第19回 コンピュータシステム・シンポジウム

TOMOYO Linuxの 2つの実装方式の性能評価

株式会社NTTデータ
技術開発本部
SIアーキテクチャ開発センタ

武田健太郎

TOMOYO Linux？

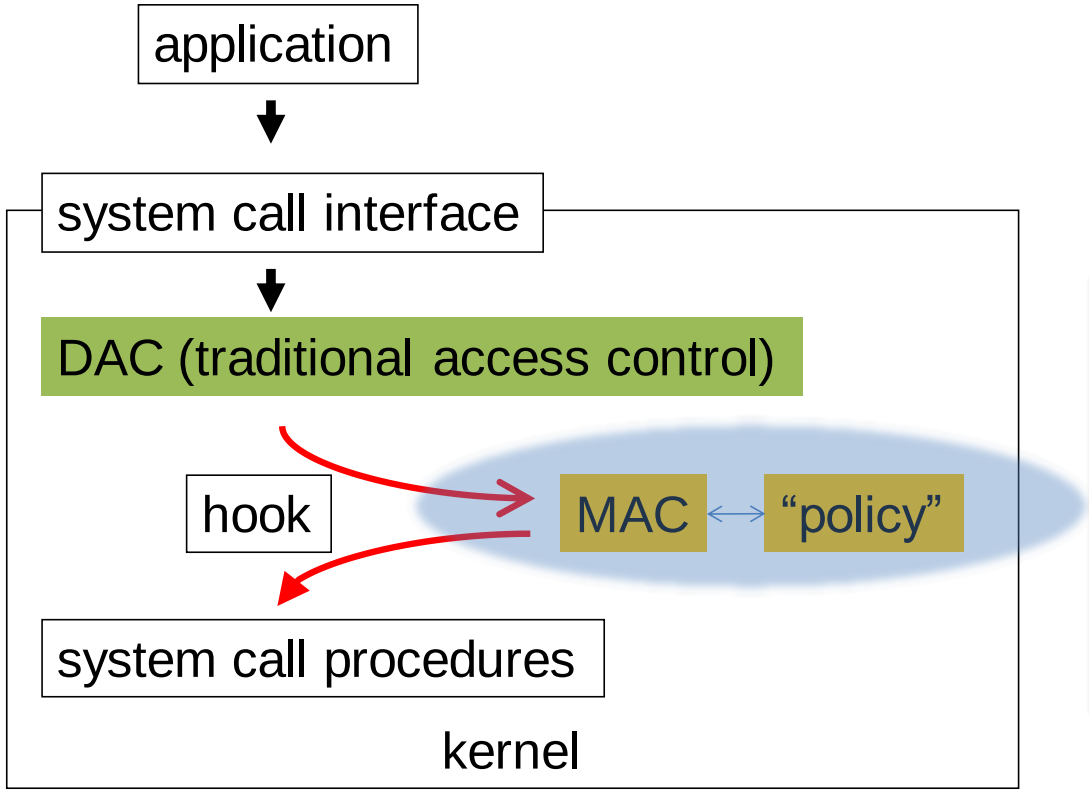
- Linux向けのセキュアOSの1種
  - 他のセキュアOS：SELinux, AppArmor, LIDS…
- Linuxカーネルにセキュリティ強化機能を追加
- 多数のディストリビューションで使用可能

- 自動学習機能で簡単に使える
  - プロセス起動履歴の記録
  - プロセスがアクセスした資源の記録

- GPLのOSSとして公開中
  - http://tomoyo.sourceforge.jp/

セキュアOSと通常のOSの違い

- システムコールにセキュアOSによるアクセス許可のチェックが追加される



通常のOSと比べて
- なにが
- どれだけ

遅くなるのか？

セキュアOSの性能評価の難しさ

- セキュアOSは設定（＝ポリシー）によって大きく性能が変わる
  - どれだけ厳密なポリシーを記述するか
  - ログ出力などの補助機能を有効にするか
- セキュアOSごとにセキュリティ強化の考え方が違う
  - 保護する資源の種類
  - 資源の指定方法（ラベル、パス…）
  - 実装の方式（LSM、独自フック…）

セキュアOSの実装まで理解したうえで
性能を測定、考察することが必要

TOMOYO Linuxの2つの実装

- TOMOYO Linux 2.x系統： LSM
  - 2.6系カーネルにあらかじめ存在するシステムコールのフック（LSM）を利用
  - フックから呼ばれる処理だけを実装すればよい
- TOMOYO Linux 1.x系統： 独自フック
  - システムコールにアクセス許可のチェック関数の呼び出しを挿入
  - カーネルに対してパッチを当てる

LSMと独自フックの具体例

- creatシステムコールから呼ばれるvfs_create

```
int vfs_create(struct inode *dir, struct dentry *dentry, int mode,
               struct nameidata *nd)
{
        int error = may_create(dir, dentry, nd);

        if (error)
                return error;

        if (!dir->i_op || !dir->i_op->create)
                return -EACCES; /* shouldn't it be ENOSYS? */
        mode &= S_IALLUGO;
        mode |= S_IFREG;
        error = security_inode_create(dir, dentry, mode);
        if (error)
                return error;
        /***** TOMOYO Linux start. *****/
        if (nd && (error = CheckSingleWritePermission(TYPE_CREATE_ACL, dentry, nd->mnt)) < 0) return error;
        /***** TOMOYO Linux end. *****/
        DQUOT_INIT(dir);
        error = dir->i_op->create(dir, dentry, mode, nd);
        if (!error)
                fsnotify_create(dir, dentry);
        return error;
}
```

LSM

独自

LSM vs. 独自フック

- LSM：
  - フックがすでにあり、呼び出し先だけを実装すればよい
  - Linuxカーネルは日々変化する

    ➔日々の変化に追従が容易

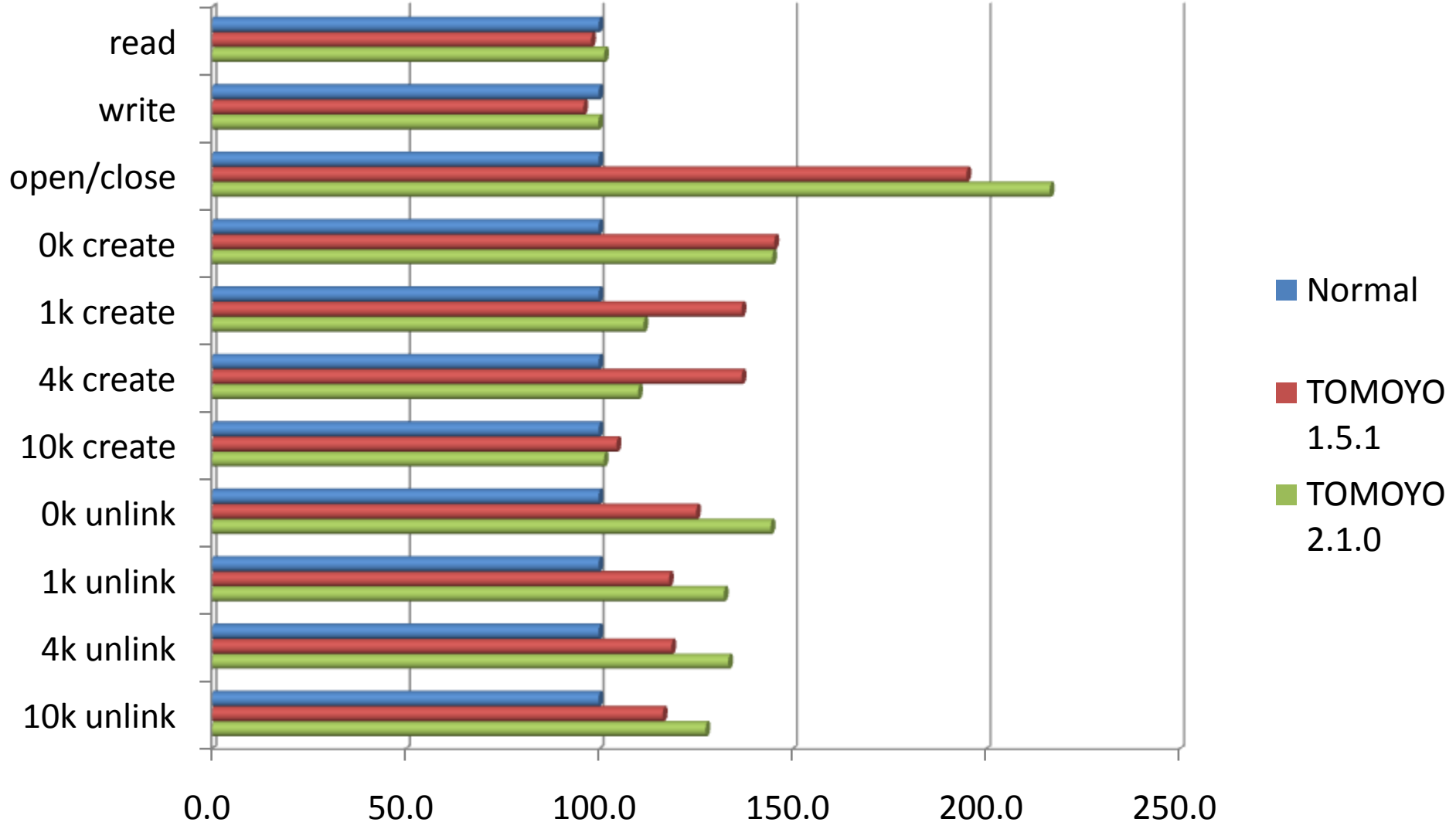
- 独自フック：
  - 好きな場所にフックを挿入できる

    ➔機能の自由度が高い

性能評価

- 測定環境
  - Core 2 Duo 6400 2.13GHz, 2GB RAM
  - LMBench 3.0-a8
- 測定対象
  - Linux 2.6.23.1
  - Linux 2.6.23.1 + TOMOYO Linux 1.5.1（独自フック）
  - Linux 2.6.23.1 + TOMOYO Linux 2.1.0（LSM）
- 測定項目
  - ファイル操作
  - プロセスの生成と実行
  - プロセス間通信

ファイル操作



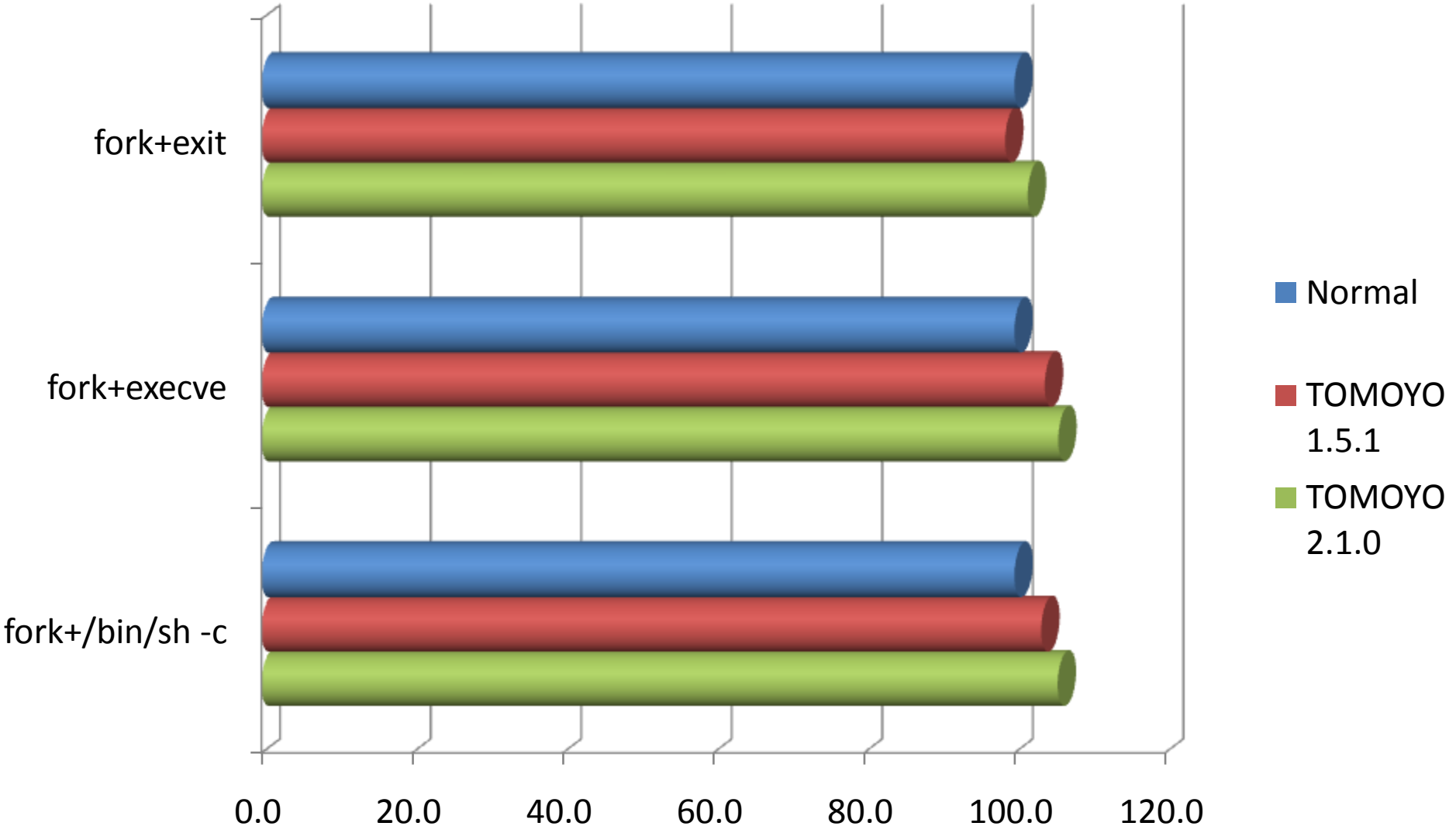
ファイル操作

- オープンに2倍程度の時間がかかる
- read/writeはオーバヘッドなし
  - そもそもフックがかかっていない
- 作成・削除は1.5倍程度の時間がかかる
  - ファイルサイズ 大、オーバヘッド 小
  - read/writeの占める割合が大きくなるため

TOMOYO Linuxのアクセス制御の考え方

- TOMOYO Linuxのアクセス制御の考え方
  - 「パス名」で制御する
  - 「入口」を制御する
- **「パス名」だから…**
  - ファイルの絶対パスを基準としてアクセス制御
    ➔文字列比較処理で時間がかかる
- **「入口」の意味するところ**
  - オープンはチェックするが読み書きはチェックしない
    ➔ファイルを開けばあとは通常のLinuxと同じ

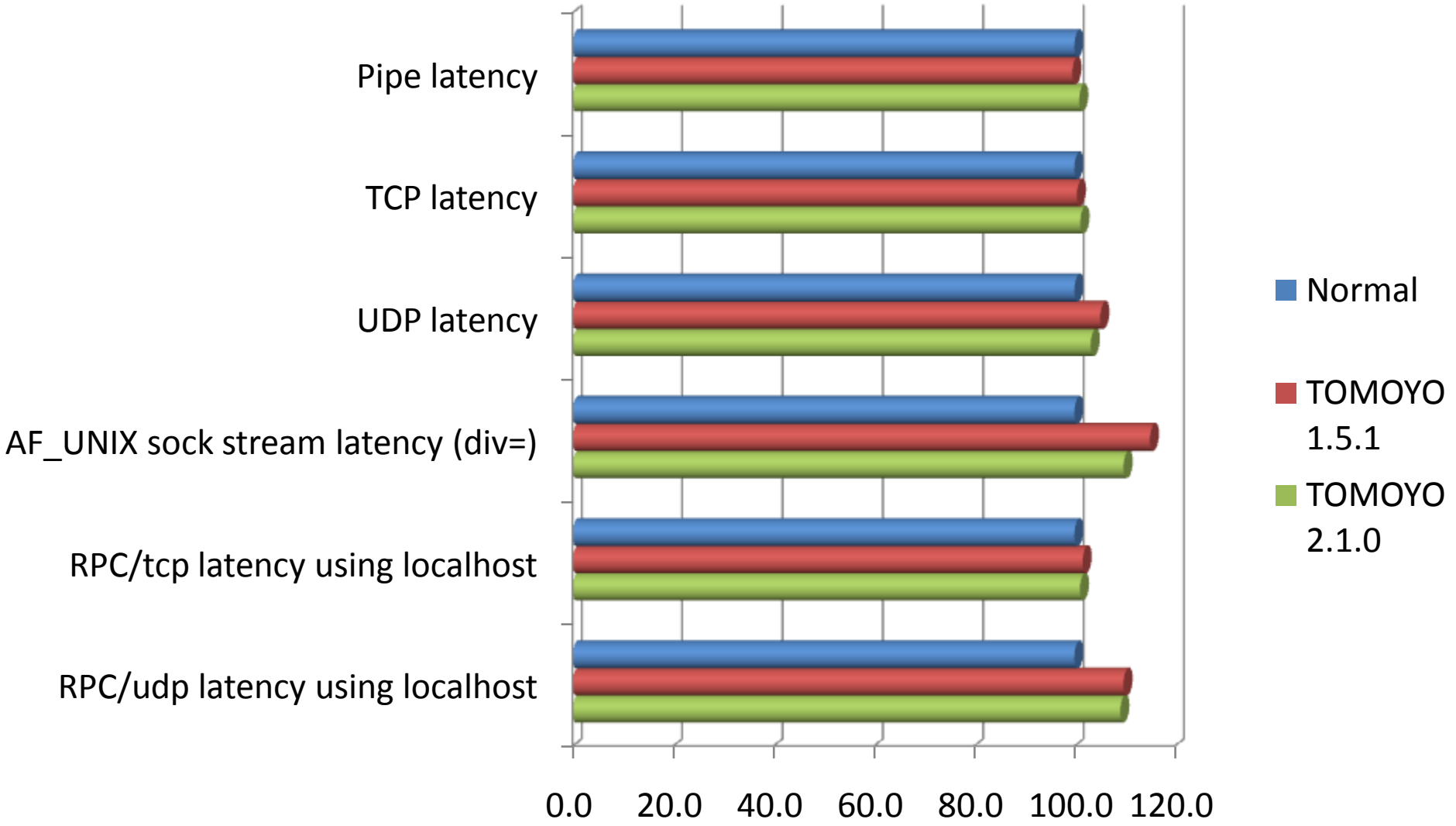
プロセスの生成と実行



プロセスの生成と実行

- 使用されるシステムコール
  - fork()：TOMOYO Linuxのフック なし
  - execve()：TOMOYO Linuxのフック あり
- execveが呼ばれる項目（fork+{execve,/bin/sh –c}）でオーバヘッドが生じる
- ファイル操作と同じく文字列比較を含む
- プログラムの実行そのものが時間がかかる処理であるため、影響は小さい

プロセス間通信



プロセス間通信

- TCPとUDPでアクセス許可のチェックが発生
  - UDPの方がオーバヘッドが大きい傾向
    - TCPは最初のコネクションを張るときだけチェック
    - UDPはコネクションレスなので送受信のたびチェック
- 10%以内の小さなオーバヘッド

メモリ消費

- カーネルロード時点でのメモリ消費
  - TOMOYO 1.5.1: 72kb増加
  - TOMOYO 2.1.0: 68kb増加
- これ以外に、ポリシーの保持にメモリを使用
  - 通常のサーバなら大きくても1MB程度

まとめ

- TOMOYO Linuxの性能への影響
  - ファイルのオープン操作は約2倍の時間がかかる
  - read/writeは変わらない
  - プログラムの実行は5%程度遅くなる
  - TCP/IPを用いたプロセス間通信は5%程度遅くなる
- 通常の用途では…
  - ファイルのオープンより読み書きの頻度の方が高い
  - プログラムの実行はそれほど多くない

➔体感できるほどの影響は出ない

デモ: TOMOYO Linux Live!

- LiveCD: CDブートできるLinux
- Ubuntu 7.10のカーネルをTOMOYO Linuxに置き換えたもの
- 学習モードに設定済み
  - ポリシーを見ればシステムの挙動が手に取るようにわかる