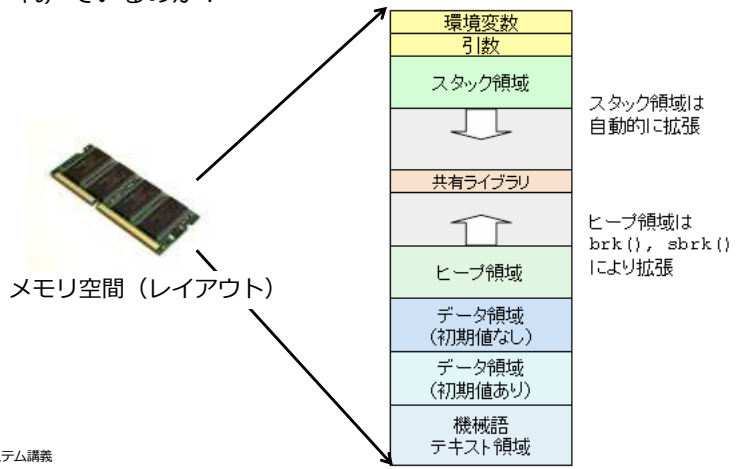
- **プロセスは、割り当てられた資源が保護される単位**
  - あるプロセスは、他のプロセスに割り当てられた資源に対して、許可なくアクセスすることはできない



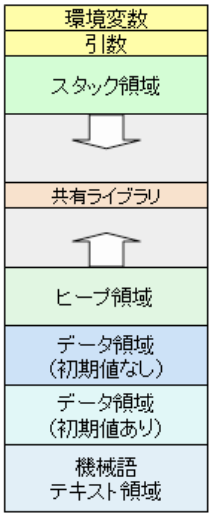
  - 保護の機能により，例えばある暴走したプロセスが他のプロセスの実行内容を破壊するようなことは起こらない．

プロセスのメモリマップ

- **メモリ上の割り当て**
  - プロセスは、メモリ上にどのように割り当てられ（マップされ）ているのか？



プロセスのメモリマップ



テキスト領域	プログラムの機械語命令が置かれる．（読み出し専用領域）同じプログラムから起動されるプロセスの間で共有可能
データ領域（初期値有）	プログラム中の0 以外の初期値を持つ大域 (global) 変数，静的局所 (static local) 変数が置かれる
データ領域（初期値無）	通称 BSS (Block Started by Symbol) セグメント．初期値を持たない，又は初期値が 0 の大域変数，静的局所変数が置かれる．プロセス作成時に確保され，0 に初期化される．
ヒープ領域	malloc()などにより，プロセス実行時に確保されるデータ領域．
共有ライブラリ	共有ライブラリのため領域はヒープとスタックの間にとられる．テキスト領域と同じく読み出し専用で，他のプログラムと共有される．
スタック領域	C言語の自動変数（staticでないローカル変数）や，引数，関数呼び出し時の戻り番地などが置かれる．
引数，環境変数	コマンドに渡される引数，環境変数は，スタック領域の最上位部分に格納されている

メモリマップを確認しよう

```
1 #include <stdio.h>
2
3 extern char **environ;
4 int data0;
5 int data1 = 10;
6
7 main(int argc, char *argv[])
8 {
9 char c;
10 printf("environ:¥t%0.8p¥n", environ);
11 printf("argv:¥t¥t%0.8p¥n", argv);
12 printf("stack:¥t¥t%0.8p¥n", &c);
13
14 printf("bss:¥t¥t%0.8p¥n", &data0);
15 printf("data:¥t¥t%0.8p¥n", &data1);
16 }
```

このプログラムをコンパイルして実行した結果を観察しよう

- 環境変数の文字列のアドレス位置は？
- コマンド引数の文字列
- Bss セクションと、データセクションのどちらが上位アドレスか？

プロセスの属性

PID (プロセスID)	それぞれのプロセスにつけられる、プロセスを識別するための番号、Linux では 0 – 32767
PPID (親プロセスID)	そのプロセスを作成したプロセス（親プロセスID)
GPID (プロセスグループID)	所属するプロセスグループのID, プロセスグループは、まとめてシグナルを送る場合などに使用される
UID (ユーザID)	プロセスを実行したユーザのID, これとは別にアクセス権限を示す実効ユーザID もある
GID (グループID)	プロセスを実行したグループのID．これとは別にアクセス権限を表す実効グループIDもある．
ファイルディスクリプタ	オープンしたファイルの数
Umask	ファイル作成時のモードを決めるときに、マスク値として使用される値

プロセスの属性 (cond't)

制御端末	シグナルを受け取る端末
カレントディレクトリ	現在のディレクトリ、相対パス名を使う場合の出発点となる. Current working directory ともいう
ルートディレクトリ	ルートディレクトリは、プロセスごとに決めることができる. 通常、アクセスできるファイルを制限するために使用する.
実行ステート	実行中か、停止中か、ゾンビ状態か、などのプロセスの実行状態
優先順位	プロセスの優先順位
シグナル制御情報	シグナルに対応して、どの処理が行われるのかの情報
利用可能資源量	プロセスの利用できる資源の上限
実行統計情報	これまでの資源利用用の統計情報

プロセスを操作するコマンド

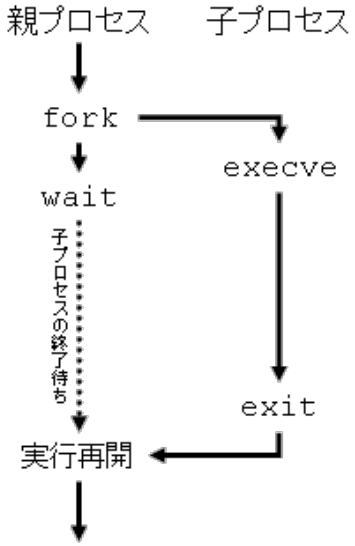
- **シェル**
  - コマンドの実行（プロセス作成、プログラム実行）、リダイレクション、パイプ、その他… etc
- **ps, pstree, top**
  - プロセスの観察
- **Kill**
  - プロセスの（強制）終了
- **nice**
  - プロセスの実行優先順位の制御
- **limit**
  - プロセスの利用可能資源の制御
- **gdb, strace**
  - デバッグツール

プロセス作成、プログラム実行、プロセス終了

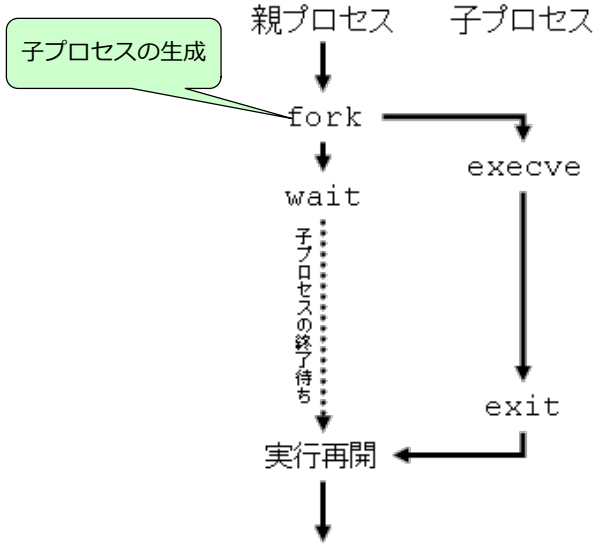
- **プロセスの作成とプログラムの実行**
  - ユーザのプログラムはプロセスにより実行される
- **プロセスによる実行の手順**
  1. シェルを実行するプロセスは，コマンドを実行するプロセスを作成する (fork)
  2. 作成されたプロセスは，指定されたコマンドを実行する (execve)
  3. シェルは，コマンドを実行するプロセスが終了するのを待つ (wait)
  4. コマンドが終了すると，シェルの実行が再開される (exit)

```
pid_t fork(void);
int execve (const char *filename, char *const argv[], char
*const envp[]);
pid_t wait(int *status)
void exit(int status);
```

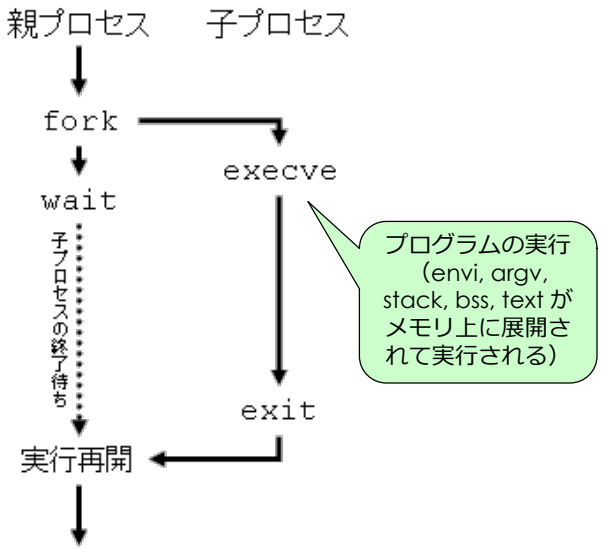
プロセス作成，実行，終了



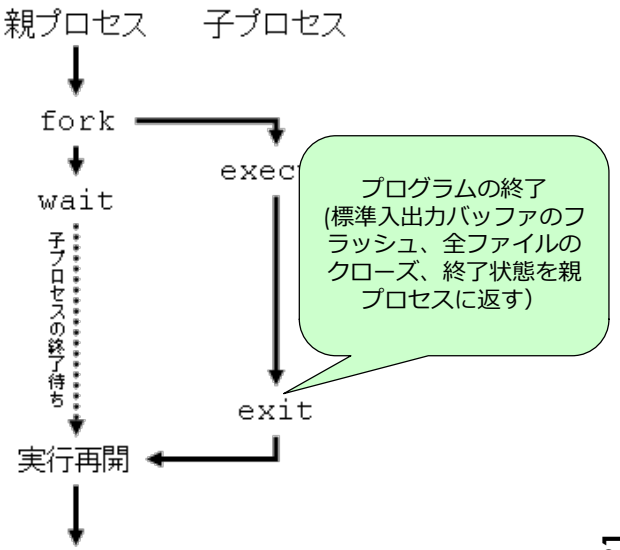
プロセス作成，実行，終了



プロセス作成，実行，終了



プロセス作成，実行，終了



プロセス作成，実行，終了

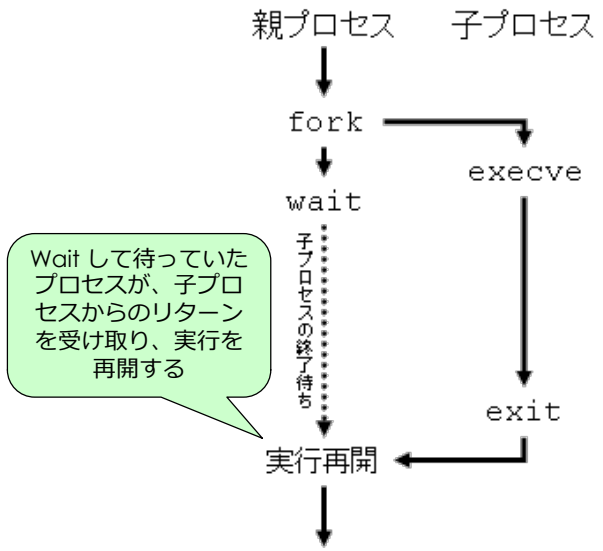
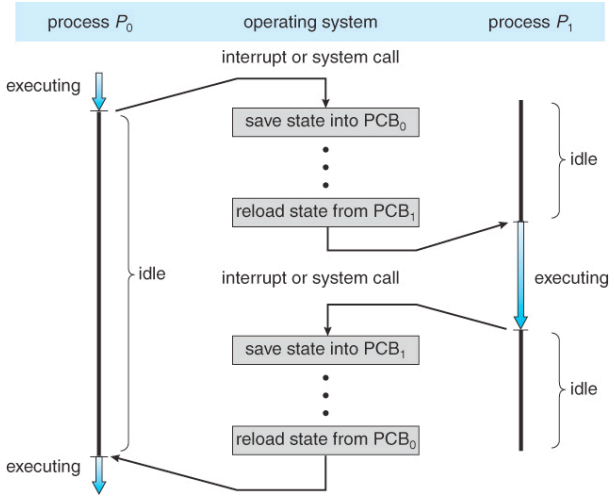


Diagram showing CPU switch from process to process



For Multi-Tasking

- **Is it sufficient for achieving multi-tasking OS**
  - That provides the mechanism to switching process mechanism?

プロセス生成プログラムを書いてみよう

- **仕様**
  - fork() APIを調べて, 親プロセスが子プロセスを生成するプログラムを作成してみる
  - 子プロセスは、生成されたら、自分のPIDを表示して終了する
    - getpid()
    - exit()
  - 親は子プロセスの実行を待つ
    - Wait()
  - 子プロセスが終了したら、親は子供が終了した、ということをコンソールに表示した上で、自分も終了する
    - printf()

演習1

- **課題１：**
  - Fork()の動作を説明してみよう
- **課題2:**
  - 親(parent) プロセスのID を取得してみよう (this is parent (pid = XXXX)). getpid() を追加
  - Parent ID と，child ID の違いを比較してみよう
- **課題3**
  - 子プロセスを沢山つくってみよう
  - １０個，１００個，１０００個
- **課題4**
  - 親プロセスと，１００個の子プロセスのプロセスID を比較してみよう
- **課題5**
  - 実行時間をはかってみよう，１個と１００個の時を比べてみよう
    - time コマンドを利用する
    - time ./fork

演習2

- **演習で相談した内容について**
  - 沢山の子プロセスに何の処理をさせると良いか相談してみよう
  - 親と子の 1-to-多の関係を応用してどんなプログラムが作成できるか相談してみよう
- **実際親と子，1-to-多の関係を用いたプログラムを作成する**
  - 非同期（同時並行的）でも良い
  - もし，互いにデータ共有したい場合は，パイプで行う
- **演習1, 2 とも、Exec05 フォルダに提出してください**

プログラム例 (execve)

```
#include <stdio.h>
2
3 extern char **environ;
4
5 main()
6 {
7       char *argv[2];
8
9       argv[0] = "/bin/ls";
10      argv[1] = NULL;
11
12      execve(argv[0], argv, environ);
13 }
```

execve は、プロセスの実行を上書きするシステムコールである

- /bin/ls のプログラムがロードされ、ls の main から実行が開始される (/bin/ls を実行するプログラムとなる)
- 引数、環境変数はそれぞれ argv 形式で渡す必要がある
- 環境変数は現在の環境変数そのまま、argv は NULL を渡している

プログラム例2 (execve) 引数

```
#include <stdio.h>
2
3 extern char **environ;
4
5 main()
6 {
7       char *argv[3];
8
9       argv[0] = "/bin/ls";
10      argv[1] = "/";
11      argv[2] = NULL;
13
14      execve(argv[0], argv, environ);
15 }
```

実行結果

```
% ./a.out
    bin dev home lhome lost+found mnt proc sbin usr work boot etc initrd lib
    misc opt root tmp var
%
```

演習2

- **課題1: 子プロセスに別のプログラム（コマンド）を実行させるように変更してみよう**

- **演習2の内容をテキストにまとめてExec05 フォルダに提出**

プログラム実行のためのライブラリ関数

- **プログラムの実行**
  - execve は引数や環境変数を argv 形式で渡す必要がある
  - プログラムをサーチしてくれないため、もう少し簡単に使えるように以下のライブラリ関数が用意されている

```
int  execl(const char *path, const char *arg, ...);
int  execlp(const char *file, const char *arg, ...);
int  execle(const char *path, const char *arg , ..., char * const envp[]);
int  execv(const char *path, char *const argv[]);
int  execvp(const char *file, char *const argv[]);
```

- **execl, execlp, execle**
  - コマンド引数の渡し方が execve とは異なり、これらのライブラリ関数の引数として、コマンド引数のように文字列を並べて渡す
  - execl(“/bin/ls”, “ls”, “/”, NULL);
  - execlp(“ls”, “ls”, “/”, NULL);