

ふふふ、Apache2 といえば、2つのメインがあるんです。俺的に。

APRの登場

HTTP以外のプロトコルでもOKになった

Apache Portable Run-time (APR) は apache がサポートするすべてのプラットフォームで同じAPIを提供するもので、APRはapacheがすべてのプラットフォームでネイティブ関数を呼び出す際にも、基本となるコードを追いやすいような見通しのよい状態に維持するために大きく寄与します。まあ、これって

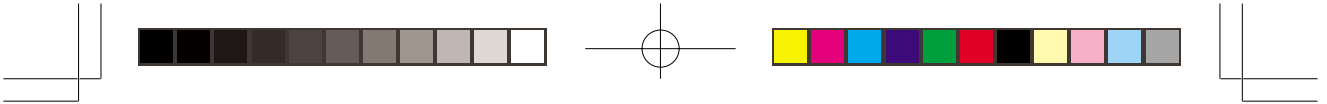
XFree86 4 でもこうなったみたいなんで、潮流なんですよ

apache1.3では、たいていのUnixプラットフォーム、windowsやいくつかのメインフレーム等に移植されました。しかし、なぜapache2でAPRを含むことになったのでしょうか？移植される一方で、apache1.3はコードが複雑になり、メンテナンスが非常に大変になり、それら移植されたプラットフォームでは、その速度を別にしても、ネイティブAPIを使うメリットが薄れてきました。

はっきりいってあらゆるUnixのネイティブAPIなんてわかんねー

と、思いませんか？

たとえば、apache1.3のwindowsでの移植では多くの問題がありました…らしい。windowsのapacheはpre forkしたサーバではなく、スレッドを用いています。ほかにもCGIプロセス生成の方法がwindowsとunixで



は異なるなど、根本の設計に関わる部分の変更があり、当然それに伴うコードの変更がありました。結果として、同じようなコードが複数存在することになり、ある特定の環境でのバグを修復してもまた別の環境バグが残ったり、副作用が発生するなどコードのメンテナンスは困難になる一方となりました。そこで、APRはこのような問題の解決策として設計されました。

まあ、スレッドは、ごとむセンサーあたりに解説してもらいましょう (わら)

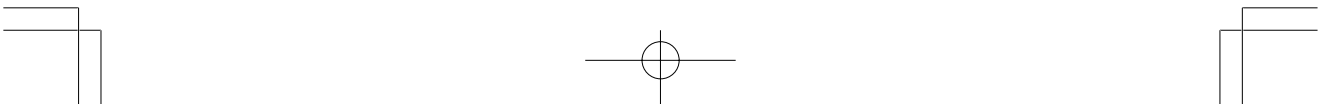
次に問題になったのは、パフォーマンスの問題です。apacheは歴史的にPOSIX準拠の関数を利用するwebサーバで、それゆえにapacheは信頼できるPOSIX準拠のOSで動作する必要がありました。しかし、現在ではapacheが動作するシステムは、POSIXに準拠したものから実装が不完全なものまで多岐に渡ります。たとえばWindowsは完全なPOSIX準拠のOSですが、ネイティブAPIを使うかわりに、POSIX関数を使うとシステムが不安定になったり十分なパフォーマンスが得られません。

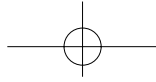
これらの問題に対する答えとして2つの選択肢がありました。ひとつの答えは、

`#ifdef` を多用し異なるプラットフォームごとにコードを分けることです (うひひ)

この利点は、すべてのコードをひとつの場所にとどめることが可能な点にあります。欠点として読むのも手を加えるのも困難になることでした。これは、あるプラットフォームでのバグ修正が必ずしもすべてのプラットフォームで修正されるとは限らないことになり、修正を難しくします。

ふたつ目の答えは、apacheから同じに見えるようなPortable Runtimeを作成することでした。利点は、apacheのコードがより読みやすく理解しやすいものになり、1つのコード修正が、全てのプラットフォームに反映されることである。たとえばAPRが不完全であれば、そのプラットフォームでのネイティブ関数を利用できないことになるため、全てのプラットフォームで動くようなAPRを作成する必要があります。





●なぜ APR を書くことになったのか

Portable Runtime を apache2 に含めることになったとき、次の問題は既存のものを使うか、それとも新たに作成するか、ということでした。最初のステップは、既存ものを評価することでした。そこで浮上してきたのは、

**Netscape Portability Run-Time (NSPR) と、
ADAPTIVE Communication Environment (ACE)**

の2つでした。

NSPR はよくデザインされており、クロスプラットフォームでも動作することがわかっていました。しかし、NSPR を apache とあわせて利用するにはいくつかの問題がありました。はじめは、NSPR のライセンスが apache のライセンスと異なっていたため、議論を経て Netscape はライセンスを apache で要求するようなものに変更することを約束しました。しかし、数ヶ月が経過しても Netscape からは新しいライセンスはリリースされませんでした。

このあいだに、ライセンスのせいでステ決定

となり、APR の作成を始めてから NSPR はライセンスが新しくなってリリースされました。しかし、その間の進歩はなかったわけで、ASF は apache に要求される機能が NSPR に多くの部分で不十分だという理由で採用は見送られました。

こうして、ライセンスのせいでソフトは進化しなかったわけ（わら

ACE は、C ではなく C++ によって書かれていたことが不採用の一番の理由でした。一般的な感覚として、C と C++ を比べたときに修得している人の数は C のほうが多いといえますし、apache でも、apache に参加する人が多くなることを目的に、長い間 C++ でなく C を採用しています。

結局、Apache group は自らランタイムライブラリを書きはじめました

●何が APR によって提供されるか

最初のバージョンの APR が提供する関数は、apache2 で使うこともあり、サーバアプリケーションを作るのによく使うものに限られています。将来のバージョンでは apache で使うことがない関数も追加される予定です。NSPR でも見られるような次の機能が必要になるでしょう。

- Status Values
- File I/O
- Network I/O
- Critical Section Locking
- Memory Mapped Files
- Threads and Processes
- Time
- Memory Management
- apache1.3 からの関数
- 移植ルーチン

うへ、多すぎ…

それぞれの APR は time 型を除いて、コンテキストへのポインタによって構成され、そのコンテキストはメモリマネージメントと、それぞれの APR 型とユーザデータを結びつけるのに使われます。

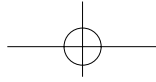
●Status Values

APR における Status Values は

- system error
- error value
- status value
- success

の 4 つから構成されています。

system error は 0 と APR_OS_START_ERROR のどちらかになります。このエラーはネイティブ関数が失敗するなどの理由で APR が不完全に終了したときに返ります。



APRのerror valueはAPR_OS_START_ERRORとAPR_OS_START_STATUSのどちらかになります。これは、system errorの場合と異なり、プラットフォームに関係ないなんらかの理由でAPR関数が失敗したときに返ります。たとえば、初期化していないファイルに対してapr_get_filenameをすると、APR_ENOFILEが返ります。

APRのstatus valueはAPR_OS_START_STATUSとAPR_OS_START_SYSERRORのどちらかの値になります。これは、APR関数が成功してからあとからなんらかの情報が届けられる必要があるときに返ります。たとえば、apr_waitは子プロセスが動作しているときには、APR_CHILD_NOTDONEを必要とし、さらにその子プロセスが終了したときにはAPR_CHILD_DONEの返りを必要とします。

success valueはAPR関数が成功し、あとから必要な情報が何もないうちに、APR_SUCCESSが返ります。

●File I/O

File I/Oはファイルやパイプの読み書きのたびにいつでも使われます。プラットフォームの違いを吸収するAPRがあれば、その見通しがよくなります。WindowsとUnixではファイルハンドルの違いは大きく、Windowsではファイルを扱うのにHANDLE型を用いますし、UnixではバッファするファイルもバッファしないファイルについてもFILE型と整数の形で扱います。この差を解決するために、APRはapr_file_t型をすべてのプラットフォームでファイルを扱うために使用します。open, read, write, close, delete, duplicate, ファイル情報の入手など、すべての標準ファイル操作はこの型を通してサポートされています。

File I/Oはパイプにももちろん使われます。Windowsでのapacheを追っている人はよく知っているように、WindowsでのパイプはUnixでのパイプサポートと同じレベルで提供されているわけではありません。たとえば、Windowsのパイプにはタイムアウトがありませんが、APRはタイムアウト値をファイルにつけることで、これをサポートします。

ディレクトリアクセスについては、apr_dir_t型を用いて、opened, closed, created, read, deletedをサポートします。

●network I/O

network I/Oはネットワーク越しに通信をする際に利用されます。現時

点ではTCPのみがサポートされています。時間に関する変更があり、ソケットは`apr_socket_t`型を用いて、`read`や`write`のタイムアウト時間を決めることができます。タイムアウト時間が来れば、`APR_ETIMEUP`が返ります。`port`番号とソケットは`apr_socket_local_port`と`apr_bind`を使ってbindすることができます。その後は`apr_listen`を使うことで待ち受けます。また、`apr_connect`は任意の2つのTCPソケットを結びつけます。

ソケットの読み書きは`apr_send`と`apr_recv`で可能です。ほかにも、`apr_sendfile`があり`apr_sendfile`をサポートしているプラットフォームでは`APR_HAS_SENDFILE`マクロが`TRUE`に定義されています。

ほかにもポーリングソケットが`apr_poll`の形で組みこまれています。このルーチンはパイプやファイルからは利用できません。ポーリングの利用にはまず、`apr_setup_poll`をして、次に`apr_add_poll_socket`でソケットをポーリング構造に入れます。現在のソケットをポーリングするには、`apr_poll`をする必要があります。それぞれのソケットに`apr_get_revents`をすることで、その結果をチェックすることができます。将来はAPRはポーリングされたソケットのリストを返す関数が含まれる予定です。

● Critical Section Locking

ロックに関しては、`apr_lock_t`型があり、`APR_CROSS_PROCESS`、`APR_INTERPROCESS`、`APR_LOCKALL`から成っています。`APR_CROSS_PROCESS`はほかのプロセスから触られないようにします。`APR_INTERPROCESS`は、通常のスレッドロックです。`APR_LOCKALL`は一番強力なロックで、ほかのスレッドからも、プロセスからも守られますが、同時に一番負荷が高いロックでもあります。たとえば、スレッドを利用しないのであれば、`APR_CROSS_PROCESS`を使うのがよいでしょう。

● Memory Mapped Files

APRにおけるメモリマップは`apr_mmap_t`型があります。プラットフォームによってはシェアードメモリを使いますが、`apr_mmap_t`型では使うことができません。APRでは移植性を高めるために、`mmap`のサブセットにとどまっています。APRの`mmap`でできるのは、`create`、`delete`、`read`だけになっています。`mmap`を作成するには、ファイルがAPRのファイルタイプである必要があります。APRがそのプラットフォームで`mmap`をサポートしているならば、`APR_HAS_MMAP`マクロが`TRUE`で定義されます。



● Threads and Processes

APRにおけるスレッドは`apr_thread_t`型になります。これは、スレッドの`crate`と`destroy`が可能です。スレッドがデタッチされると、即使用されていたリソースが解放されますが、同じプロセスの他スレッドがそれを利用していれば、そのスレッドの終了を待ちます。スレッドがデタッチされていなければ、同プロセスの異なるスレッドが加わることができます。加わったスレッドは他のスレッドの動作が終わるまで通知されています。`apr_detach`することで現在動作しているスレッドをデタッチできます。多くのプラットフォームにおいてメモリリークや不安定な動作を引き起こすので、推奨されることではありませんが動作しているスレッドを`kill`することもできます。

動作しているスレッドにデータを付けることもできます。これで、スレッドローカルストレージや、スレッドプライベートストレージとして参照されます。APRでは`apr_threadkey_t`型で行われ、スレッドにデータを付けたり、そのデータを検索することができます。

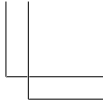
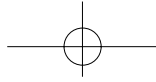
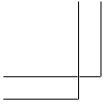
プロセスも同様にAPRで生成することができます。これには`apr_fork`を使いますが、これは`portable`ではありません。`apr_fork`をサポートしているプラットフォームでは`APR_HAS_FORK`が`TRUE`になっています。より`portable`な方法は、`apr_create_process`でプロセスを新規に作成する方法です。両者の違いは、`apr_fork`がプロセスを作って、そのプロセスはコードの同じ部分で動作しますが、`apr_create_process`はプロセスを作って新たなプログラムが動作するところです。`apr_create_process`は`stdin`、`stdout`、`stderr`へのパイプを作ることもできます。

子プロセスが終了した場合は`APR_CHILD_DONE`を、終わっていない場合は`APR_CHILD_NOTDONE`が返ります。

いまのところ、他プロセスへ送ることのできるシグナルは`TERM`シグナルだけですが、`ap_kill`でほかのシグナルも送れるようになるはずですが。

● Time

APRの中で唯一`status value`が返らないのが時間に関するライブラリです。APRでは2つの異なる`time`型があります。ひとつは`apr_time_t`型で、UTCの1970年1月1日零時を基点とするマイクロ秒単位です。`apr_now`は経過時間を`apr_time_t`フォーマットで返します。この関数はグローバル変数として使用する必要があります。もうひとつの型は



apr_exploded_time_t 型です。これは時間値を分類するものになります。2 つの型をコンバートする関数が定義されています。ひとつは apr_rfc822_date で、RFC822 で定義されている形にコンバートします。同じく apr_ctime は POSIX 関数の ctime と同じ形にコンバートします。ユーザが定義した形で出力する関数として apr_strftime があり、これも POSIX の strftime 関数と同じように利用します。

● Memory Management

APR ではメモリ管理はコンテキストを使って実装されています。コンテキストはメモリプールとユーザデータと関数へのポインタによって構成されます。メモリプールはプログラマが解放を意識する必要はありません。すべてのメモリは apr_palloc か apr_pcalloc を使ってアロケートされます。メモリのすべてで特定のメモリプールがいらなくなったならば、apr_clear_context や apr_destroy_context を使って解放することができます。また、apr_create_context を用いてサブコンテキストを作れます。

● Functions Taken from Apache1.3

apache1.3 で定義されていた ap_ から始まる関数セットもあります。テーブルや、getopt、snprintf、cpystrn、MD5 や SHA1 の計算ルーチンなどが含まれます。

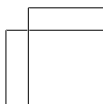
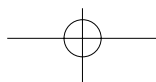
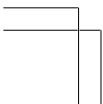
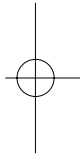
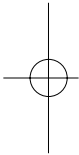
● Portability Routines

はあ、なんか疲れてきましたね (わら
そこで手抜きの方法が考案されました

APR 開発者とはいっても、すべてのプログラムが APR で動作していない世界で生きていかねばなりません。apr_get_os_* と apr_put_os_* は APR とネイティブ関数の相互変換をします。これら関数は安定していますが、非常にパフォーマンスが悪いため、これはあくまで APR を使っていないプログラムが APR を使うようになるまでの過渡的なものです。

● APR Success Stories

APR はすでに実行可能なライブラリとなっています。ApacheBench は





Apache から来たベンチマークツールです。問題は ApacheBench が歴史的に Unix の上でのみ動作するツールだということです。ApacheBench を windows に移植する作業は行なわれていましたが、windows での変更はコードを読むのに非常に邪魔でした。APR の開発が進み、ApacheBench を APR がサポートするすべてのプラットフォームへの移植は楽になりました。結果として今では ApacheBench は Unix、Windows、OS/2、BeOS で動作しています。そのためのコードの変更はわずかで、読みにくくなることもありませんでした。

APR は apache のパスワードファイルを生成するのに用いる htpasswd の移植にも使われています。

今、Apache Group では suexec、logresolve、rotetelogs を APR に移植する作業を行っています。これらは、apache を直接サポートするプログラムですが、apache2.0 のリリース後に、apache に直接の関係がないプログラムも APR に移植する予定です。

● Future Work in APR

APR は安定した、フレキシブルなツールですが、APR はもっと一般的になり多くのプログラマが使うようになり、APR がさらによいものになりマルチプラットフォーム対応のプログラムが書きやすく、読みやすいものになることが、APR のこれから先の目標になります。

さーて、つーわけで妄想は、

Posix ステ、APR ありゃいいや！

って次第なんでありませう。

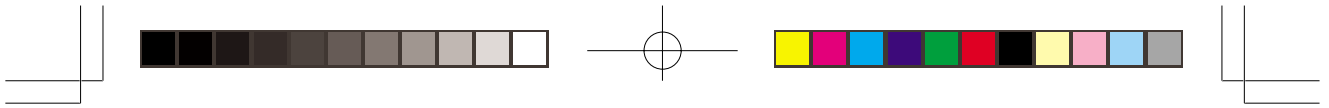
まあ、さらなる機能として、apache2 になってどんな tcp サーバにもなれるんで、

APR をつかって MTA でも IMAP でも書いてやるぜうひひ

が妄想なのでした。

Hack make dream real

これでもうたって、すすんでいくっす (わら



De'an BNU/Linux 不彻底入门 2000 年冬号

