Linux とスレッド

Debian/Debian JP Project
後藤 正徳
GOTO, Masanori
gotom@debian.or.jp

プログラムと並行動作

▽1つのプログラム中で並行動作を行いたい

▽プロセッサが複数あるとき処理を分散させたい

→ こういうときこそ「スレッド」を活用

並行動作がうれしいときは?

▽Un*x ではプログラムはプロセス毎にわかれて各々並行に動作

▽各プロセスは相互に独立

→ 共有するためには特別な方法が必要

スレッドの基礎概念

スレッド (thread) とは何か?

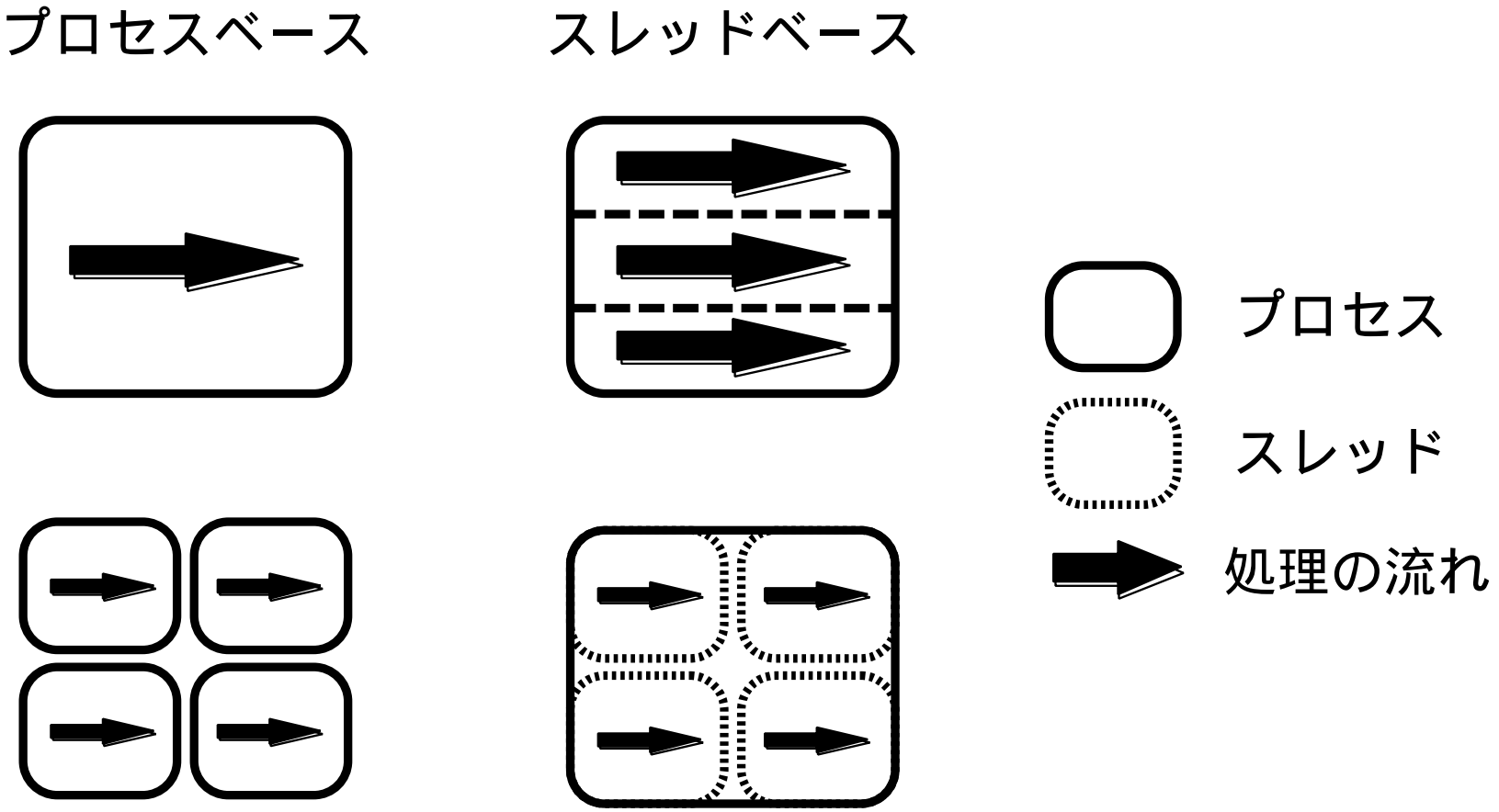
▽並行動作を実現するための仕組み

▽Un*x のプロセスにかなり似ていて、プロセス中の制御を更に分解した単位

▽各制御の流れの単位をスレッドと呼ぶ

プロセスとスレッド

プロセスとスレッドの違い



プロセスベースとスレッドベース

- プロセスベースのモデル
  - プログラムの制御の流れはプロセス中で単一
  - 並行処理の概念は一般に含まれない
  - 複数の制御にわけて動作 → fork() で子プロセス生成
  - プロセス間通信 (IPC) などの手法が必要
- スレッドベースのモデル
  - プログラムの制御の流れはプロセス中で複数
  - 並行処理の概念を本質的に含む
  - 複数の制御 → 各言語・ライブラリの機能で実現
  - プロセス間通信は必要なし

プロセス間通信

- IPC: Inter Process Communication
- プロセス間で通信を行うための仕組み
- いくつかの種類が考えられる
  - SYSV IPC (メッセージキュー・共有メモリ・セマフォ)
  - POSIX IPC (メッセージキュー・共有メモリ・セマフォ)
  - FIFO
  - パイプ
  - ソケット

→ スレッドではこれらの機能を利用しなくてもスレッド間で通信可能

スレッドの利点/欠点

- スレッドの利点
  - ○ 並行処理を比較的簡単に実現できる
  - ○ プロセスの資源を所属する全スレッドで共有可能
  - ○ 非同期I/O
  - ○ コンテキストスイッチの負荷軽減
  - ○ マルチプロセッサを有効に活用
- スレッドの欠点
  - × 並行性・非同期性
  - × 変数の値を予想外に変更される危険性
  - × スレッドセーフなライブラリが必要
  - × シグナルなど一部の既存手法の変更
  - × デバッグが困難

スレッドの利点 (1)

- 並行処理を比較的簡単に実現できる

▽言語にもよるが比較的簡単に並行処理が記述可能

タスクの分割にスレッドを利用しないと...

▽プロセス間通信(IPC)など OS が提供するその他の手法が必要

スレッドの利点 (2)

- プロセスの資源を所属するスレッド全てで共有可能

- プロセス毎に管理されている資源をそのプロセスに属するスレッド全てで共有
  - 具体的にはコンテキスト、レジスタ、スタック、ヒープ、ファイルディスクリプタなど

- 各スレッドで大域変数が共有されている

スレッドの利点 (3)

○ コンテキストスイッチの負荷軽減

コンテキストスイッチ (context switch)

→ プロセスの切り替え

▽コンテキストスイッチで擬似的に複数の仕事を並列実行しているように見せている

▽プロセスのスイッチにかかるコスト (例えばレジスタやスタックの待避) はかなり大きい

▽スレッドでは待避のコストが少ない

▽頻繁なスイッチにも高速に対応できる

スレッドの利点 (4)

- 非同期I/O

▽デバイスと非同期にI/Oが生じる場合有効

▽単一プロセッサでもスレッドによって速度が向上する
  ▷例: bind 9.x

スレッドの利点 (5)

- マルチプロセッサを有効に活用
  - マルチプロセッサに処理を容易に分散可能
  - プロセスベースの場合、システム依存な設計になりやすい
  - 大域変数を共有しているため、プロセッサ間通信を考えなくても良い

スレッドの欠点 (1)

× 並行性・非同期性

▽各スレッドが非同期に動作する

▽各スレッドはバラバラな場所で動作する

→ 非同期性を考慮してプログラムする必要

スレッドの欠点 (2)

× 変数の値を予想外に変更される危険性

▽スレッドでは大域変数が各スレッドで共有されている

▽各スレッドの変更が非同期に行われる可能性

→ mutex を用いて解決可能

▽mutex などは明示的にプログラム中で指定しなければならない

スレッドの欠点 (3)

× スレッドセーフなライブラリが必要

▽ スレッドに対応していないライブラリでは、大域変数を予想外にアクセスする設計になっている可能性がある

▽ スレッドを使うときはスレッドセーフなライブラリが必要

スレッドセーフ(thread safe) → スレッドを使用しても安全

スレッドの欠点 (4)

× シグナルなど一部の既存手法の変更

▽スレッドを用いない場合
→ シグナルは単純に送りたいプロセスに向かって送信すれば良い

▽スレッドを用いる場合
→ どのスレッドがシグナルを受け取るのかなどを考慮する必要がある

スレッドの欠点 (5)

× デバッグが困難

スレッドを使用する → デバッグが困難になる

▽スレッドを使う場合、複数のスレッドが同じプロセス内のデータを書き換えながら動く

- ▷→ いつ/どのスレッドが走っているか知る必要
- ▷→ 再現性が失われる可能性

▽デバッガもスレッドに対応する必要がある

- ▷最近の gdb ではマルチスレッドに対応されつつある

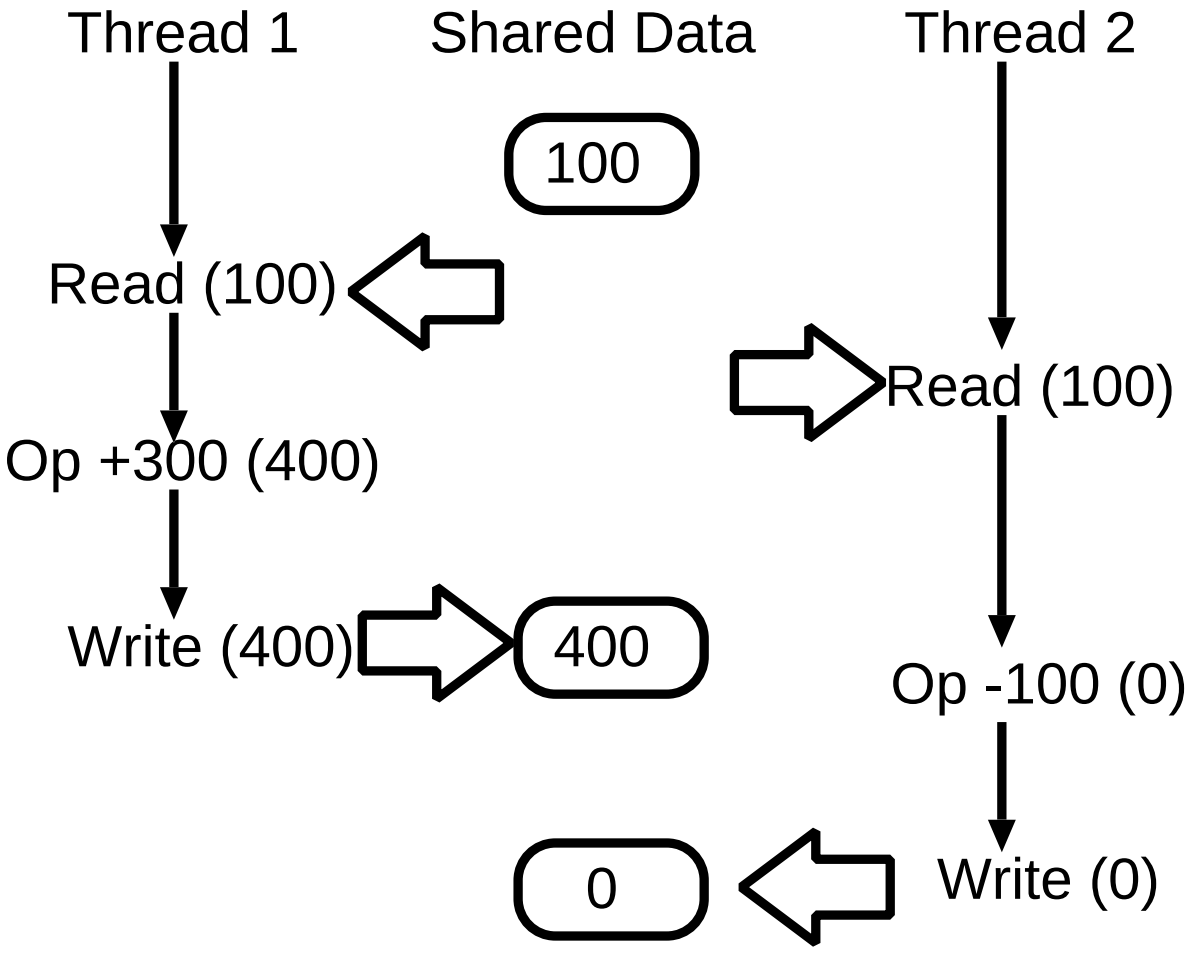
スレッドのライフサイクル

生成 ⇒ (停止 ⇔ 実行 ⇔ 待機) ⇒ 終了 ⇒ 合流

▽プロセスに似たライフサイクル
▽以上の遷移を繰り返す

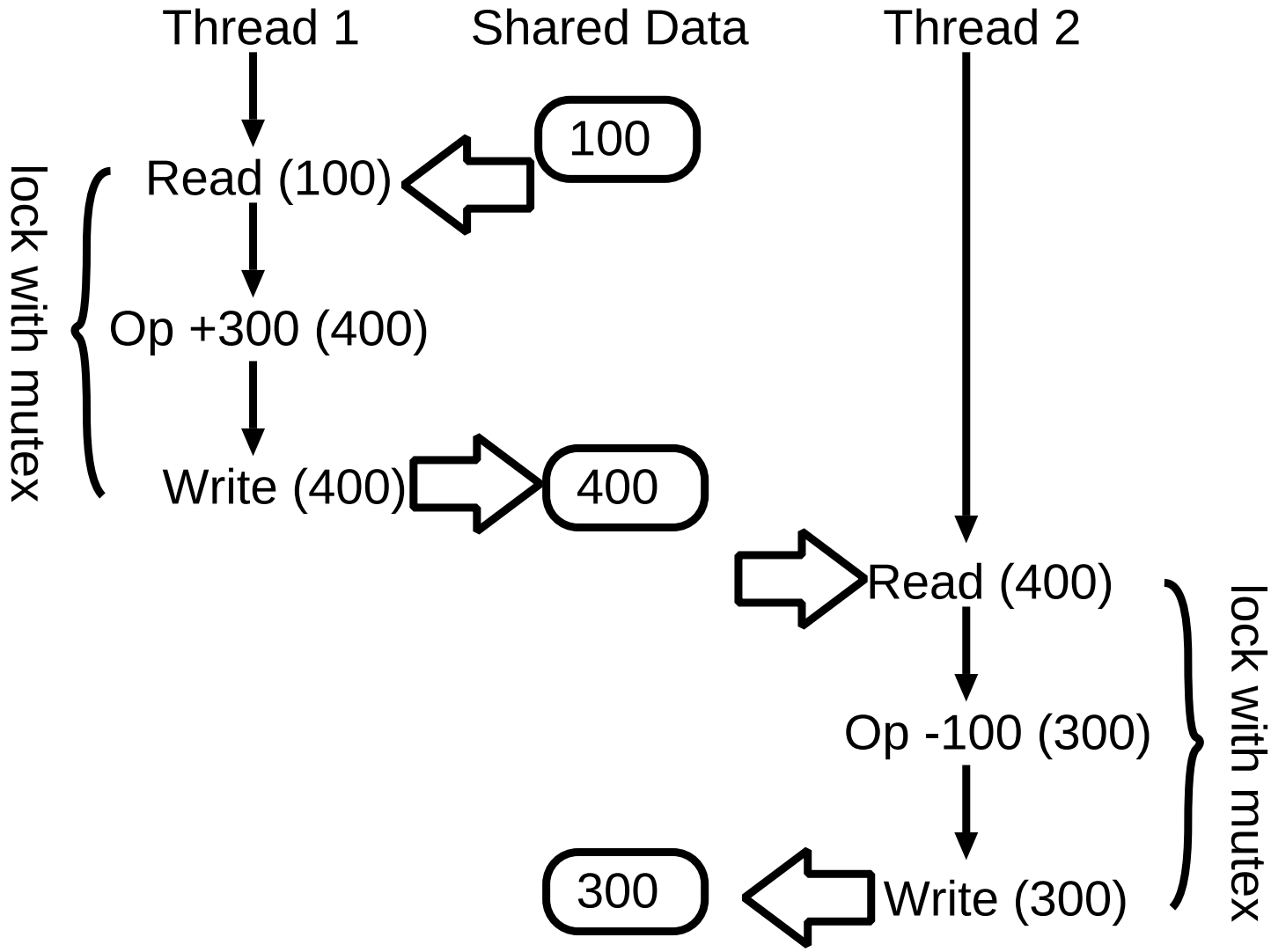
同期 (1)

▽同期がないとき


Thread 1
Shared Data
Thread 2
100
Read (100)
Read (100)
Op +300 (400)
Write (400)
400
Op -100 (0)
0
Write (0)

同期 (2)

▽同期があるとき

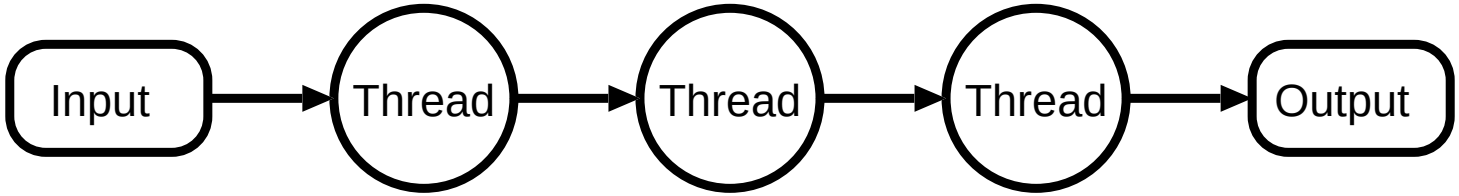

Thread 1
Shared Data
Thread 2
100
Read (100)
lock with mutex
Op +300 (400)
Write (400)
400
Read (400)
Op -100 (300)
lock with mutex
300
Write (300)

スレッドの動作形態 (1)

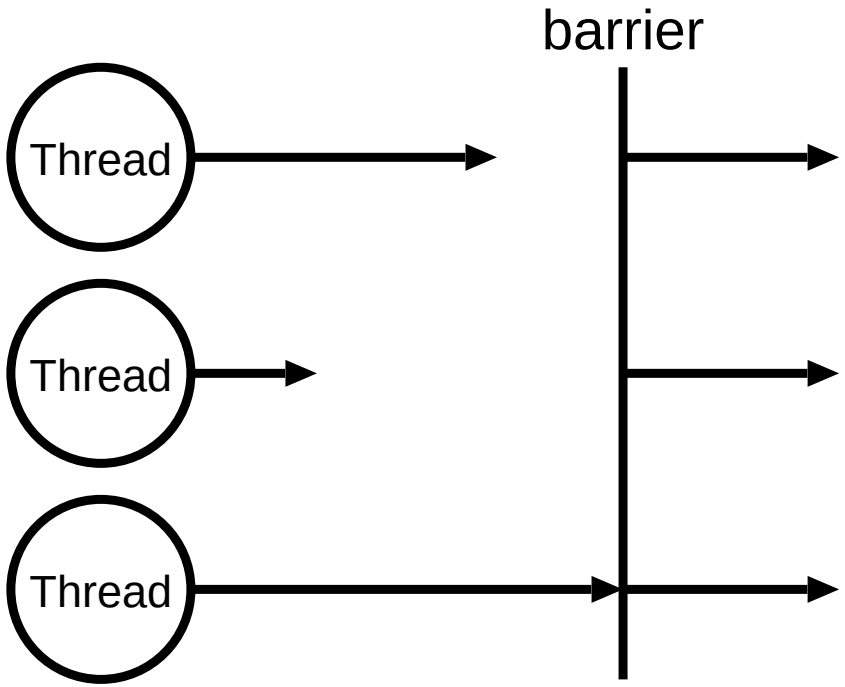
- クライアント/サーバー
- 生産者/消費者
- リード/ライト
- パイプライン
- バリア

スレッドの動作形態 (2)

パイプライン



バリア



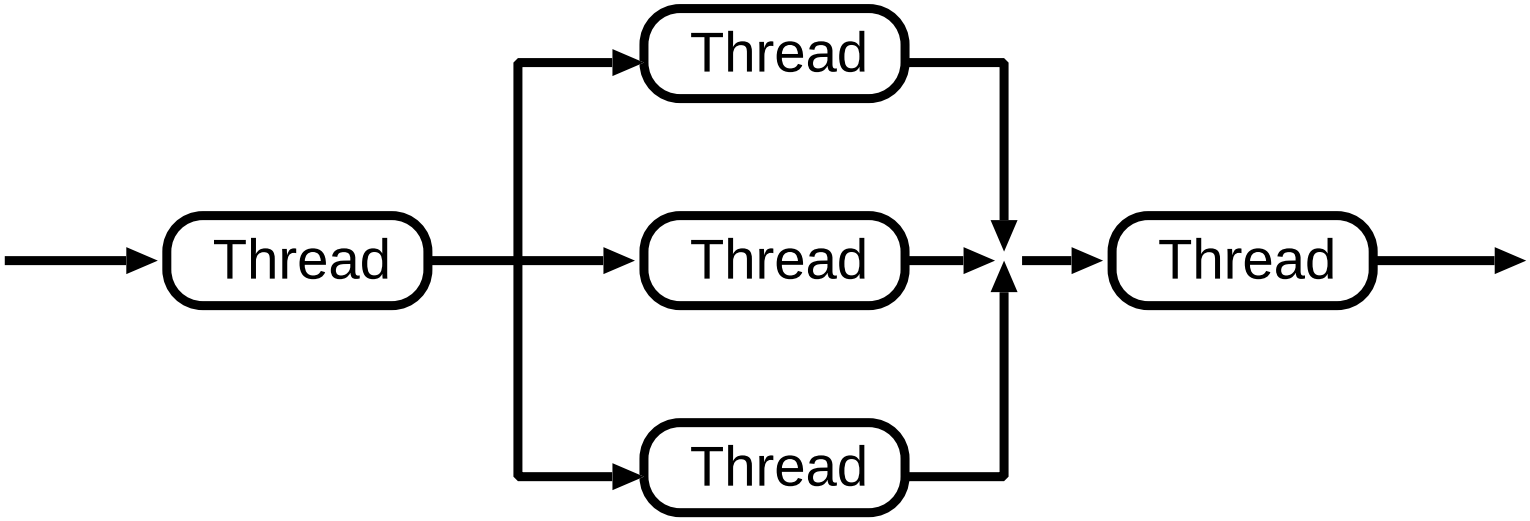
スレッドに向くアプリケーション

▽複数の処理が同時並行に進行するもの

▽同時に複数I/Oを行うもの

スレッドを使うときの注意 (1)

▽並行処理の度合い



スレッドを使うときの注意 (2)

- アムダールの法則
  - 速度の向上 = 1/((1-p)+p/n)
    - p: 並行処理の割合，n: プロセッサ数
- スレッドの無理な多用は全体の速度をかえって低下させる
- 移植性に気を使うならプロセスベースも有効な解

Linux カーネルとスレッド

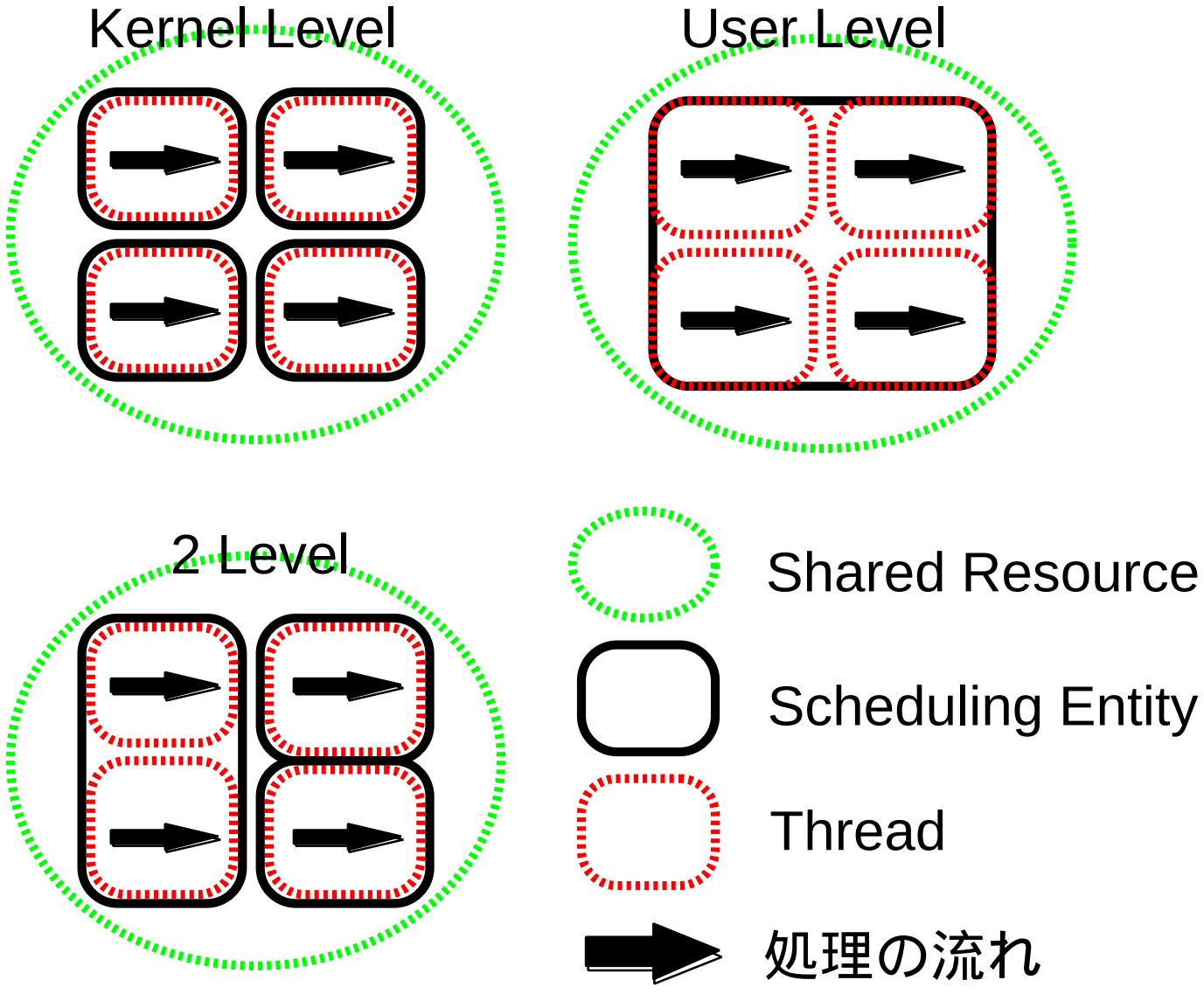
スレッドの実装レベル

同じスレッド実装でも、システムによって速度が全く変ってくる
→ カーネル内のスレッドの表現方法が違うため

カーネル (kernel) におけるスレッド実体の表現
→ 大きく3種類に分けられる

▽ユーザーレベル
▽カーネルレベル
▽2レベル

図: 各実装の違い


Kernel Level
User Level
2 Level
Shared Resource
Scheduling Entity
Thread
処理の流れ

ユーザーレベル (user level)

▽別名: 多対一 (many-to-one)

▽プログラムが全スレッドをスケジューリング

▽カーネルはスレッドが属すプロセスのみ認識

○ 利点

▽スレッド切り替えが高速

▽移植性が良い

× 欠点

▽あるスレッドがブロックされると、スレッド全体がブロックされる

▽マルチプロセッサを活用できない

カーネルレベル (kernel level)

▽別名: 一対一 (one-to-one)

▽カーネルが各スレッドをスケジューリング

○ 利点

▽あるスレッドのブロックが他に影響しない

▽マルチプロセッサに対応

× 欠点

▽スレッドの切り替えが重い

▽スケーリングが悪い

▽ライブラリが OS に依存

2レベル (2 level) (1)

▽別名: 多対多 (many-to-many) / mixed レベル

▽ユーザーレベル・カーネルレベルを組合わせたもの

○ 利点

▽あるスレッドのブロックが他に影響しない

▽マルチプロセッサに対応

▽スケーラビリティが良い

▽ユーザレベルとカーネルレベルが組合わせ可能

2レベル (2 level) (2)

× 欠点

▽ユーザの直接的な制御が難しくなる

▽実装が複雑

▽ライブラリが OS に依存

▽プログラムで並行性レベルを指定する必要がある場合がある

どの実装が良いか?

それぞれのアプリケーションに依存

- ▽カーネルレベル
  - ▷I/O処理を含むアプリケーション
- ▽ユーザーレベル
  - ▷I/O処理を含まないアプリケーション
  - ▷スレッドのスイッチが多発するアプリケーション
- ▽2 レベル
  - ▷この両者を兼ね備えている点で魅力的
  - ▷対応 OS は少ない

→ 実装レベルを考慮に入れることが、性能差に大きく結び付く

Linux カーネルとスレッド

▽Linux カーネルはスレッドに対応しているか?
→ Linux はバージョンによってスレッド対応が異なる

▽2.0 以前は特別な対応はされていない
▽2.0 以降はカーネルスレッド実装が利用可能
→ clone() システムコールによって実現

▽現在も対応・拡張作業が続けられている

スレッド対応の歴史 (1)

カーネル 1.2 まで

▽スレッドは一切考慮されておらず、特別なサポートはなし

▽ユーザーレベル実装のスレッドのみ利用可能

カーネル 2.0 から

▽スレッドのサポートが開始

▽カーネルレベル実装のスレッドが利用可能に

スレッド対応の歴史 (2)

カーネル 2.2 から

▽スレッド用のシグナルが新たに用意

カーネル 2.3 と 2.4

▽/proc/kernel/sys/threads-max の対応

▽SYSV 共有メモリのスレッド対応

▽多量のプロセスが生成できないバグのfix

▽その他、カーネルの内部関数や機能のスレッド対応化

clone() システムコールとは

カーネルスレッドのサポート
→ __clone(2) システムコール (sched.h)

- fork(2) と同様に子プロセスを生成する関数
- fork() と __clone() は同じ実装になっている。
- 以下のパラメータの共有について指定可能
  - CLONE_VM メモリ空間
  - CLONE_FS ファイルシステム情報
  - CLONE_FILES ファイルディスクリプタ
  - CLONE_SIGHAND シグナルハンドラテーブル
  - CLONE_PID PID

clone() システムコールについて

▽カーネルレベル実装である
→ スケジューリングはプロセスと同じ単位

▽CLONE_PID の共有は今のところ未実装
→ スレッド毎に PID も異なる
= 見た目はプロセスと同じ

▽Linux に固有の関数なため、通常ユーザープログラムで使う必要はない

スレッドの最大生成数

スレッド (プロセス) の最大可能生成数の調整

▽/proc/kernel/sys/threads-max で可能

▽2.3 以前はコンパイル時に指定しなければならなかった

Linux と 2 レベル

Linux はカーネル実装 → 2レベル実装ではない

理由:

- ▽Linux の context switching のコストが低いため
- ▽単純な設計ができるため
- ▽Linus 氏を含めた開発者達がカーネルレベルを推しているため

→ 残念ながら2レベルはしばらく利用できそうにない

Pthread

Pthread とは

- 正式名称: POSIX 1003.1c-1995
- C 言語等からスレッドを操作する API を規定した規格
- Un*x でのスレッドライブラリ規格の代表格

Pthread の特徴

- POSIX, ISO/IEC, Unix98 等で標準として規格化
- 特定 OS やシステムに依存しない汎用性を持つ
  - → Pthread をサポートするシステム間での移植が容易
- ライブラリ開発者は OS と Pthread の仕様に沿ってライブラリを実装可能

Pthread の関数群

生成 pthread_create

終了 pthread_exit

合流 pthread_join

同期 pthread_mutex

条件変数 pthread_cond

属性の変更 pthread_ ~ attr ~

Pthread の型

- pthread_t Pthread を表わす
- pthread_mutex_t Pthread mutex 型
- pthread_cond_t Pthread cond 型
- pthread_key_t スレッド固有データキー
- pthread_attr_t 属性
- pthread_mutexattr_t mutex 属性
- pthread_condattr_t 条件変数属性
- pthread_once_t 一度切り初期化

Pthread の例 (1)

▽早速使ってみる
▽最も単純なプログラム例
▽コンパイル時の注意

---------- コンパイル
$ gcc -o test test.c -lpthread -D_REENTRANT

▷-lpthread と _REENTRANT フラグを忘れずに!
▷-D_REENTRANT はリエントラントな関数を使うよう指示する. Pthread を使うときは必ずつけること

Pthread の例 (2)

---------- 実行結果

```
$ ./test
Type some character:
abcde
-- Thread A: abcde
fghij
-- Thread B: fghij
exit A!
exit B!
```

Linux と Pthread

Pthread は Linux で使えるか?
→ 使用可能!

▽Linux で利用可能なスレッドライブラリ
→ 様々な種類のものが提供されている

▽最も代表的なのが LinuxThreads
  ▷大半のディストリビューションで標準的に利用可能

▽他にもあるが、あまり利用されていない

LinuxThreads (1)

- Linux で最も広く利用されている Pthread スレッドライブラリ
- Xavier Leroy 氏によって開発
- 現在は add-on パッケージとして glibc の一部になっている
  - ほぼ標準で利用可能
- glibc LinuxThreads はカーネルレベル実装
  - Linux で唯一のカーネルレベル実装ライブラリ...?

LinuxThreads (2)

- スレッドの実装種類
  - glibc に含まれる前 → ユーザーレベル実装
  - glibc に含まれる後 → カーネルレベル実装
- 各スレッドの他にマネージャスレッドを起動する
  - → ps で見るとプロセス数は (スレッド数+1) になる
- 最近はスレッドデバッガをサポートしたライブラリも含む

LinuxThreads (3)

LinuxThreads でスレッド機能を利用する際の注意

- 出来るだけ最新を使うべき
  - ライブラリ・カーネルは出来るだけ最新を使うことが推奨される
  - 現在も活発にバグフィックスがされているため
- 未実装機能の存在
  - 未実装の機能や関数がいくつかある
  - 例: pthread_mutexattr_setpshared() など

その他の Pthread ライブラリ (1)

- Linux で利用可能なスレッドライブラリは多数存在
- ほぼ全てがユーザーレベル実装
- 以降に紹介するのは、ここ最近もメンテナンスがされているもの

その他の Pthread ライブラリ (2)

PTL

- ▽BSD 系を中心とした OS で動作確認済み
- ▽日本語で多数のドキュメントが提供
- ▽簡易デバッガ (PDB) のサポート
- ▽Linux + glibc では現在動作しない

GNU Pth

- ▽現在も開発中
- ▽Pthread だけでなく abort をサポートするなど意欲的・本格的

その他の **Pthread** ライブラリ **(3)**

MIT Pthreads

▽libc5 システムの dist. などに入っている

▽そこそこ関数はそろっている

▽既に開発は終了. 派生物として pmp (Patched MIT Pthreads) がある

FSU Pthreads

スレッドと言語

スレッドと言語の関係

- 各言語ごとにスレッドの利用方法が異なっている
- 大きく次の3種類に分けられる
  - 1. 言語仕様に組み込まれているもの
  - 2. ライブラリの協力が必要なもの
  - 3. 何も考慮されていないもの
- 言語によってはスレッドとは呼ばないか、似たような概念を提供している場合がある

C/C++

アプリケーション作成時にスレッドを利用する

C:

▽Pthread ライブラリ他、数多くのライブラリが提供されている

C++:

▽C のライブラリを流用可能

▽動作確認がとれていない場合もある

Java

Java

- ▽言語仕様としてスレッドをサポート
- ▽比較的簡単に利用できる
- ▽blackdown 版 JDK のバージョン 1.1.7 から Linux のネイティブスレッド(native thread) に対応
- ▽グリーンスレッド(green thread)版ではユーザーレベル実装のスレッドが利用可能

Perl/Ruby

Perl

▽Perl 5.6 よりユーザーレベル実装が利用可能

Ruby http://www.ruby-lang.org/

▽ユーザーレベル実装が利用可能

▽使い方が比較的簡単

その他

Ada

- 言語仕様にスレッドが含まれている
- LinuxThreads を利用して、カーネルレベルで利用可能にしているものがある

並列言語など

- 並列言語の多くが言語レベルでスレッドに類似の機能をサポート
- カーネルレベル実装として Linux で利用可能なものは少ない

スレッドに関する情報 (1): 規格

- POSIX Pthread
  - POSIX 1003.1c-1995 など
- Unix98/SUSv2
  - http://www.opengroup.org/
- ISO/IEC
  - ISO/IEC 9945-1:1996

スレッドに関する情報 (2):library

- LinuxThreads
  - http://pauillac.inria.fr/~xleroy/linuxthreads/index.html
- glibc
  - http://www.gnu.org/software/libc/
- PTL
  - http://www.media.osaka-cu.ac.jp/~k-abe/PTL/index.html
- GNU Pth
  - http://www.gnu.org/software/pth/
- PMP
  - http://www.informatik.hu-berlin.de/~mueller/pthreads/
- FSU Pthreads
  - http://www.cs.fsu.edu/~mueller/pthreads/

スレッドに関する情報 (3): その他

- スレッドライブラリリスト
  - http://www.gnu.org/software/pth/related.html
- Thread Information
  - http://www.media.osaka-cu.ac.jp/~k-abe/PTL/thread-info-ja.html
- comp.programming.threads
  - http://www.serpentine.com/~bos/threads-faq/
- Linux Threads FAQ
  - http://linas.org/linux/threads-faq.html (若干古い)

Q&A