オペレーティングシステム

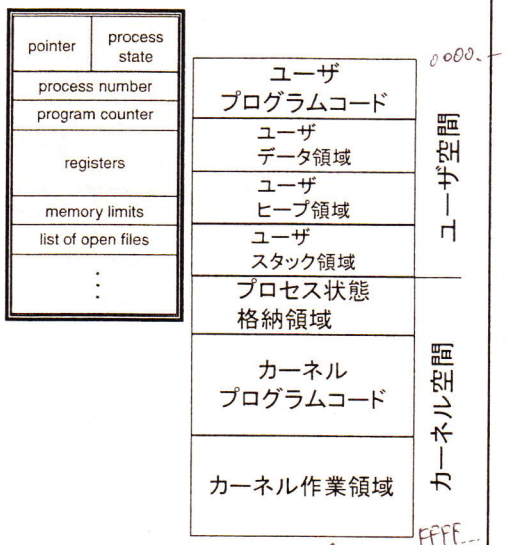
教授　石川　裕
理学部7号館　505
Ishikawa@is.s.u-tokyo.ac.jp

講義スケジュール (予定)

1. OSの概要(4/13)
2. プロセス管理(4/20)
3. プロセス間交信、スレッド(4/27)
4. プロセス同期、シグナル(5/11)
5. CPUスケジューリング 1 (5/18)
6. CPUスケジューリング 2 & トランザクション処理&メモリ管理1 (5/25)
7. 休講 (6/1)
8. 中間試験 (6/8)
9. メモリ管理2 (6/15)
10. メモリ管理3 (6/22)
11. I/Oシステム (6/29)
12. ファイルシステム (6/30)
13. プロテクション＆セキュリティ (7/6)
14. 分散システム&まとめ (7/13)
15. 試験 (7/20)

プロセス管理

- プロセスとは？
  - プログラムの実行単位
  - プロセスが必要とする資源
    - CPU時間
    - メモリ
    - ファイル
    - I/Oデバイス
- OSのプロセス管理
  - プロセス生成・削除
  - プロセス停止・再開
  - プロセス間同期
  - プロセス間通信
  - デッドロック処理

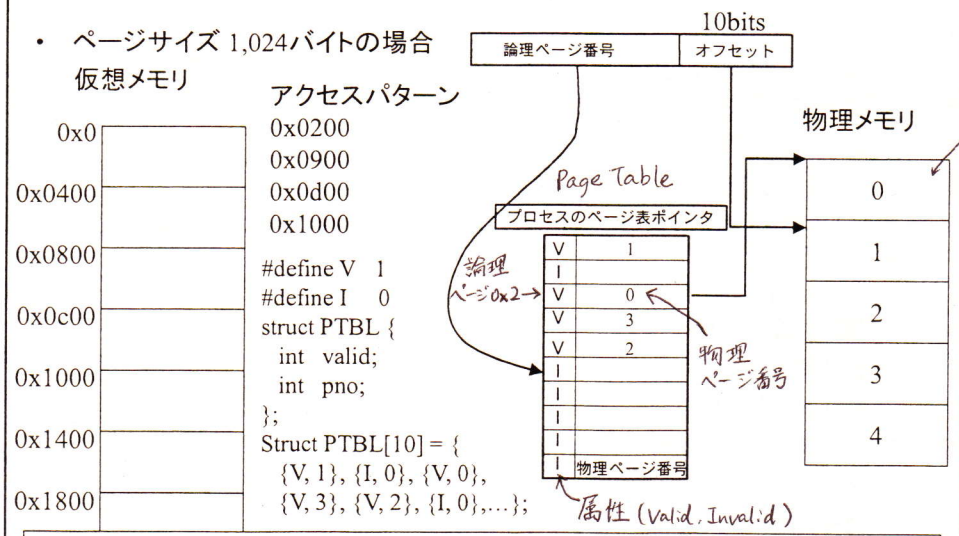


メモリ空間
1プロセスに1アドレス空間
(メモリ空間)

1プロセス1アドレス空間を実現する仕組み

仮想記憶の仕組み

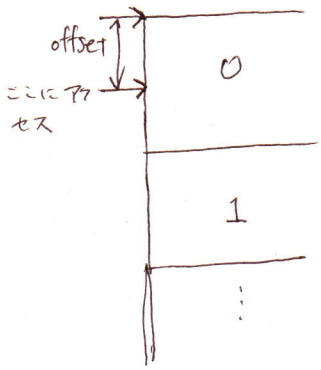
- ページサイズ 1,024バイトの場合

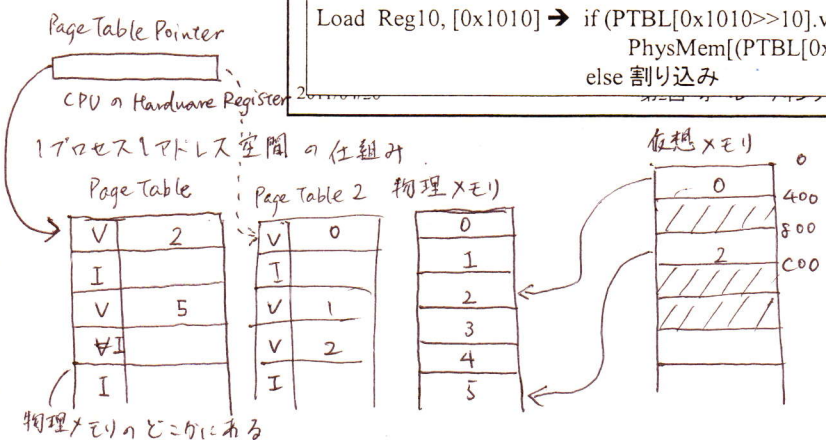



```
Load Reg10, [0x1010] ➔ if (PTBL[0x1010>>10].valid)
                          PhysMem[(PTBL[0x~~1000~~>>10].pno<<10)+0x1010&0x3ff]
                       else 割り込み
```

1010

メモリアドレス
0x0810
=
0 0001 000 0001 0000
論理ページ番号 = 0x2
オフセット 10bit = 0x10



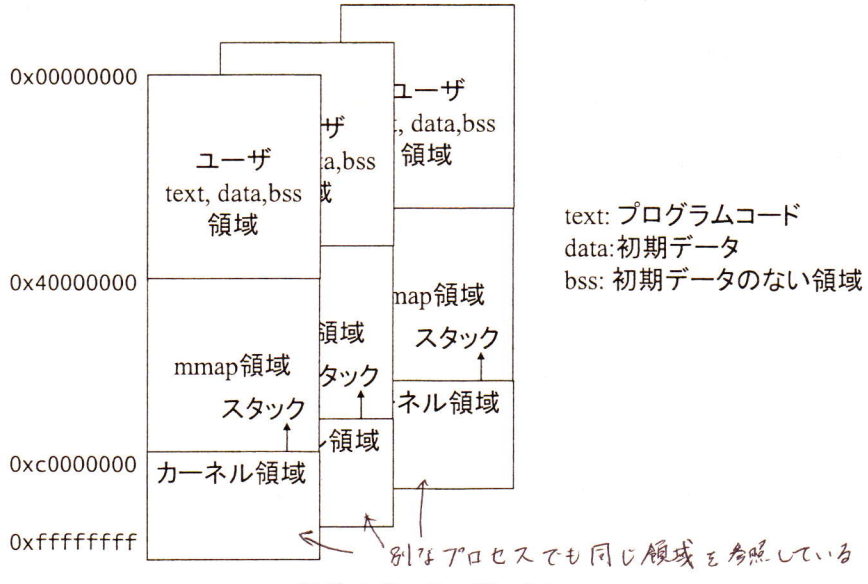


他のプロセスに切り替わるときには,
Page Table Pointer が~~母~~
別の Page Table を参照するようになる.

どんなプロセスが動いている？

- Linuxの場合
  - topコマンド
  - psコマンド
    - ps axwwwl
- Windowsの場合
  - Ctl+Alt+Delして、タスクマネジャーを選択する

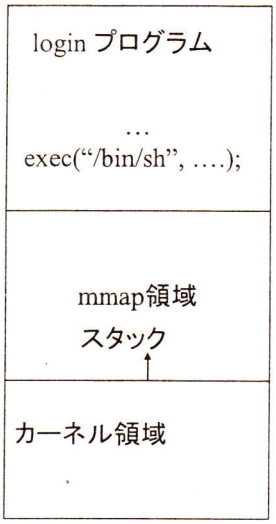
x86系Linuxの仮想メモリ空間使用



プロセス

loginプロセス

メモリイメージ



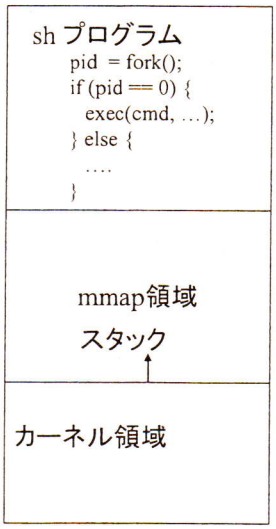
execシステムコールが実行されると、、

切り替わる

プロセス

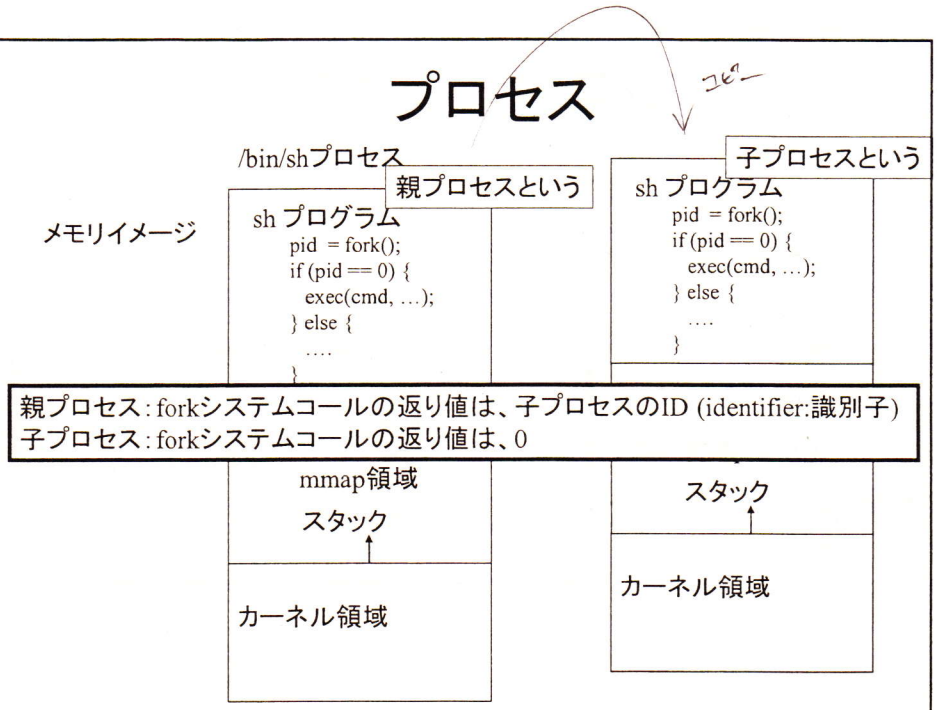
/bin/shプロセス

メモリイメージ

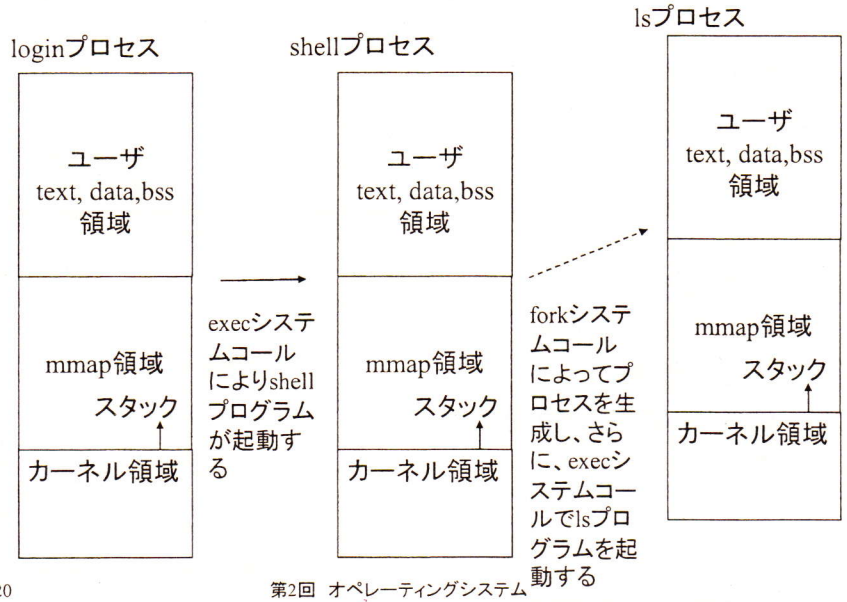


forkシステムコールが実行されると、、

プロセス


コピー
/bin/shプロセス
親プロセスという
子プロセスという
メモリイメージ
sh プログラム
pid = fork();
if (pid == 0) {
exec(cmd, …);
} else {
….
}
sh プログラム
pid = fork();
if (pid == 0) {
exec(cmd, …);
} else {
….
}
親プロセス：forkシステムコールの返り値は、子プロセスのID (identifier:識別子)
子プロセス：forkシステムコールの返り値は、0
mmap領域
スタック
カーネル領域
スタック
カーネル領域

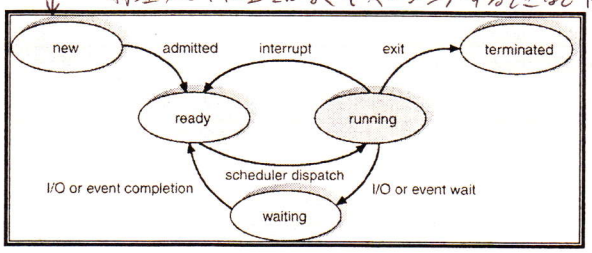
プロセス


loginプロセス
shellプロセス
lsプロセス
ユーザ
text, data,bss
領域
mmap領域
スタック
カーネル領域
execシステムコールによりshellプログラムが起動する
ユーザ
text, data,bss
領域
mmap領域
スタック
カーネル領域
forkシステムコールによってプロセスを生成し、さらに、execシステムコールでlsプログラムを起動する
ユーザ
text, data,bss
領域
mmap領域
スタック
カーネル領域

プロセスの状態

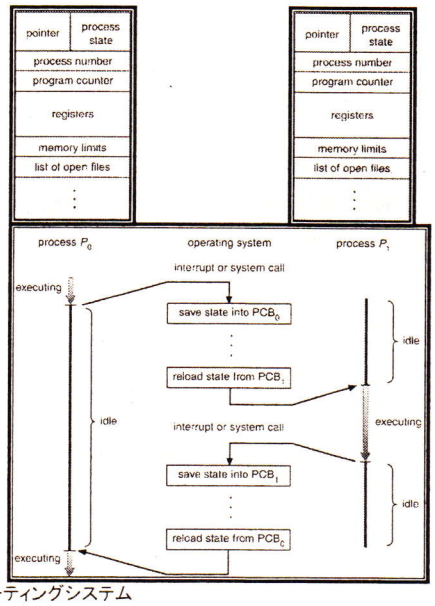
- New
  - プロセス生成中
- Running
  - 実行中
- Waiting
  - 何かの事象待ち
- Ready
  - 実行可能.
- Terminated
  - 実行終了

fork() プロセスを生成するときに
物理メモリに空きがなくてページングするときなどに他のプロセスに切り替わることがある.
そのため, 生成中の状態が必要.

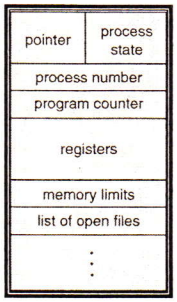


Process Control Block (PCB)

- プロセス毎に必要な情報を格納
  - プロセスの状態
  - プログラムカウンタ
  - CPUレジスタ
  - CPU スケジューリング情報
    - プライオリティ等
  - メモリ管理情報
    - ページテーブル等
  - アカウント情報
    - CPU時間やI/O使用時間等
  - I/O状態



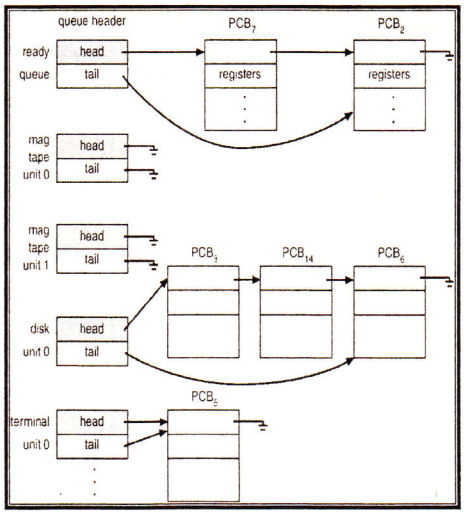
コンテキストスイッチ



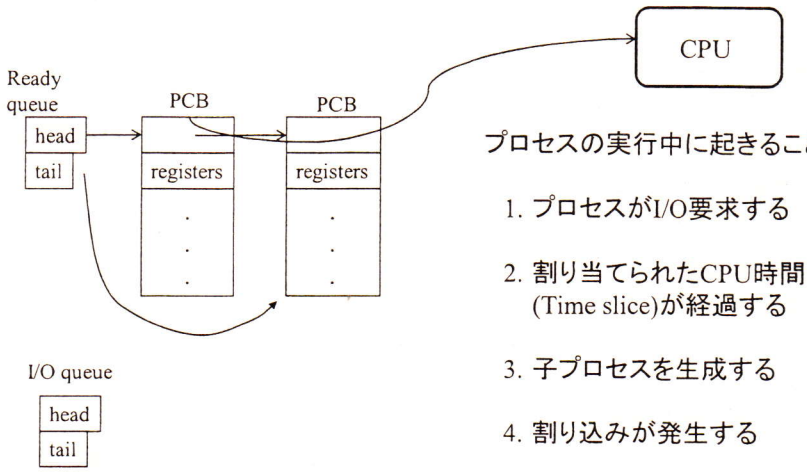
- 実行中のプロセスから他のプロセスに実行を移すためには、
  - 実行中のプロセスの状態を保存し
  - 次に実行するプロセスの状態を取り出す
- この仕事をコンテキストスイッチと呼ぶ
- プロセスの状態は、PCBに保存する
- 一般にコンテキストスイッチは時間がかかる
  - 一般に高速化手法のひとつに
    - 処理を先延ばしすると、その処理をやらなくても良くなる場合がある
  - コンテキストスイッチ高速化手法のひとつ
    - 浮動小数点演算器を使わないプロセスが多いという事実
    - コンテキストスイッチの時には浮動小数点用レジスタの退避、復帰をおこわない
    - 浮動小数点演算が使用されるときに割り込みがかかる

プロセススケジューリングキュー

- Job queue – 全プロセスリスト
- Ready queue – 実行可能状態のプロセスリスト
- Device queues – I/O待ち状態プロセスリスト



プロセスの実行とキュー



プロセスの実行中に起きること

1. プロセスがI/O要求する
2. 割り当てられたCPU時間 (Time slice)が経過する
3. 子プロセスを生成する
4. 割り込みが発生する

スケジューラの種類

- 長期スケジューラ (or job scheduler)
  - バッチシステムでは、ジョブ(プロセス)が投入されると、それらは2次記憶に置かれる(spool)
  - スプールされたジョブの中からいくつかのジョブを取り出して実行可能キューに入れる
- 短期スケジューラ (or CPU scheduler)
  - どのプロセスを次に実行するか

短期スケジューラ

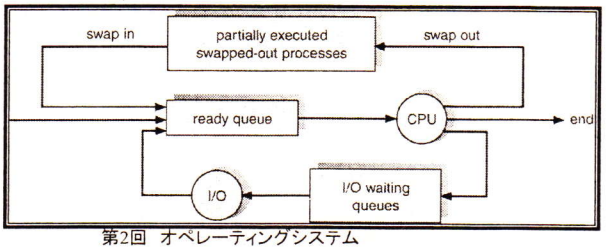
- 短期スケジューラ
  - プロセスが待ち状態になるときに、次に実行すべきプロセスを決めるために起動される
  - プロセスが実行を続けている場合でも、100ミリ秒毎に起動される
  - 処理が早いこと
    - 100ミリ秒毎に10ミリ秒もかかると、~~10(100+10)~~ 10/(100+10) = 9%がオーバヘッドになってしまう。普通は数十から数百マイクロ秒のオーダ

長期スケジューラ

- 秒あるいは分毎に実行される
- マルチプログラミングの度合いを制御
  - 同時に実行するプロセスの数を決める
  - プロセス数を一定にするならば、長期スケジューラはプロセスが消滅したときにのみ実行される
- 遅くても良い
- 長期スケジューラはプロセスの性質に応じて実行するプロセス群を決めるべき
  - I/O-bound プロセス
    - I/O処理が多く、CPUはあまり使わない
  - CPU-bound プロセス
    - CPUを沢山使う

TSSにおけるスケジューラ

- TSS(Time Sharing System)では長期スケジューラは存在しない
- 長期スケジューラの替わりに中期スケジューラがある場合がある
  - マルチプログラミングの度合いを減らすために実行可能プロセスを実行可能キューからはずす(swap out)



プロセス生成一般論(1/2)

- プロセスはプロセスを生成することができる
  - 親プロセス一生成側
  - 子プロセス一作られた側
  - 子プロセスはさらにその子プロセスを生成できる
  - プロセスツリーが形成される
- 資源の共有可能性
  - 親と子供は全ての資源を共有する
  - 子プロセスは親プロセスの一部の資源を共有する
  - 親子で共有する資源はない
- 実行関係
  - 親と子供は同時に実行される
  - 親プロセスは子プロセスの終了を待つ

プロセス生成 一般論(2/2)

- メモリ空間
  - 子プロセスは親プロセスのメモリ空間の複製を持つ
  - 子プロセスはプログラムを読み込み実行する
- UNIX系の例
  - forkシステムコール
    - 子プロセスは親プロセスのメモリ空間の複製を持つ
      - 実現上は共有＆copy on write
    - 子プロセスは親プロセスのファイルディスクリプタの複製を持つ
  - execシステムコール
    - プログラムをファイルから読み出し実行

プロセスの終了

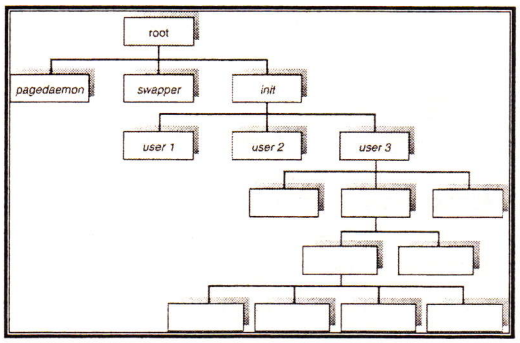
- プロセスは最後にexitシステムコールをして、プロセスを終了する
  - Exitシステムコール時に終了コードを渡すことができる
  - 親プロセスは、子プロセスの終了をwaitシステムコールで待つことができ、子プロセスの終了コードを得ることができる
- プロセス終了時にすること
  - プロセスが保持している資源の解放：メモリ領域、I/Oバッファなど
- 親プロセスはabortシステムコール(Unix系の場合kill)で子プロセスを終了させることができる
  - 親プロセスが子プロセスを終了させる理由
    - 子プロセスが確保した資源を使い切った
    - 子プロセスを必要としなくなった
- 親プロセスは子プロセスの終了を待つことが出来る
  - Linuxの場合、waitシステムコール
- 親プロセスが終了すると子プロセスも終了させられるシステムもある
  - Cascading terminationと呼ばれる
  - Linuxの場合は？
    - 親プロセスが終了すると子プロセスは、initプロセスを親とする

Unix系：プロセスグループ

- 複数のプロセスをまとめて管理
  - 端末のアクセス制御
  - シグナル
- プロセスは親プロセスと同じプロセスグループに属する
- プロセスグループを変更する命令がある

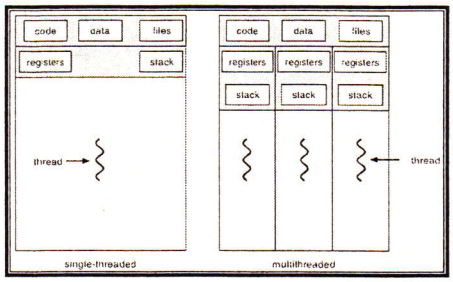
UNIX におけるプロセスツリー

- カーネルをメモリに読み込み実行
- カーネルの初期化が終了すると、カーネルは/sbin/initプログラムを探し、プロセスを起動する
- initプロセスは以下の処理を行う
  - /etc/rc.d/rc.sysinitスクリプトの実行
  - /etc/inittabスクリプトの実行
  - /etc/rc.d/rc.localスクリプトの実行



スレッドとは？

- Thread of Control
  - プロセス内での一連の実行の流れ
  - 単にスレッドと呼ばれている
- プロセス内に複数スレッドを動かすような実行モデルをマルチスレッドモデルと呼ぶ
- マルチスレッドモデルは共有メモリ型並列コンピュータが一般に使えるようになった1970年後半から1980年代に多くの研究がなされた

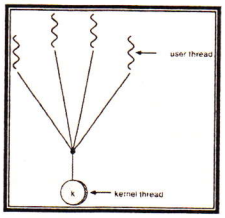


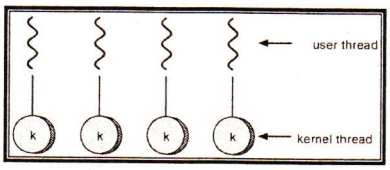
スレッドの実現方法

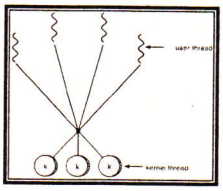
- ユーザレベルスレッド
  - ユーザライブラリで実現
    - setjmp, longjmpを使って実現
      - Linux/Unixのマニュアル参照
    - スケジューリングもユーザレベル
  - スレッドがI/O要求してwaitすると、他のユーザレベルスレッドも実行できなくなる
    - スレッドスケジューリングはユーザレベルで行っている
    - カーネルから見ると一つのスレッドを実行しているだけ
- カーネルレベルスレッド
  - オペレーティングシステムカーネルで実現

マルチスレッド実現モデル

- Many-to-One
  - 複数ユーザレベルスレッドをカーネルで実現されているスレッドひとつにマッピングする
- One-to-One
  - ひとつひとつのユーザスレッドにカーネルで実現されているスレッドが対応している
- Many-to-Many
  - 複数ユーザレベルスレッドを複数カーネルレベルスレッドにマッピング







Linux Threads

- POSIX Pthreadが提供されている
  - スレッドライブラリの標準規格 (IEEE 1003.1c)
  - 多くのUnix系システムで実装されている
- Pthread実現方法
  - clone() システムコール

マルチスレッドの利点

- 並列コンピュータ上でのプロセッサの有効利用
  - 共有メモリ型並列コンピュータ上でプロセッサを有効利用できる
  - マルチコアの時代に突入
    - 半導体集積度が向上、でもクロックを上げられない(リーク電流＆熱問題)
    - 余った回路の使い道
      - チップ内に複数プロセッサを搭載
- 資源の共有
  - ファイルディスクリプタの共有やメモリ領域の共有ができる
- 応答性
  - 例えばマルチスレッドWEBブラウザでは、イメージを読み込んでいるスレッドとユーザの入力を同時に処理できる。
    - シングルスレッドの場合は、selectを使ったevent driven処理になる

協調プロセス

- 独立プロセス同士はお互い影響しあわない
- 協調プロセスはプロセス同士で影響しあう
- 協調プロセスの利点
  - 性能向上
    - ある仕事を複数のサブタスクにわけて並列実行
      - ワーカー、パイプライン、データ並列
  - モジュラリティ
    - 機能毎にプロセスが作られていれば、それらを組み合わせて複雑な処理が実現可能
    - 以下は、shellを使ったパイプライン処理の一例
      - ps axwwl | grep pts/1 | less

プロセスのリストをとりだし、pts/1デバイスを使用しているプロセスを探し、lessで表示

Shellのパイプライン処理、リダイレクションとは？
$ man bash
参照