ウィンビー国際的アイドル化計画

Win32 on BeOS

プラットフォームの違いとは？

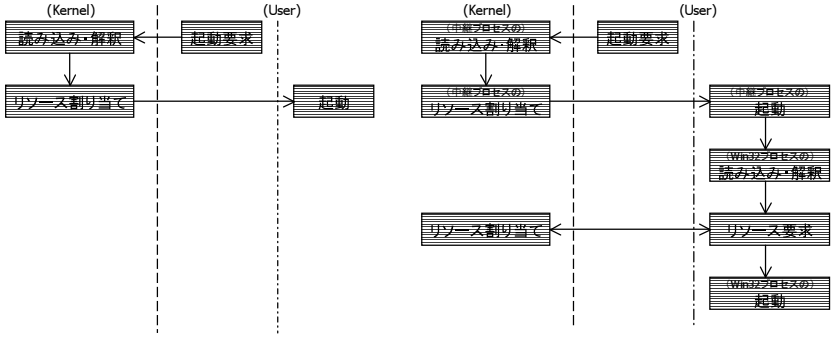
- CPUの違い
  - エミュレーション可能だが、パフォーマンスは落ちる
- APIの違い
  - 仕様がわかっていれば互換APIを用意する事で解決
- 実行形式の違い
  - 直接実行するにはOSのサポートが必要だが、間接的に実行するならアプリケーションでのサポートで十分

BeOSとWin32の場合

- CPUは同じx86
- Win32 APIについてはMicrosoftから十分な資料が公開されている
- OS内部に介入できないため、アプリケーションから間接的に実行するしかない

アプリケーションの実行

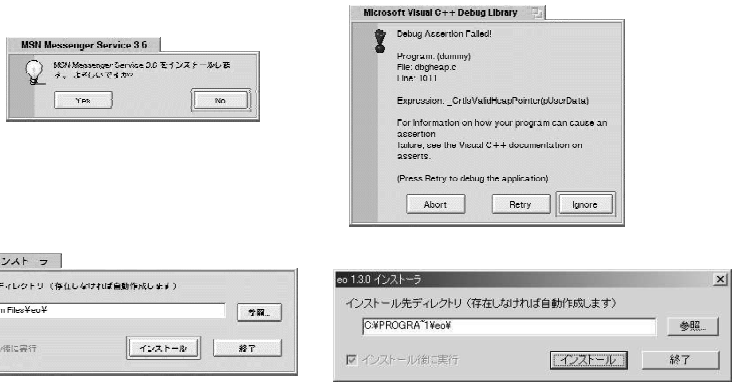
- 直接実行（OS）
- 間接実行（ユーザー）



実装ポリシー

- ユーザーインターフェースやルック&フィールは、OS全体で統一がとれている事が望ましい（Win32アプリケーションについてもBeOS準拠のユーザーインターフェース、ルック＆フィールを実現したい）。

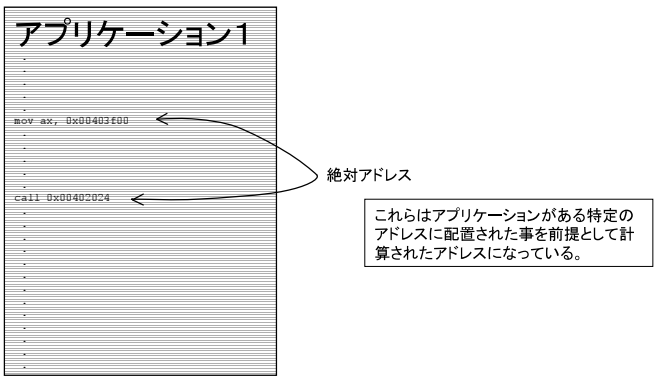
実装ポリシー（具体例）



問題点

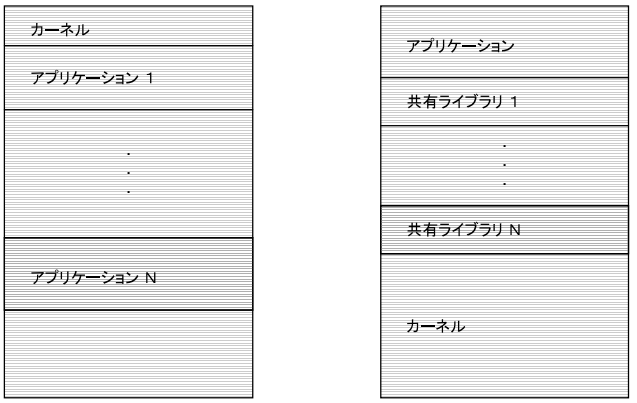
- 膨大な数のAPIを実装
- 双方のAPIについて整合性をとる難しさ
- COMとgccにおける仮想テーブルの仕様の違い
- メモリ空間の使用法の違いによる再配置不能問題

絶対アドレス

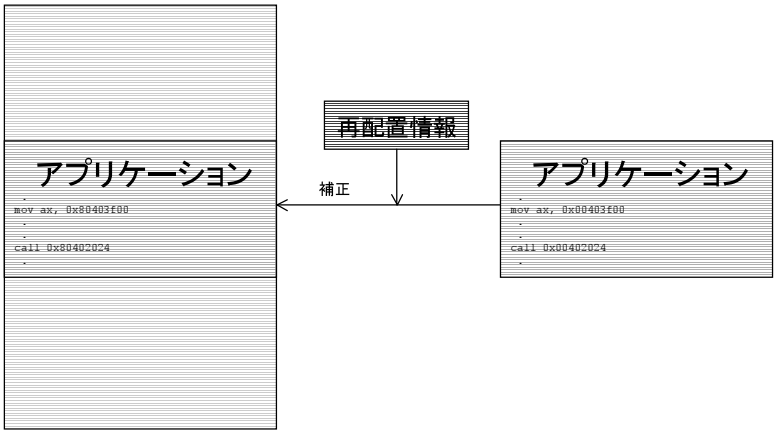


メモリ配置

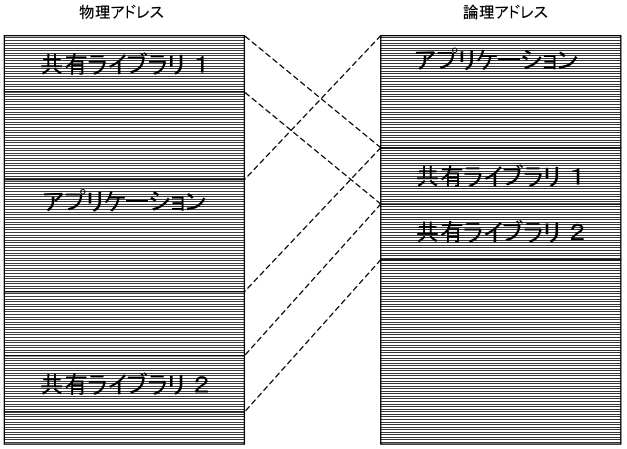
実際には、メモリの使用状況によって実行ファイル・共有ライブラリの配置アドレスは変化する



再配置情報



仮想メモリ

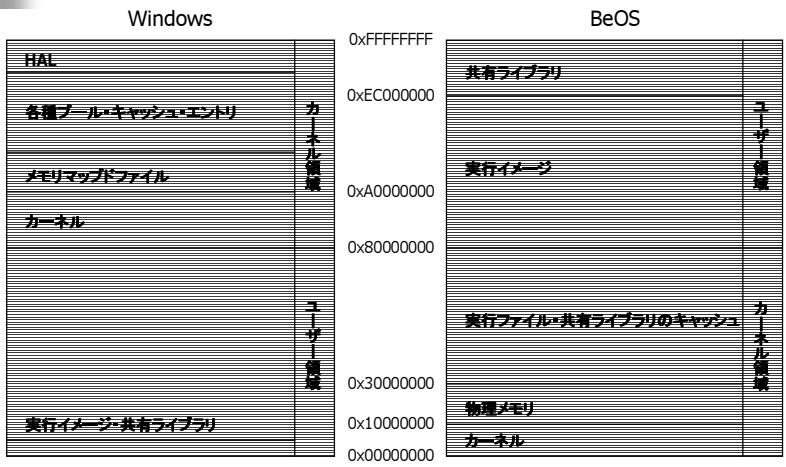


論理アドレスはアプリケーション毎に個別に設定されるため、実行ファイルは常に仮定していたアドレス上に配置される。

ただし、共有ライブラリについては、リンクされるアプリケーションによって配置可能な場所が変化するため、再配置は必要。

最近のWindowsでは、アプリケーション起動の速度を稼ぐため、実行ファイルには再配置情報を含めない事が多い。

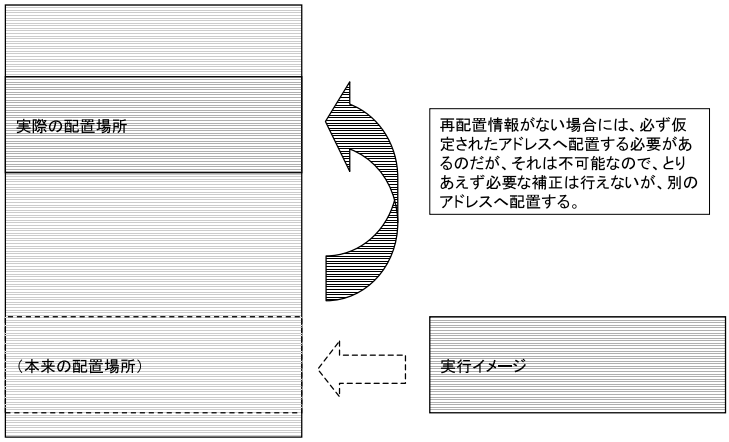
Windows/BeOSのメモリ配置



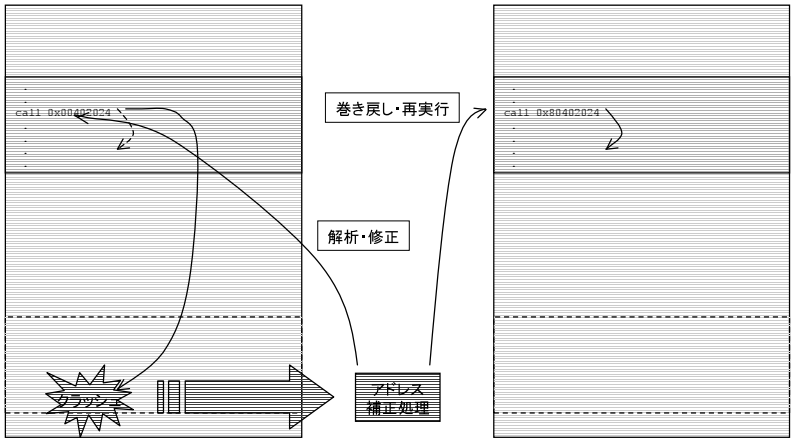
Windowsに置ける標準的な実行イメージ配置場所にBeOSではカーネルが存在している!!

⇒ 再配置が必要不可欠

動的再配置（1）- 仮配置



動的再配置（2）- クラッシュ



動的再配置（3） - 予測

下の例のように、単純に問題個所だけ補正していたのでは解決できない場合が多々ある！

```
char buffer[1024];
char *p;
for (p = buffer; p != &buffer[1024]; p++) *p = 0;
```

1. 最初の「*p = 0」で、不正なアドレスへのアクセスが発生。「p」の指すアドレスが補正される。
2. この状態では「&buffer[1024]」という値は補正前の値になっている。
3. 終了条件に一致せず、永遠にメモリを「0」で埋め尽くす。
4. スタックの破壊。この状態からはもはや復帰不能。

問題個所周辺のプログラム・データの流れについて、十分な解析・予測の上で補正する必要があり、これを自動的に完全動作させるのは不可能。

今後の流れ

- 自分自身、忙しくなってきてしまったので、ひとりで開発を継続するのは難しい。
- 海外からプロジェクトの進展やソースの公開についての問い合わせが多い。

■ソースを公開し、共同開発の形態を取る。