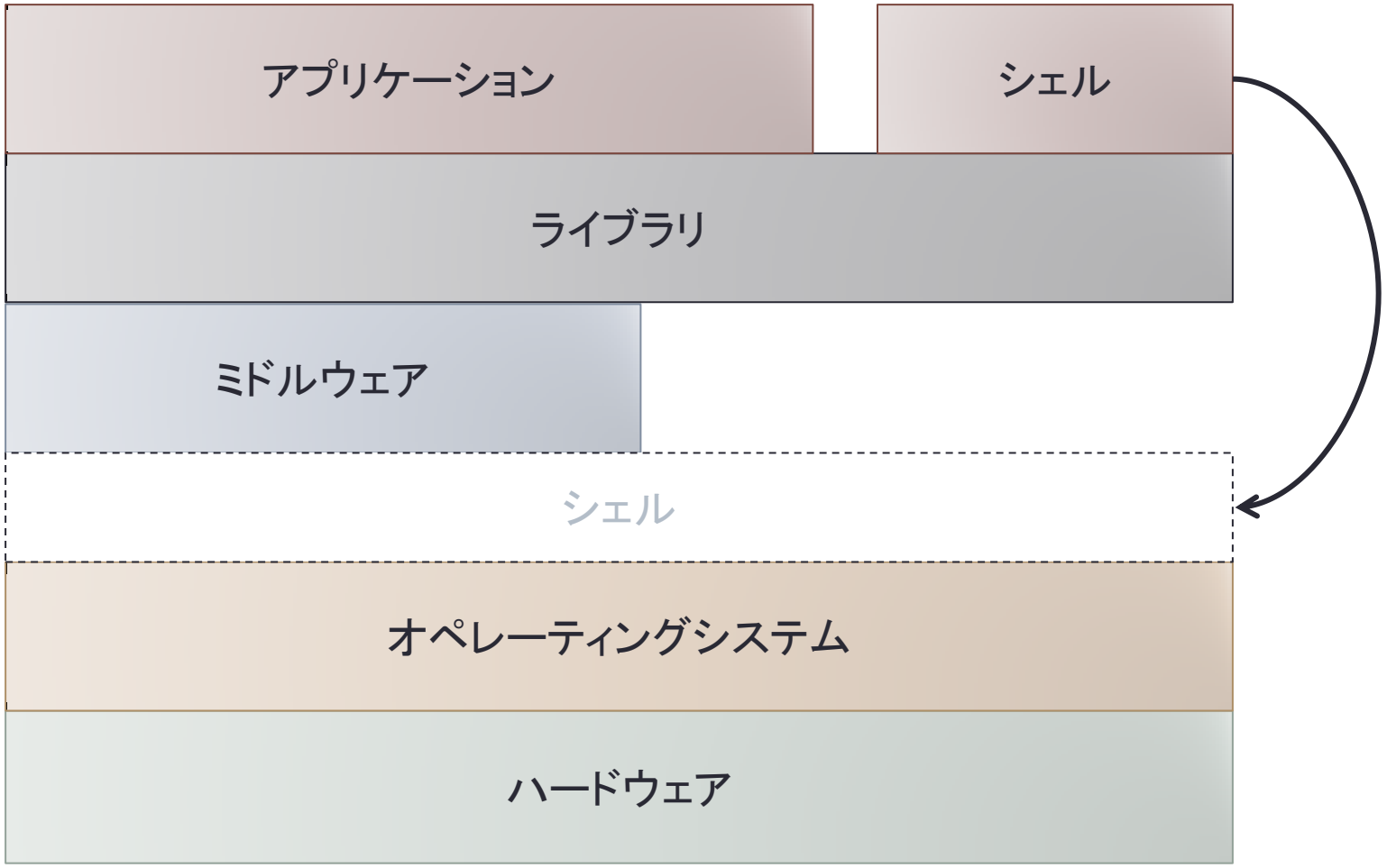
ソフトウェアアーキテクチャ
第3回　シェル

環境情報学部
萩野　達也

スライドURL

https://vu5.sfc.keio.ac.jp/slide/

ソフトウェア階層



シェルの機能

- プログラムの起動
- プログラムの制御
  - バックグラウンド
  - フォアグラウンド
- プログラムの入出力のリダイレクション
  - パイプ

プログラムの起動・制御

- 環境変数・シェル変数の設定

プログラムの実行環境の設定

- ワイルドカードの展開
- ヒストリー機能
- エイリアス機能
- ファイル名の補間

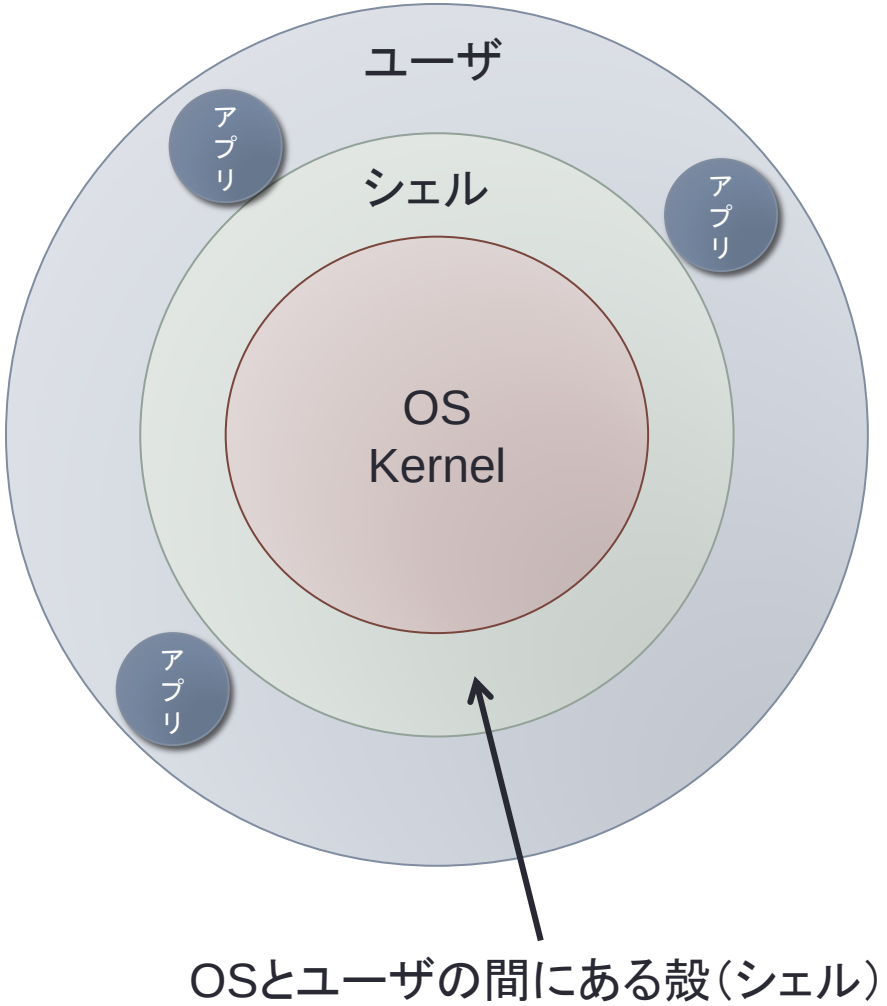
便利な機能の提供

- 繰り返しや条件分岐
- シェルスクリプトの実行

スクリプトの実行

シェルの種類

- Windows
  - Command Prompt
  - PowerShell
  - Explorer
- Mac
  - Finder
  - Launcher
- UNIX
  - sh 系
    - Bourne Shell
    - Korn Shell
    - Z Shell
  - csh 系
    - C Shell
    - TENEX C Shell



UNIXシェルのコマンド

- シェルはユーザからの指示をOSに伝える役割を果たす

- シェルの組み込みコマンドは非常に少ない
  - **set**, **alias**, **cd**, **setenv** など

- ほとんどのコマンドはプログラムとして実行しているだけ
  - ディレクトリの中身を見る **ls** もプログラム
  - ファイルの中身を見る **cat** もプログラム

シェルの大まかな動き

```
shell() {
  char buf[512], char *argv[512];
  for (;;) {
1     printf("% ");
2     if (!fgets(buf, sizeof(buf), stdin)) break;
3     parse(buf, argv);
4     execute(argv[0], argv);
  }
}
```

1. プロンプトを出す
2. 1 行読み込み
3. 空白で区切られたコマンドと引数を切り分け
4. コマンドを与えられた引数のもので実行
5. 最初に戻る

プログラムの起動

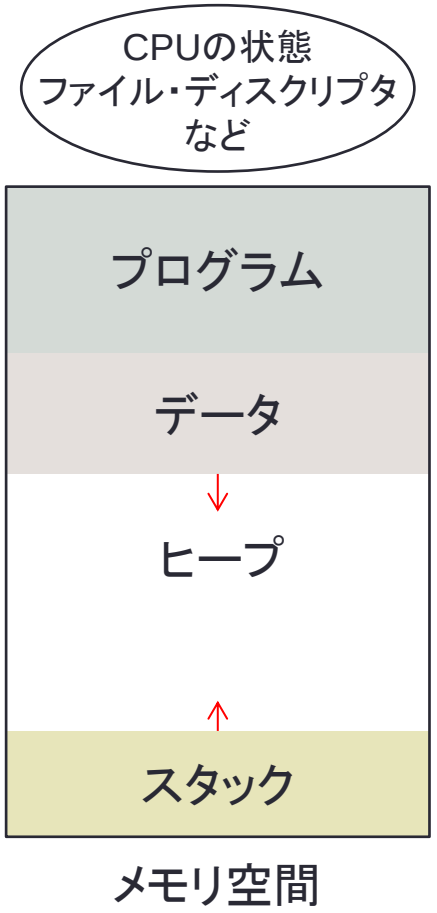
- 子プロセスを作りプログラムを起動

```
execute(char *cmd, char *argv[]) {
  int pid, status;
  pid = fork();
  if (pid == 0) {
    execve(cmd, argv, NULL);
    fprintf(stderr, "command %s not found¥n", cmd);
    exit(1);
  }
  while (wait(&status) != pid);
}
```

- シェルは親プロセスとして子プロセスが終了を待つ

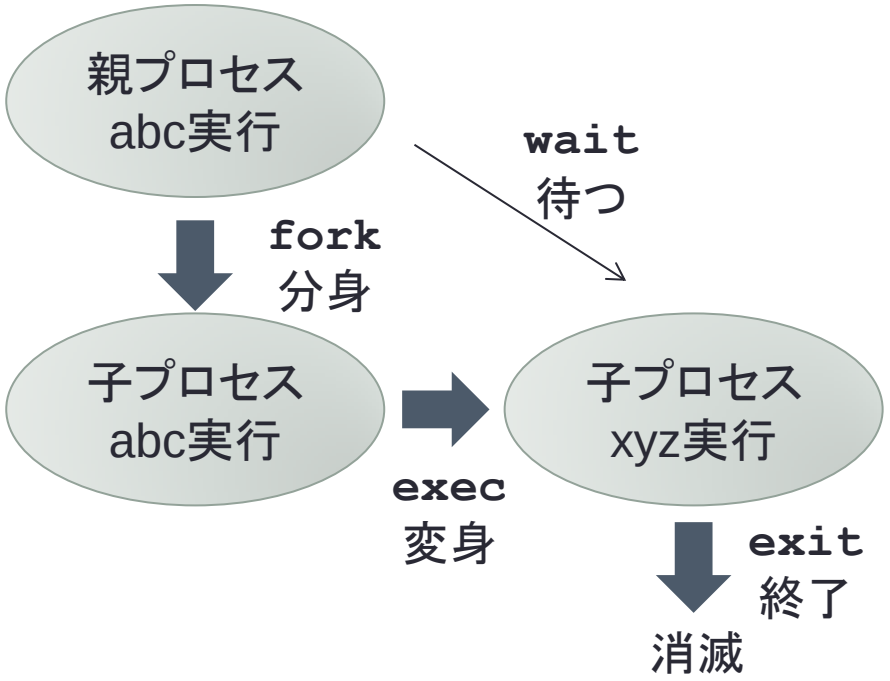
プロセス

- 実行状態のプログラム
  - 同じプログラムを実行している複数のプロセスが存在する.
- プロセスは以下のもので構成されている:
  - プログラム
  - CPUの状態(レジスタ, PC, SP)
  - データ
  - メモリ空間
  - ファイル・ディスクリプタ
  - 現在の作業ディレクトリ
  - ルートディレクトリ
  - その他のプロセス関係の状態



プロセス関係のシステムコール

- **fork**
  - 子プロセスを作る
  - 中身は親プロセスと同じ
  - 分身
- **exec**
  - 実行するプログラムを指定する
  - 現在のプログラムを捨てる
  - 変身
- **wait**
  - 子プロセスが停止するまで待つ
- **exit**
  - プログラムを停止する
- その他
  - **signal**
    - 割り込み処理の指定
  - **kill**
    - 処理の割り込み



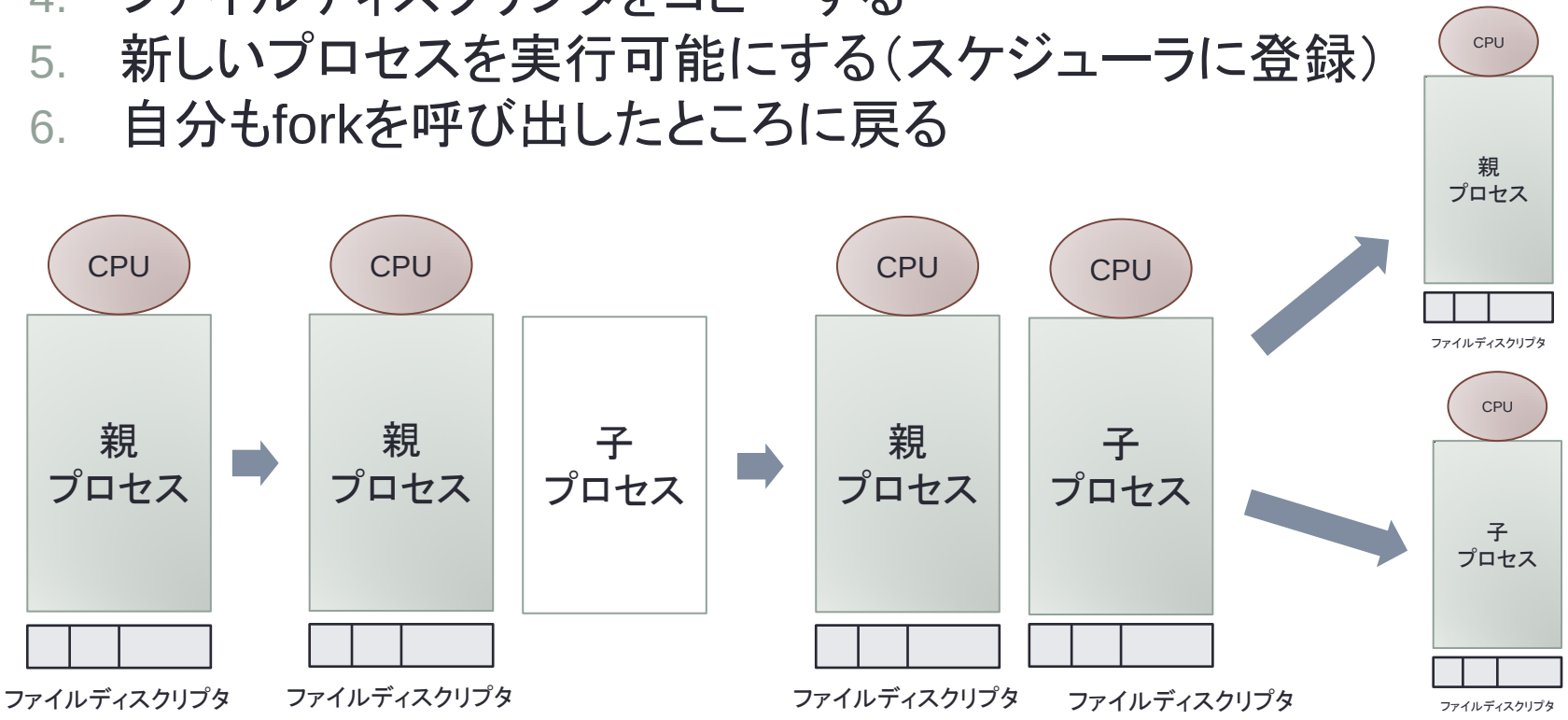
プロセスには親子関係がある

forkとexec

- fork
  - 親からすべて受け継ぐ
    - 分身
  - メモリ空間
    - プログラム
    - データ
    - スタック
  - ファイルディスクリプタ
  - 環境変数
  - メモリはコピーされるのではなくシェアされる
    - どちらかが変更したときにコピー
    - copy-on-write
- exec
  - 親からの自立
    - 現在のメモリの内容を捨てる
    - 指定されたファイルのプログラムの内容にメモリを置き換える
  - プログラムのエントリポイントから実行を開始
    - main
  - 引き継ぐもの
    - ファイルディスクリプタ
    - execへの引数
      - 環境変数を含む
  - プログラムはすぐには読み込まれない
    - 必要になってから読み込まれる
    - demand paging

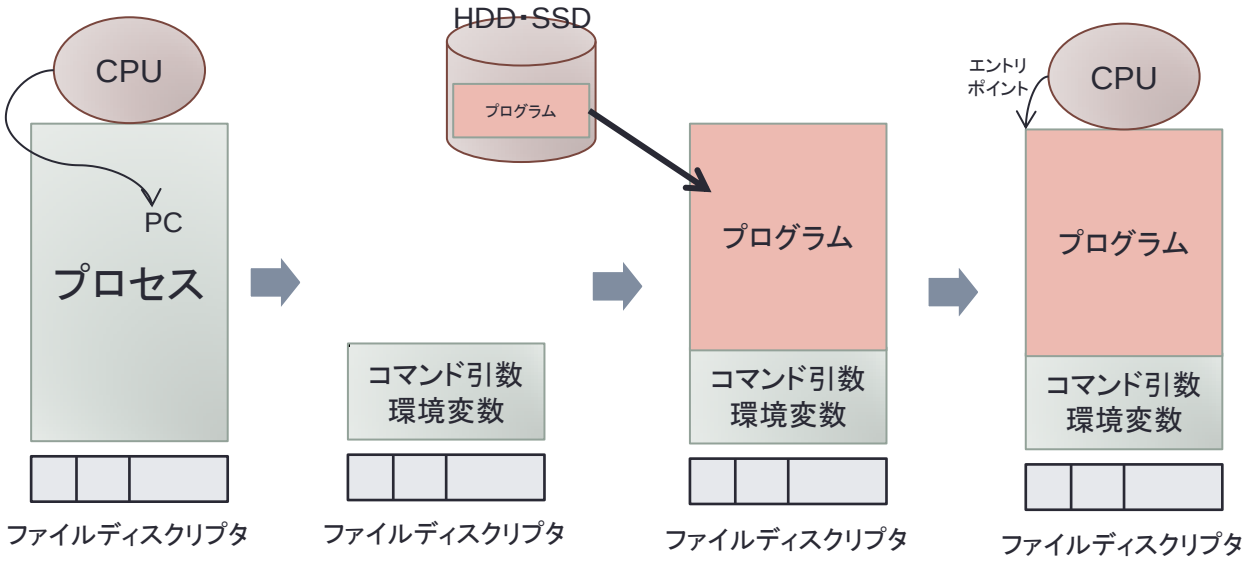
forkの処理

1. 新しいプロセスを作り，現在のプロセスの子供にする
2. CPUの内容をコピーする
   - アキュムレータやプログラムカウンタ，スタックポインタなどすべて
3. メモリの内容をコピーする
   - 実際にはcopy-on-writeによる共有設定
4. ファイルディスクリプタをコピーする
5. 新しいプロセスを実行可能にする（スケジューラに登録）
6. 自分もforkを呼び出したところに戻る



execの処理

1. 作業のために新しいメモリ(作業メモリと呼ぶ) を確保
2. コマンド引数と環境変数を作業メモリにコピー
3. 現在プロセスが使っているメモリをすべて解放
4. コマンドで指定されたプログラムをメモリ上に展開
5. スタックにコマンド引数を環境変数を作業メモリからコピー
6. 作業メモリを解放
7. プログラムのエントリポイントから実行を開始



コマンドPATH

- execve の第1引数はプログラムのパス名
  - execve("ls", ...) は実行できない
  - execve("/bin/ls", ...) でなくてはならない
- いちいち絶対パスを書くのは面倒
- 環境変数 PATH にコマンドのあるディレクトリを設定

  /bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin
- それぞれのディレクトリの中のプログラムの実行を次々と試していく

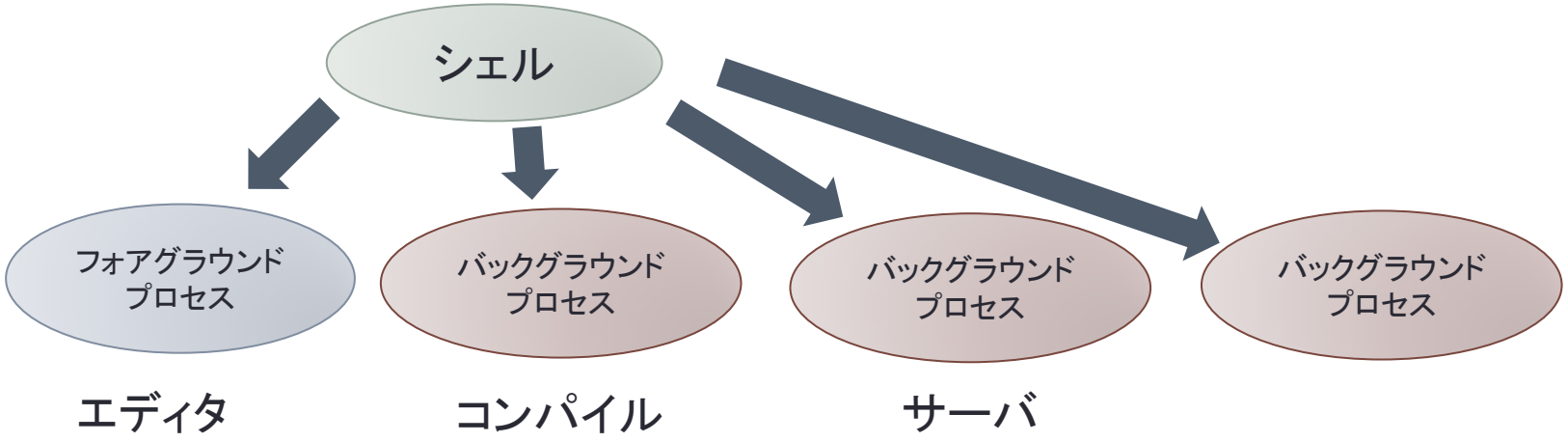
```
execute(char *cmd, char *argv[]) {
  int pid, status;
  pid = fork();
  if (pid == 0) {
    execve("/bin/" + cmd, args, NULL);
    execve("/usr/bin/" + cmd, args, NULL);
    execve("/sbin/" + cmd, args, NULL);
    execve("/usr/sbin/" + cmd, args, NULL);
    execve("/usr/local/bin/" + cmd, args, NULL);
    fprintf(stderr, "command %s not found¥n", cmd);
    exit(1);
  }
  while (wait(&status) != pid);
}
```

成功した後のexecveは
実行されない

バックグラウンドとフォアグラウンド

- フォアグラウンド
  - 通常の実行
  - コマンドを一つずつ実行

- バックグラウンド
  - コマンドの後に & を付けて指定する
  - シェルはコマンドの終了を待たずに次のコマンドを実行
  - 同時に複数のバックグラウンドコマンドを実行可能



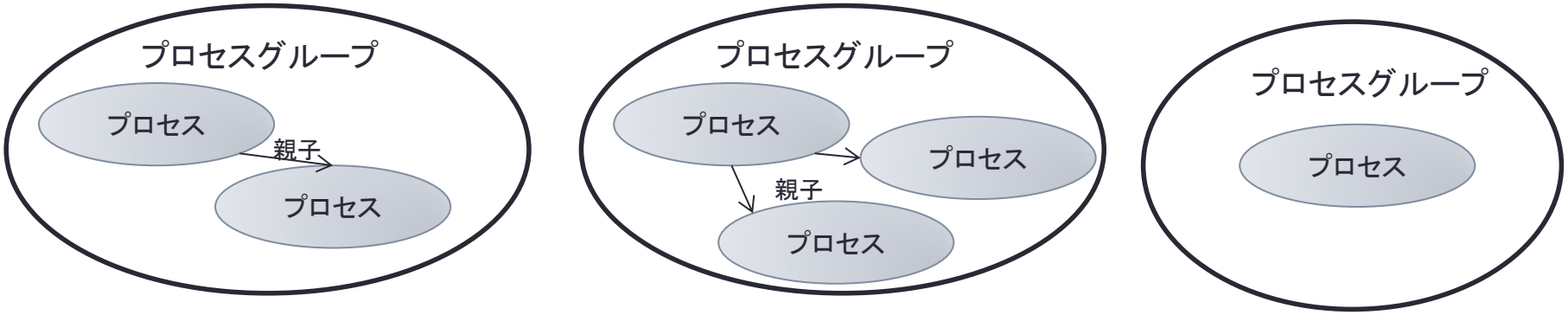
バックグラウンドの実装

- シェルは **wait** でコマンドの終了を待つ
- 終了を待たなければバックグラウンド

```
execute(char *cmd, char *argv[], int foreground) {
  int pid, status;
  pid = fork();
  if (pid == 0) {
    execve(cmd, argv, NULL);
    fprintf(stderr, "command %s not found¥n", cmd);
    exit(1);
  }
  if (foreground) {
    while (wait(&status) != pid);
  }
}
```

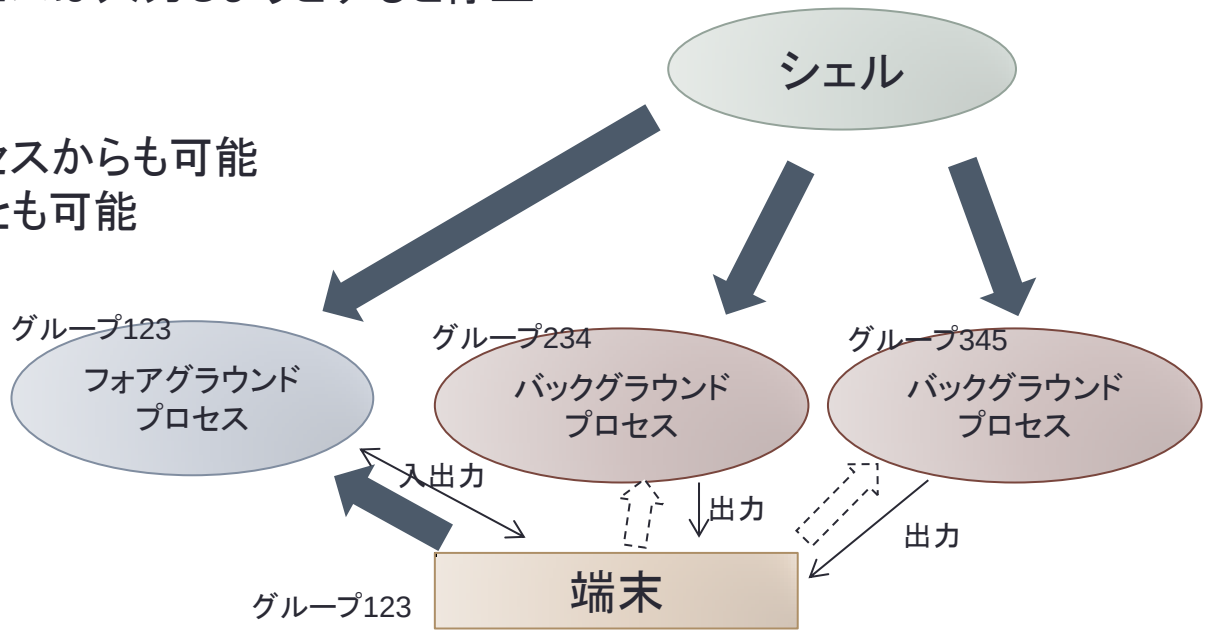
ジョブ制御

- 一つの処理が一つのプロセスとは限らない
  - パイプによるコマンドの結合
  - コマンドが`fork`で子プロセスを作る
- 処理のかたまりを**ジョブ**という
  - ジョブ単位でフォアグラウンド，バックグラウンドは切り替わる
- UNIX 的にはプロセスグループを用いる
  - プロセスはあるプロセスグループに所属
  - 子プロセスは通常同じプロセスグループに所属
  - シェルはコマンドを起動するたびに別のプロセスグループを生成



端末からのジョブ制御

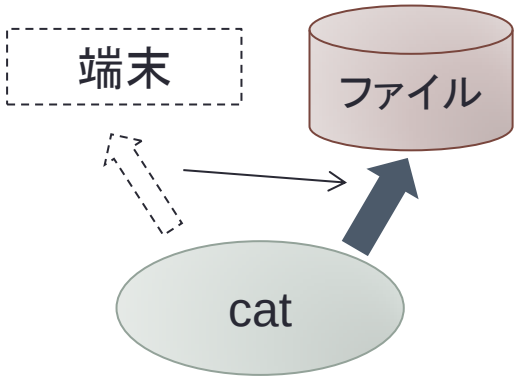
- 端末もプロセスグループを持っている
  - `ioctl` の`TIOCSPGRP` で設定
- フォアグラウンド
  - 端末のプロセスグループと一致するプロセスグループ
- 端末からの入力
  - フォアグラウンドプロセスにのみ送られる
  - バックグラウンドプロセスは入力しようとすると停止
- 端末への出力
  - バックグラウンドプロセスからも可能
  - できないようにすることも可能



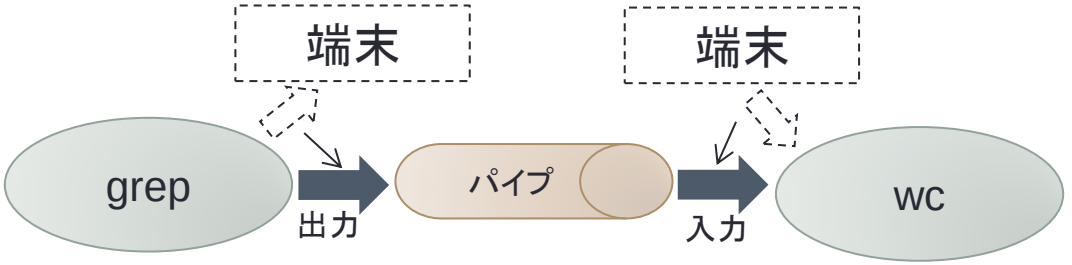
リダイレクション

- コマンドの標準入力・出力・エラー出力を端末からファイルに変更できる
  - `% cat /etc/passwd > /tmp/aaa`
  - `% wc < /etc/passwd`

- パイプにより2つのコマンドを結合
  - コマンドの出力をもう一つのコマンドの入力にする
  - `% grep abc /etc/passwd | wc`



標準出力の切り替え



パイプによりコマンドを結合

リダイレクションの実装

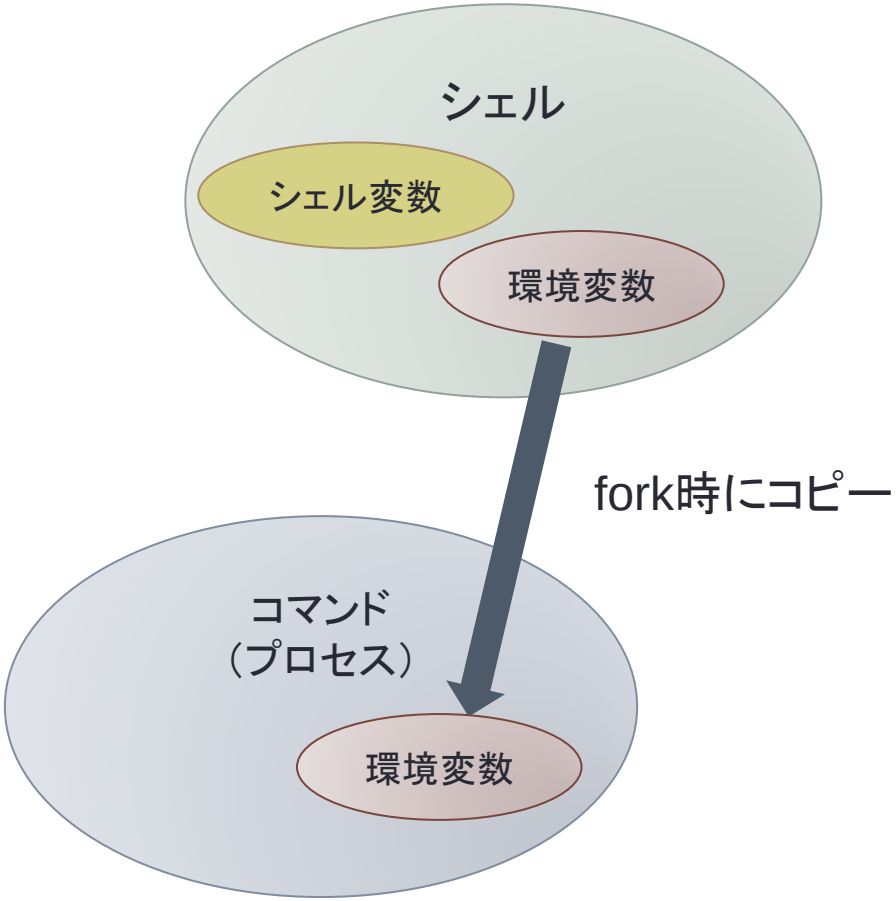
- ファイルディスクリプタは **exec** で引き継がれる
- **exec** を行なう前にファイルディスクリプタを変更

```
execute(char *cmd, char *argv[]) {
  int pid, status;
  pid = fork();
  if (pid == 0) {
    fd = open("/etc/passwd", O_RDONLY);
    dup2(fd, 0);
    close(fd);
    execve(cmd, argv, NULL);
    ...
  }
  while (wait(&status) != pid);
}
```

- **dup2** はファイルディスクリプタをコピーするシステムコール
- **fork** した後でないとシェルのファイルディスクリプタも変更してしまう
- **fork** と **exec** が分離している理由の一つ

シェル変数と環境変数

- シェル変数
  - シェルでの利用
  - シェルの設定の変更
  - シェルプログラムでの活用
- 環境変数
  - コマンドに受け渡される
  - ユーザ名
  - ホームディレクトリ名
  - PATH



ワイルドカードの展開

- シェルのコマンドには‘*.c’などを指定可能
- ‘*.c’の展開はシェルが行なう

```
% ls *.c
% cat a???.b?
```

```
DIR *dp;
struct dirent *de;
dp = opendir(dir);
while (de = readdir(dp)) {
  if (match(name, de->d_name)) {
    strcpy(argv[argc++], de->d_name);
  }
}
closedir(dp);
```

```
int match(char *pattern, char *p) {
  char ch;
  while (ch = *pattern++) {
    if (ch == '*') {
      while (*p) {
        if (match(pattern, p)) return 1;
        p++;
      }
      return (*pattern == 0);
    }
    else if (ch == '?') {
      if (*p++ == 0) return 0;
    }
    else if (*p++ != ch) return 0;
  }
  return (*p == 0);
}
```

- ‘*’や‘?’を含んだ文字列とのマッチング

シェルスクリプト

test

```
echo Hello
date
ls
```

← `chmod a+x test`

実行可能にする

3つのコマンドを組み合わせてtestと呼ぶ

- シェルのコマンド列を組み合わせて新しいコマンドを作る
  - コマンド列のファイルを作成し **chmod** でモードで実行可能に変更
  - OS はバイナリファイルでないときに，シェルを起動してプログラムを引数として渡す
  - 条件分岐や繰り返しなども可能
  - シェルはスクリプトの先頭行で指定可能

test

```
#!/bin/csh -f
echo Hello
date
ls
```

まとめ

- シェルの機能
  - コマンドの実行
  - ジョブ制御
  - リダイレクション
  - 環境変数
  - ワイルドカードの展開
  - シェルスクリプト

- その他の機能
  - コマンドの別名
  - ファイル名の補完
  - ヒストリー機能