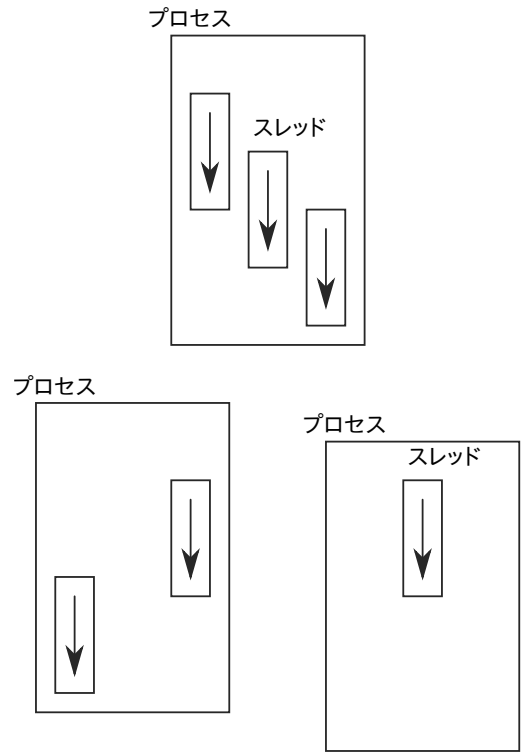
スレッド

田浦健次朗

スレッド

- CPU を分け合うための抽象化
  - スレッドを作らなければ CPU は割り当てられない (従って計算は出来ない)
  - CPU コアを N 個使って処理がしたければ N 個 (以上) スレッドが必要
- ん? プロセスのときもそんな事言ってなかった?

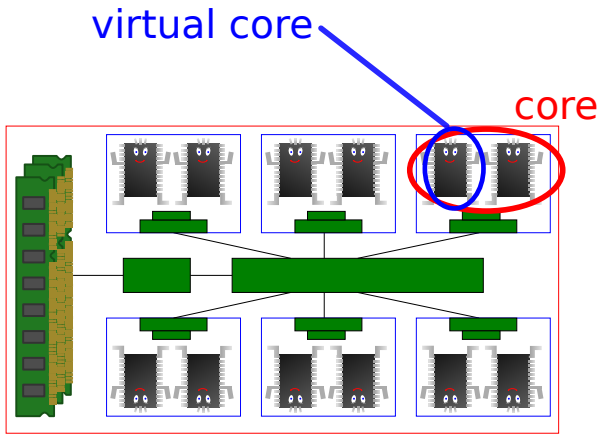


- 1 プロセス中に複数 (≥ 1) のスレッドが存在しうる
- プロセスを作るともれなく一つスレッドができる
  - C 言語ならば main 関数を実行するスレッド

プロセス = アドレス空間 (箱) + 1 つ以上のスレッド

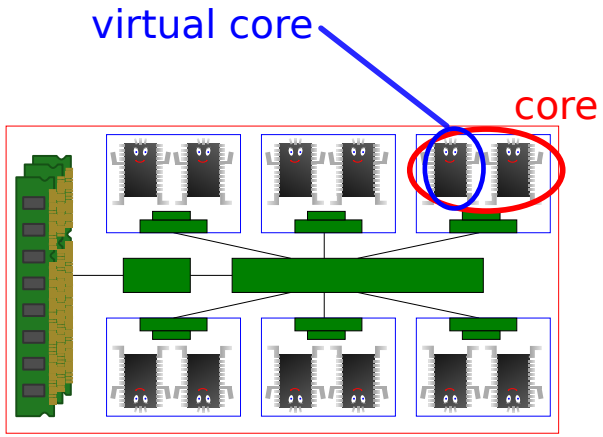
用語の注: CPU, コア, プロセッサ, ...

- CPU ⊃ 物理コア ⊃ 仮想コア
  - 1 CPU に複数 (2〜64 程度) の物理コア (マルチコア CPU)



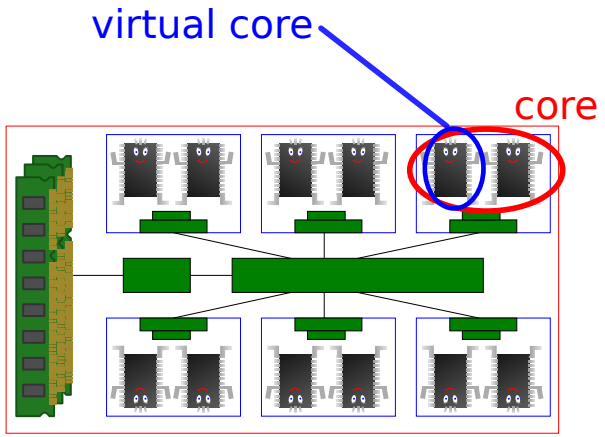
用語の注: CPU, コア, プロセッサ, ...

- CPU ⊃ 物理コア ⊃ 仮想コア
  - 1 CPU に複数 (2〜64 程度) の物理コア (マルチコア CPU)
  - 1 物理コア に複数 (2〜8 程度) の仮想コア (Simultaneous Multithreading (SMT), ハイパースレッディング (Intel 用語))



用語の注: CPU, コア, プロセッサ, ...

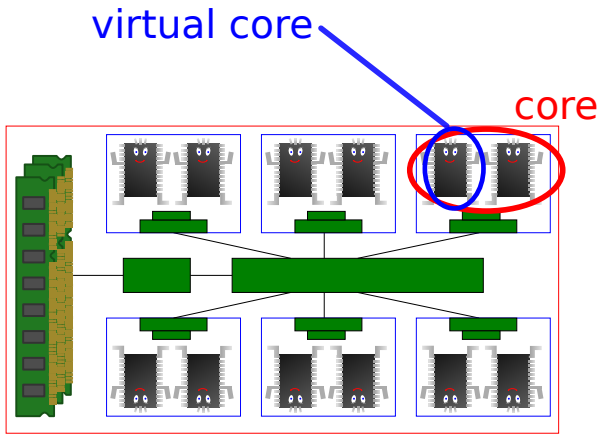
- CPU ⊃ 物理コア ⊃ 仮想コア
  - 1 CPU に複数 (2〜64 程度) の物理コア (マルチコア CPU)
  - 1 物理コア に複数 (2〜8 程度) の仮想コア (Simultaneous Multithreading (SMT), ハイパースレッディング (Intel 用語))



- 各 仮想コア = 「命令の取り出し, 実行」を実行するハードの機能

用語の注: CPU, コア, プロセッサ, ...

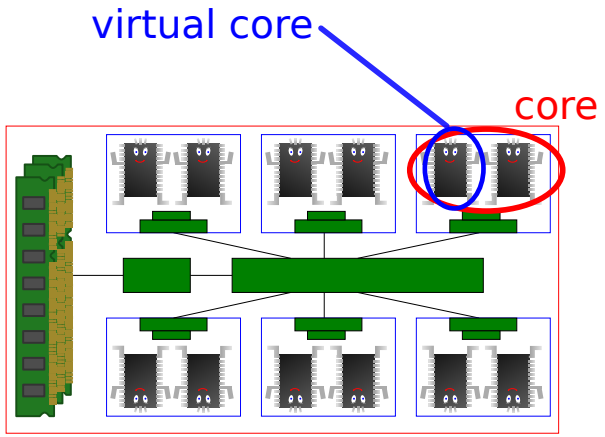
- CPU ⊃ 物理コア ⊃ 仮想コア
  - 1 CPU に複数 (2〜64 程度) の物理コア (マルチコア CPU)
  - 1 物理コア に複数 (2〜8 程度) の仮想コア (Simultaneous Multithreading (SMT), ハイパースレッディング (Intel 用語))



- 各 仮想コア = 「命令の取り出し, 実行」を実行するハードの機能
- つまり仮想コア数分までのスレッドは「ほんとに同時に」実行される

用語の注: CPU, コア, プロセッサ, ...

- CPU ⊃ 物理コア ⊃ 仮想コア
  - 1 CPU に複数 (2〜64 程度) の物理コア (マルチコア CPU)
  - 1 物理コア に複数 (2〜8 程度) の仮想コア (Simultaneous Multithreading (SMT), ハイパースレッディング (Intel 用語))



- 各 仮想コア = 「命令の取り出し, 実行」を実行するハードの機能
- つまり仮想コア数分までのスレッドは「ほんとに同時に」実行される
- スレッドに「CPU を割り当てる」と言うが実際に割り当てているのは, 1 つの仮想コア

ややこしい注

- 物理コア vs. 仮想コア
  - 性能を気にしなければ, 仮想コアと物理コアの違いを意識する場面はほぼ無い
  - 1 物理コア中の仮想コアは演算器を共有
  - ⇒1 物理コアに複数のスレッドを同時に走らせても, (1 つのスレッドだけですでに性能を出し尽くしている場合) 性能が向上しない場合がある
- OS が見せるプロセッサ数は, 普通, 仮想コア数
  - Linux : `cat /proc/cpuinfo`, システムモニタ
  - Windows : タスクマネージャ, リソースモニタ
- 仮想コア = ハードウェアスレッド (同義語)
- プロセッサは曖昧な言葉
  - CPU, 物理コア, 仮想コアなどの意味で使われる
- 単にコアということもしばしば; 物理コア, 仮想コアのどちらを意味するか曖昧

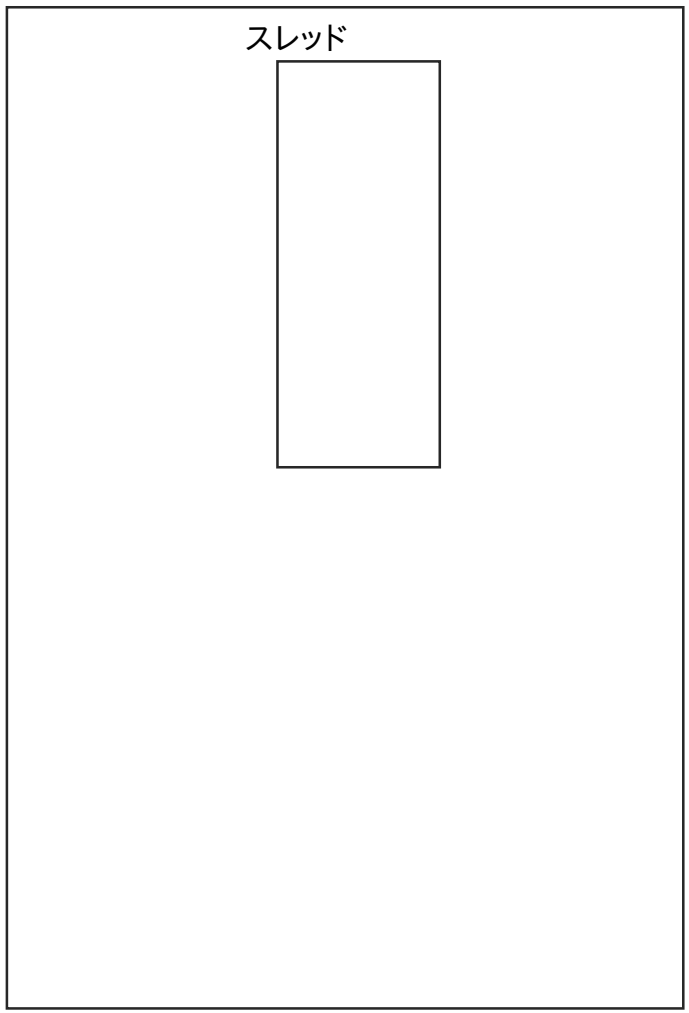
スレッドを観察するコマンド

- Unix CUI
  - `ps auxmww`
  - `top -H` (H で Toggle)
- Linux
  - `/proc/`*pid*`/task/`*tid*

Unix: スレッド関連の API

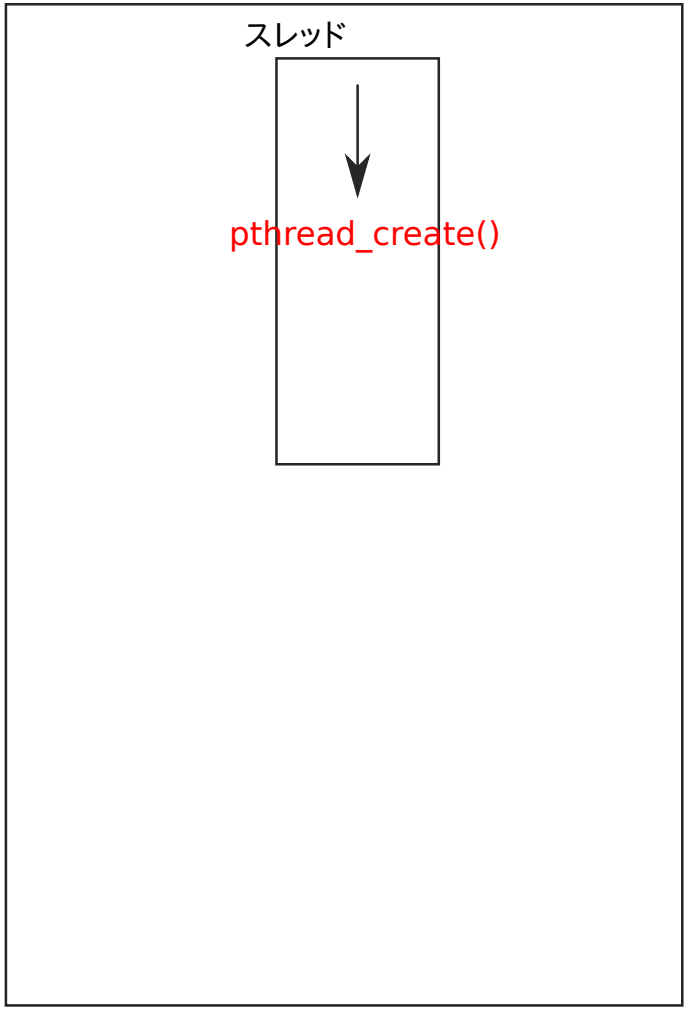
- POSIX スレッド (または単に `Pthreads`)
  - POSIX : Portable Operating System Interface (Unix 系 OS 共通の API 仕様)
- `pthread_create`
  - スレッドを作る
- `pthread_exit`
  - 現スレッドを終了
- `pthread_join`
  - スレッドの終了を待つ
- その他多数 (後の週)

スレッドの生成～終了～処理



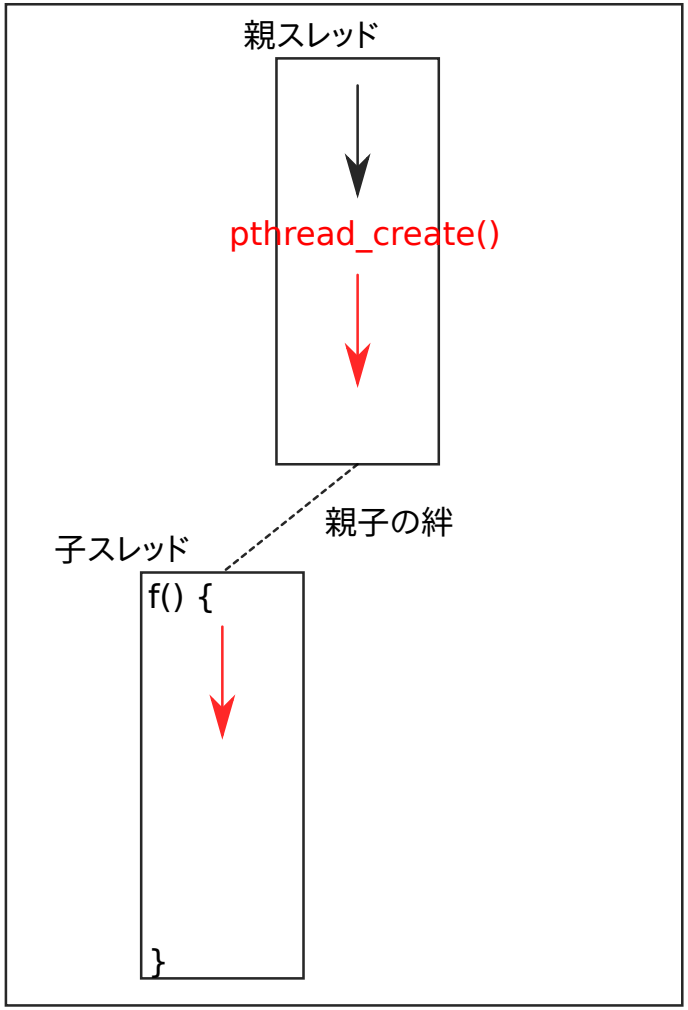
スレッドの生成〜終了〜処理

1. pthread_create により
   子スレッドが生成される



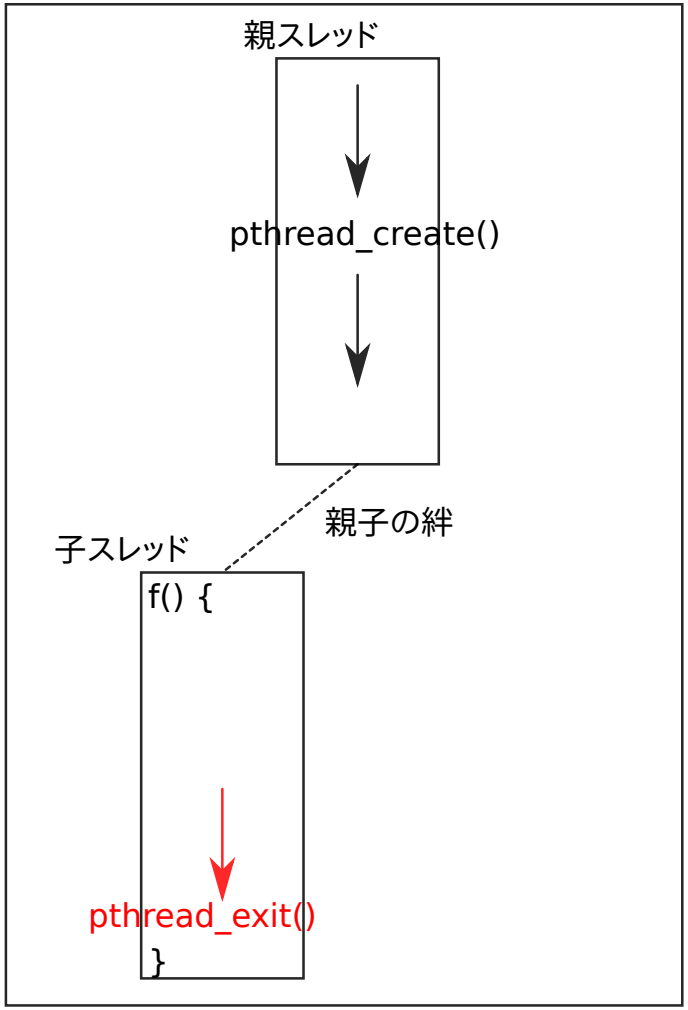
スレッドの生成〜終了〜処理

1. `pthread_create` により
   子スレッドが生成される



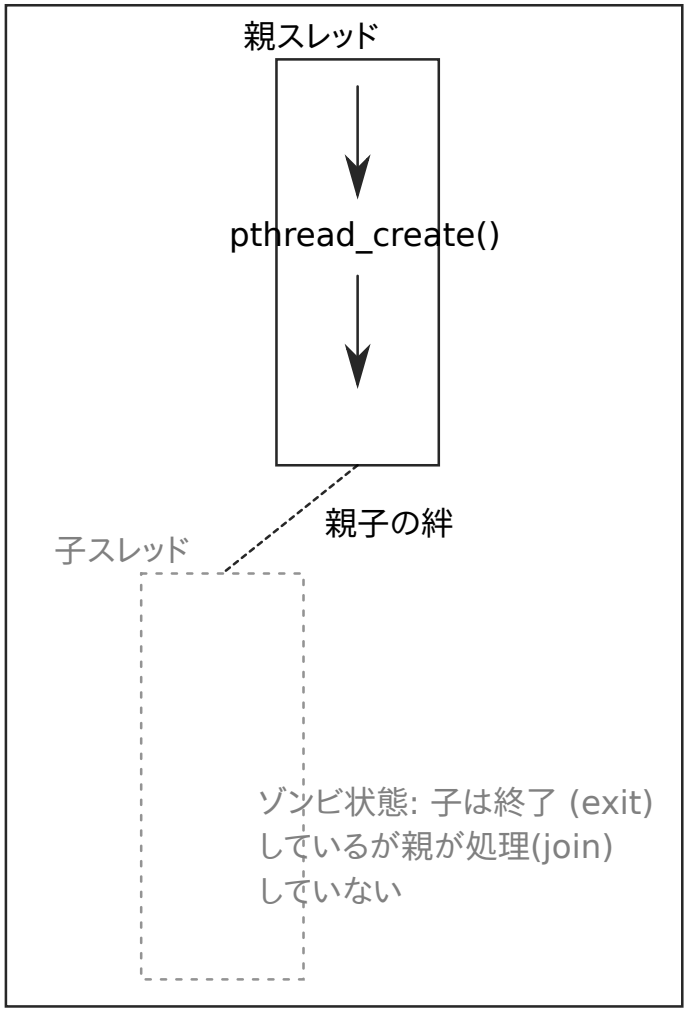
スレッドの生成〜終了〜処理

1. pthread_create により
   子スレッドが生成される
2. 子スレッドが終了する
   (pthread_exit を呼ぶ,
   または子スレッド生成時
   に指定した関数が終了)



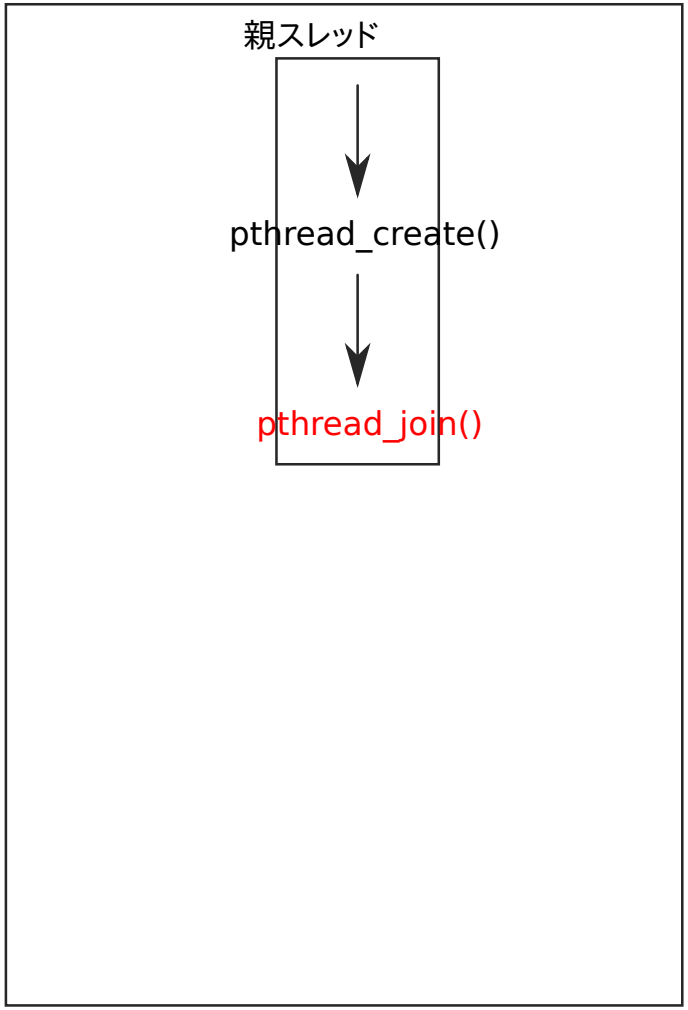
スレッドの生成〜終了〜処理

1. pthread_create により
   子スレッドが生成される
2. 子スレッドが終了する
   (pthread_exit を呼ぶ,
   または子スレッド生成時
   に指定した関数が終了)



スレッドの生成〜終了〜処理

1. pthread_create により
   子スレッドが生成される
2. 子スレッドが終了する
   (pthread_exit を呼ぶ,
   または子スレッド生成時
   に指定した関数が終了)
3. どれかのスレッドが
   pthread_join を呼ぶ



pthread_create

- pthread_create(*tid*, *attr*, *f*, *arg*)
- *f*(*arg*) を実行するスレッド (子スレッド) を作る ≡ pthread_create 呼び出し以降と, *f*(*arg*) が並行して実行される
- *f* は void*を受け取り void*を返す関数
- スレッドの開始関数と呼ぶ (あまり一般的でない用語)
- 諸々の属性を *attr* で指定 (man を見よ)
- pthread_create の return 後, 子スレッドの ID が**tid* に書き込まれる

pthread_exit(p)

- pthread_exit を呼んだプロセスを, 指定した終了ステータス p で終了させる
- p : ポインタ (void *)
- スレッドの開始関数が終了した場合も同じ効果
  - 開始関数の返り値 (return value) が終了ステータス
- p は pthread_join で取得可能

pthread_join

- pthread_join(*tid*, *q*)
- 子スレッド *tid* の終了を待つ
- *tid* の終了ステータスが*q* に返される
- waitpid のスレッド版だが細かい違い:
  - pthread_join(*tid*, *q*) を呼ぶのは, *tid* の親スレッドである必要はない (同じプロセス中のどのスレッドでもよい)

プロセス vs. スレッド

- 同じプロセス内のスレッドは「アドレス空間」を共有する
  - ≈ プログラム内のデータを共有する
  - ≈ あるスレッドが書き込んだ値は他のスレッドも自動的に観測する
- 複数のプロセスはそれぞれ独立した「アドレス空間」を持つ
  - ≈ プログラム内のデータは共有されない
  - ≈ あるプロセスが書き込んだ値が他のプロセスに観測されることはない (その仕組みおよび例外は後の週で)