プロセス

分散システム
2013年11月25日
建部修見

プロセス

- 独立したプログラムの実行
  - 互いに影響を及ぼさない（同一CPUを透明に利用，並行透明性）
  - 独立した論理アドレス空間をもつ
- プロセステーブル

CPUレジスタ
メモリマップ
オープンファイル
アカウント情報
権限（privileges）

プロセス生成とプロセス切替

- プロセス生成
  - 論理アドレス空間の生成
    - メモリセグメントの初期化（データセグメントをゼロクリア，テキストセグメントにプログラムをロード，スタック領域の設定）
- プロセス切替
  - CPUコンテキストの保存
    - レジスタ，プログラムカウンタ，スタックポインタなど
  - メモリ管理ユニット（MMU）のレジスタの修正とtranslation lookaside buffer（TLB）の無効化
  - （メモリ不足の場合）プロセスのスワップイン

スレッドとは？

- プロセス内の＊独立した＊プログラム実行
  - 同一プロセスID
- プロセスほど高い透明性を持たない
  - 論理メモリ空間を共有
  - ファイルディスクリプタなどプロセス資源を共有
- 一般にスレッド生成はプロセス生成より軽い

スレッド切替

- スレッドコンテキスト
  - CPUコンテキスト
    - レジスタ，プログラムカウンタ，スタックポインタなど
  - スレッド管理情報
- ほかの情報は持たない
  - 同一プロセスのスレッド間のデータ保護は開発者に任せられる
- マルチスレッドプログラムの性能はシングルスレッドに比べ悪くならない
  - 多くの場合で性能向上が見込まれる
- スレッドは保護されない
  - 適切な設計とKISS（Keep it simple, stupid）が有効

プロセスvsスレッド

	スレッド	プロセス
生成、実行オーバヘッド	小	大
メモリ	共有	別々
プロセス資源	共有	別々
データ共有	メモリのポインタ渡し（コピーは不必要）	パイプ、ソケット、ファイルなど
保護	他スレッドのメモリ、資源を破壊する可能性あり。スレッドの実行制御が必要	他プロセスのメモリ、資源は保護される

非分散システムにおけるスレッドの利用

- プログラム構造の単純化
  - 表計算における入力処理（スレッド１）と合計値などの値更新（スレッド２）

  スレッド１は入力を処理、スレッド２は変更を待ち変更があれば更新
- マルチプロセッサでは並列に実行可能
- プロセス間通信のオーバヘッド削減
  - （名前付）パイプ，メッセージキュー，共有メモリセグメント
  - カーネル経由のためコンテキスト切替のコスト大

スライドショウのプログラム例

```
main(int argc, char *argv[]) {
    for (i = 1; i < argc; ++i) {
        if (i == 1)
            /* get the first picture */
            buf = get_next_pic(argv[i]);
        else
            /* wait for the previous process */
            pthread_join(tid, &buf);
        if (i < argc – 1)
            /* get the next picture */
            pthread_create(&tid, NULL,
                    get_next_pic, argv[i + 1]);
        display_buf(buf); /* display the picture */
        free(buf);
        if (getchar() == EOF)
            break;
    }
}
```

表示している合間に次の画像を読込

スレッドの実装

- スレッドの生成と解放，mutexロック，条件変数
- ユーザレベルスレッド
  - 軽い（カーネル経由ではない）
  - 入力待ちや無限ループでブロックするとプロセス全体がブロックしてしまう
- カーネルレベルスレッド
  - スレッドスケジューリングはカーネルが行うため、ブロックしても大丈夫
  - スレッド操作のコスト大（システムコール）
- ハイブリッド
  - ユーザレベルスレッドとカーネルレベルLightweight processes（LWP）

並列性の制御の必要性

- スレッド間でメモリ、プロセス資源を共有しているため、破壊してしまう可能性がある
- 例：共有カウンタのインクリメント

counter++ →

1. counterの値をレジスタにロード
2. レジスタの値をインクリメント
3. レジスタの値をcounterにストア

スレッド1

1. counterの値をレジスタにロード
2. レジスタの値をインクリメント
3. レジスタの値をcounterにストア

スレッド2

1. counterの値をレジスタにロード
2. レジスタの値をインクリメント
3. レジスタの値をcounterにストア

共有カウンタのインクリメントの例

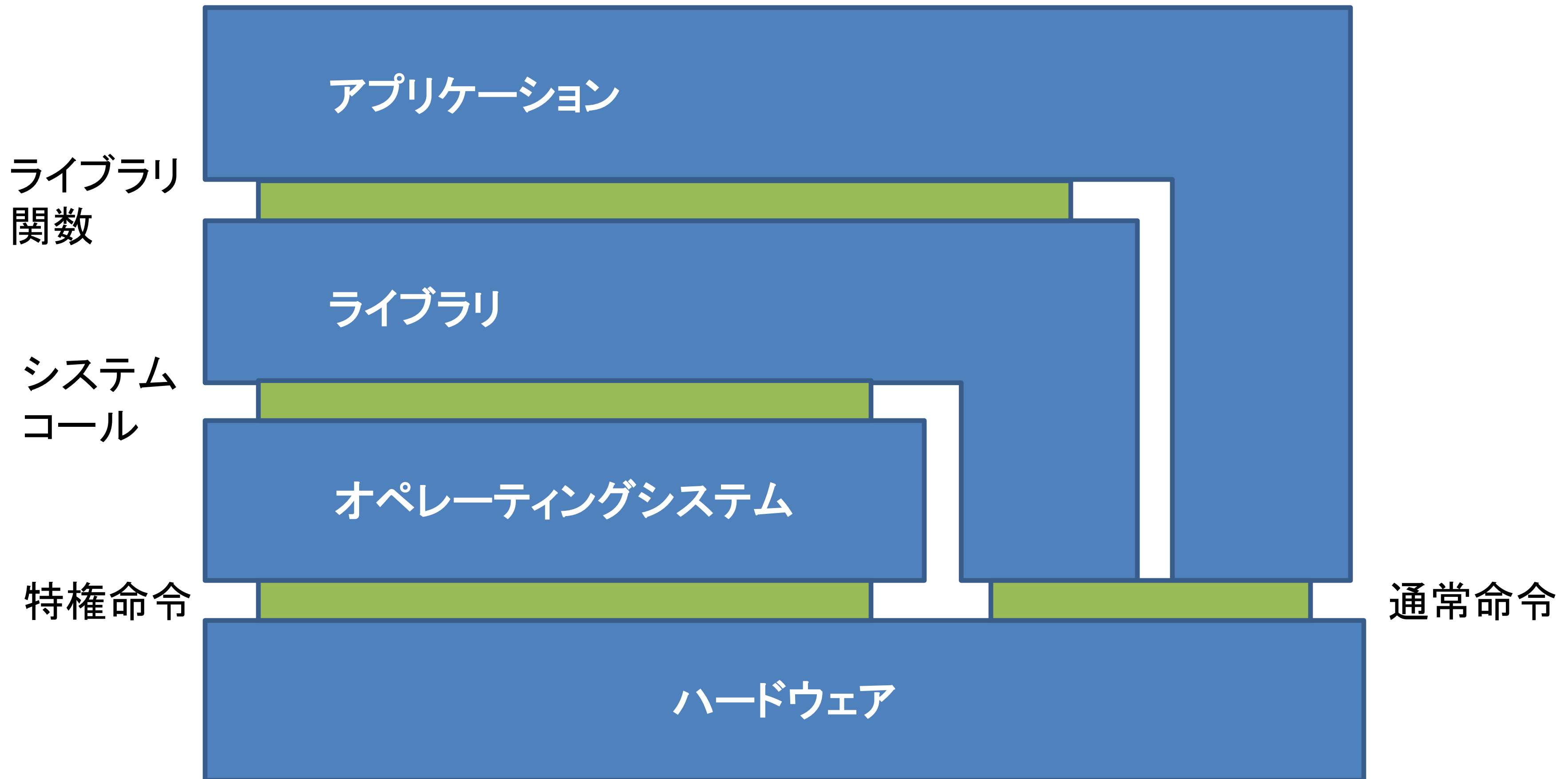
```
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
int count;

void increment_count() {
    pthread_mutex_lock(&count_mutex);      // ensure atomic increment
    count++;                               データ競合領域
    pthread_mutex_unlock(&count_mutex);
}

int get_count() {
    int c;
    pthread_mutex_lock(&count_mutex);      // guarantee memory is synchronized
    c = count;                             データ競合領域
    pthread_mutex_unlock(&count_mutex);
    return (c);
}
```

仮想化

- （異なるシステムのインターフェースをもち）異なるシステムのように振る舞う

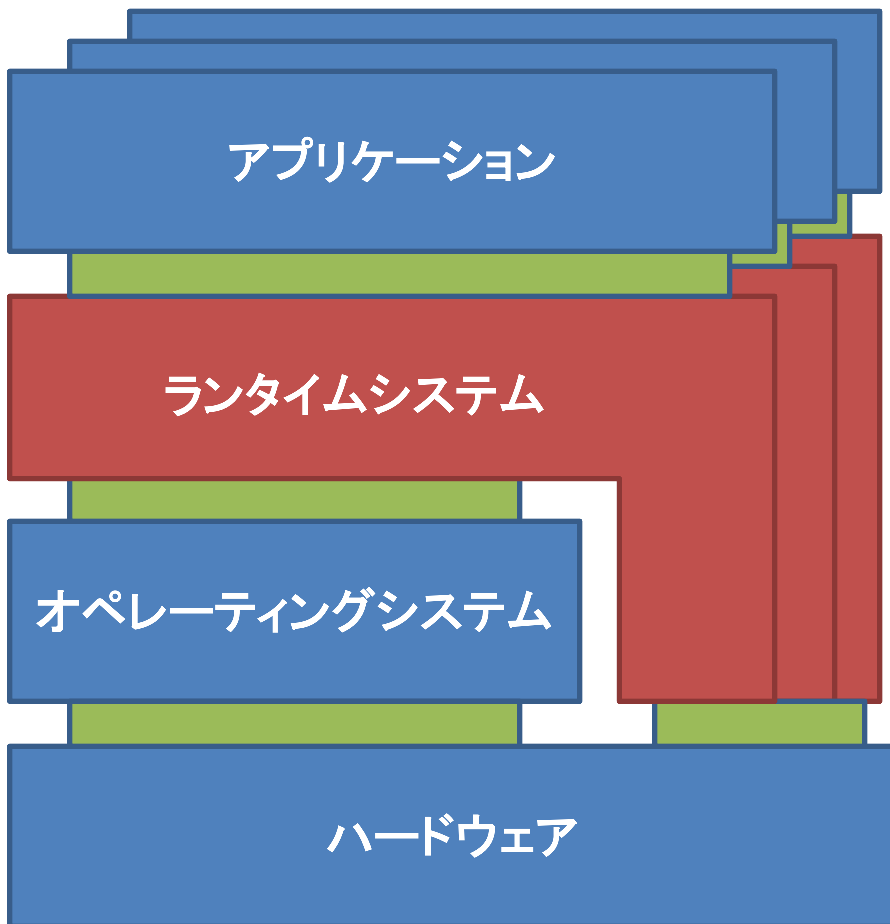
- 既存ソフトウェア（legacy software）の実行
- 他OS，アーキテクチャのプログラムの実行
  - IBM 370[1970]の仮想マシン
  - Cygwin, Wine
  - VMWare，Xen，KVM

コンピュータのインターフェース



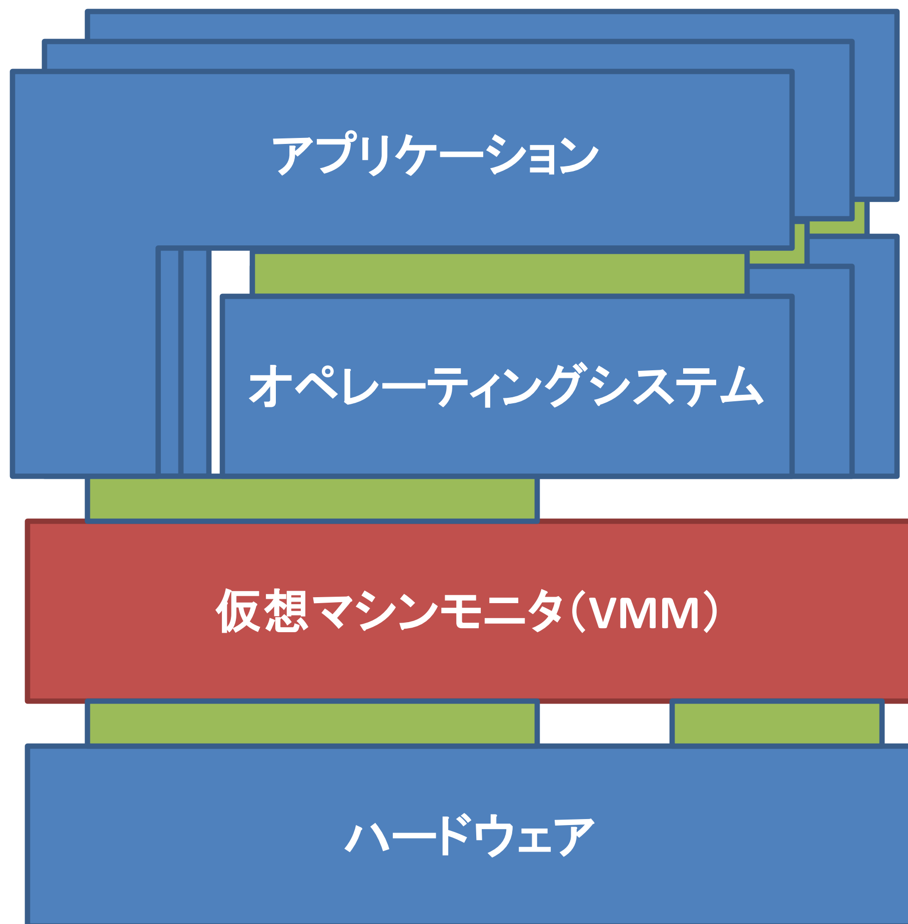
仮想化は上記のインターフェースを模擬すること

仮想マシンのアーキテクチャ



プロセス仮想マシン

Java VMなど



仮想マシンモニタ
(ハイパーバイザ）

サーバ設計における一般的なこと

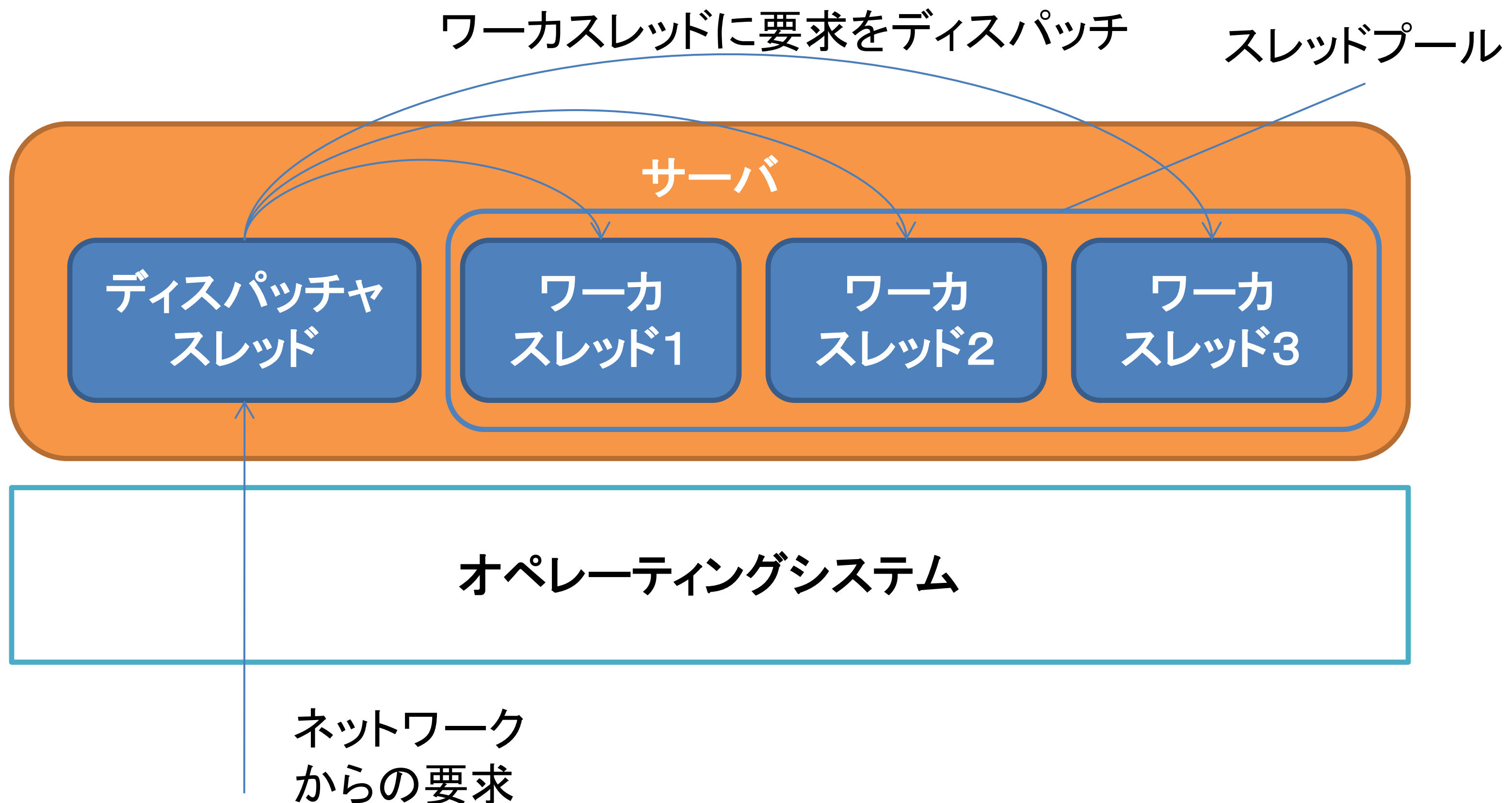
- サーバ
  - クライアントからのリクエストを待ち，実行する
- 反復サーバ（iterative server）
  - サーバがリクエストを実行し，レスポンスを返す
- 並行サーバ（concurrent server）
  - 別のスレッドorプロセスにリクエストを依頼
  - 直ちに次のリクエストを待つ

分散システムにおけるスレッド

- プロセスをブロックさせないでブロッキングシステムコールを呼べる
- 複数のコネクションの管理に有用
- マルチスレッドクライアント
  - 広域ネットワークの遅延（数百ミリ秒～数秒）隠蔽
  - （例）Webブラウザ
    - ページ内複数ドキュメントの並列受信
    - 受信しながら表示

マルチスレッドサーバ

- 典型的なマルチスレッドサーバの例



サーバのコンタクト先

- クライアントはサーバのエンドポイント（end point）orポート（port）にリクエストを発行
- TCP，UDPのポート番号
  - Internet Assigned Numbers Authority (IANA)が管理

範囲	種類	備考
0～1023	Well Known Ports	登録なしに利用不可 UNIXではroot権限が必要
1024～49151	Registered Ports	登録なしに利用不可
49152～65535	Dynamic and/or Private Ports	

代表的なポート番号

キーワード（サービス）	番号	説明
ftp-data	20/tcp, 20/udp, 20/sctp	File Transfer [Default Data]
ftp	21/tcp, 21/udp, 21/sctp	File Transfer [Control]
ssh	22/tcp, 22/udp, 22/sctp	Secure Shell, RFC 4251, 4960
telnet	23/tcp, 23/udp	Telnet
smtp	25/tcp, 25/udp	Simple mail transfer
http	80/tcp, 80/udp, 80/sctp	World Wide Web HTTP
imap	143/tcp, 143/udp	Internet Message Access Protocol
https	443/tcp, 443/udp, 443/sctp	http protocol over TLS/SSL
imaps	993/tcp, 993/udp	Imap4 protocol over TLS/SSL

http://www.iana.org/assignments/port-numbers

http://www.rfc-editor.org/

HTTPサーバへのアクセス例

$ **telnet www.tsukuba.ac.jp 80**
Trying 130.158.69.246...
Connected to www.tsukuba.ac.jp.
Escape character is '^]'.

telnetコマンドでwww.tsukuba.ac.jpのport 80/TCPに接続

GET /index.html HTTP/1.0（リターンを2回）

HTTPサーバへのリクエスト

HTTP/1.1 200 OK
Date: Mon, 14 Dec 2009 22:37:09 GMT
Server: Apache/2.2.3 (CentOS)
...
Content-Length: 451
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
...
</HTML>

HTTPサーバからのレスポンス

Connection closed by foreign host.

HTTPサーバが接続を切断

その他の事項

- 割込
  - 例：FTPサーバへの大きなファイル転送を取り消したい
  - コネクションの切断
  - Out-of-band（OOB）データの送信
    - サーバでシグナル割込がかかる
- 状態
  - ステートレスサーバ
    - クライアントの状態をサーバ側で保持しない。クライアントとは関係なく状態を変更
    - FTP，HTTP，NFSv2，NFSv3
  - ステートフルサーバ
    - 状態を持つ。クライアントが状態を消去する
    - 性能向上のため。障害時の復旧を考える必要がある
    - NFSv4

セッション状態（session state）

- 単一ユーザの状態を一定の期間保持
  - 永遠ではない
  - 失われてもまたやり直せばよい
- Webブラウザのクッキー（cookie）
  - クライアント側にサーバの状態を保持
  - 消去してもまたやり直せばよい

まとめ

- 特に分散システムでは、スレッドは有用
  - ブロッキングI/O操作を行いながらCPUを活用
  - ただし、データ競合を起こさないよう並列性の制御が必要
- 仮想化により既存アプリケーション，他OSのアプリケーションが実行可能に
  - VMMでは実行環境ごと保存，移動が可能
- サーバ設計に関しての一般的事項

演習問題

- プロセスとスレッドの違いは何か？
- XenとKVMはそれぞれどのように仮想化を実現しているか？
- telnetでwww.cs.tsukuba.ac.jpのhttpポートにアクセスしてみよ
- telnetでpoplar.cs.tsukuba.ac.jpのsmtpポートにアクセスし，VRFY tatebe，EXPN c-sotsuken，QUITを入力してみよ