perlembed

Table des matières

1 NAME/NOM

perlembed - Utiliser Perl dans vos programmes en C ou C++

2 DESCRIPTION

2.1 PRÉAMBULE

Désirez-vous :

Utiliser du C à partir de Perl ?
Consultez *perlxstut*, *perlxs*, *h2xs*, et *perlguts*.

Utiliser un programme Unix à partir de Perl ?
Consultez les paragraphes sur les BACK-QUOTES et les fonctions `system` et `exec` dans *perlfunc*.

Utiliser du Perl à partir de Perl ?
Consultez do in *perlfunc*, eval in *perlfunc*, require in *perlfunc* et use in *perlfunc*.

Utiliser du C à partir du C ?
Revoyez votre analyse.

Utiliser du Perl à partir du C ?
Continuez la lecture de ce document...

2.2 SOMMAIRE

2.3 Compiler votre programme C

Si vous avez des problèmes pour compiler les scripts de cette documentation, vous n'êtes pas le seul. La règle principale : COMPILER LES PROGRAMMES EXACTEMENT DE LA MÊME MANIÈRE QUE VOTRE PERL A ÉTÉ COMPILÉ. (désolé de hurler.)

De plus, tout programme C utilisant Perl doit être lié à la *bibliothèque perl*. Qu'est-ce ? Perl est lui-même écrit en C ; la bibliothèque perl est une collection de programmes C qui ont été utilisés pour créer votre exécutable perl (*/usr/bin/perl* ou équivalent). (Corollaire : vous ne pouvez pas utiliser Perl à partir de votre programme C à moins que Perl n'ait été compilé sur votre machine, ou installé proprement – c'est pourquoi vous ne devez pas copier l'exécutable de Perl de machine en machine sans copier aussi le répertoire *lib*.)

Quand vous utilisez Perl à partir du C, votre programme C –habituellement– allouera, « exécutera » et désallouera un objet *PerlInterpreter*, qui est défini dans la bibliothèque perl.

Si votre exemplaire de Perl est suffisamment récent pour contenir ce document (version 5.002 ou plus), alors la bibliothèque perl (et les en-têtes *EXTERN.h* et *perl.h*, dont vous aurez aussi besoin) résiderons dans un répertoire qui ressemble à :

```
/usr/local/lib/perl5/votre_architecture_ici/CORE
```

ou peut-être juste

```
/usr/local/lib/perl5/CORE
```

ou encore quelque chose comme

```
/usr/opt/perl5/CORE
```

Pour avoir un indice sur l'emplacement de CORE, vous pouvez exécuter :

```
perl -MConfig -e 'print $Config{archlib}'
```

Voici comment compiler un exemple du prochain paragraphe, Ajouter un interpréteur Perl à votre programme C, sur ma machine Linux :

```
% gcc -O2 -Dbool=char -DHAS_BOOL -I/usr/local/include
-I/usr/local/lib/perl5/i586-linux/5.003/CORE
-L/usr/local/lib/perl5/i586-linux/5.003/CORE
-o interp interp.c -lperl -lm
```

(Le tout sur une seule ligne.) Sur ma DEC Alpha utilisant une vielle version 5.003_05, l'incantation est un peu différente :

```
% cc -O2 -Olimit 2900 -DSTANDARD_C -I/usr/local/include
-I/usr/local/lib/perl5/alpha-dec_osf/5.00305/CORE
-L/usr/local/lib/perl5/alpha-dec_osf/5.00305/CORE -L/usr/local/lib
-D__LANGUAGE_C__ -D_NO_PROTO -o interp interp.c -lperl -lm
```

Comment savoir ce qu'il faut ajouter ? Dans l'hypothèse où votre Perl est postérieur à la version 5.001, exécutez la commande `perl -v` et regardez les valeurs des paramètres "cc" et "ccflags".
Vous aurez à choisir le compilateur approprié (*cc*, *gcc*, etc...) pour votre machine : `perl -MConfig -e 'print $Config{cc}'` vous indiquera ce qu'il faut utiliser.
Vous aurez aussi à choisir le répertoire de bibliothèque approprié(*/usr/local/lib/...*) pour votre machine. Si votre ordinateur se plaint que certaines fonctions ne sont pas définies, ou qu'il ne peut trouver *-lperl*, vous devrez alors changer le chemin à l'aide de l'option `-L`. S'il se plaint qu'il ne peut trouver *EXTERN.h* et *perl.h*, vous devrez alors changer le chemin des en-têtes à l'aide de l'option `-I`.
Vous pouvez avoir besoin de bibliothèques supplémentaires. Lesquelles ? Peut-être celles indiquées par

```
perl -MConfig -e 'print $Config{libs}'
```

Si votre binaire perl est correctement installé et configuré, le module **ExtUtils::Embed** pourra déterminer toutes ces informations pour vous :

```
% cc -o interp interp.c `perl -MExtUtils::Embed -e ccopts -e ldopts`
```

Si le module **ExtUtils::Embed** ne fait pas partie de votre distribution Perl, vous pouvez le récupérer à partir de http://www.perl.com/perl/CPAN/modules/by-module/ExtUtils::Embed. Si cette documentation est livrée avec votre distribution Perl, c'est que vous utilisez la version 5.004 ou plus et vous l'avez sûrement.)
Le kit **ExtUtils::Embed** du CPAN contient aussi l'ensemble des sources des exemples de ce document, des tests, des exemples additionnels et d'autres informations qui pourraient vous servir.

2.4 Ajouter un interpréteur Perl à votre programme C

Dans un sens perl (le programme C) est un bonne exemple d'intégration de Perl (le langage), donc je vais démontré l'intégration avec *miniperlmain.c*, inclus dans les sources de la distribution. Voici une version bâtarde, non portable de *miniperlmain.c* contenant l'essentiel de l'intégration :

```
#include <EXTERN.h>               /* from the Perl distribution     */
#include <perl.h>                 /* from the Perl distribution     */

static PerlInterpreter *my_perl;  /***    The Perl interpreter    ***/

int main(int argc, char **argv, char **env)
{
    my_perl = perl_alloc();
    perl_construct(my_perl);
    perl_parse(my_perl, NULL, argc, argv, (char **)NULL);
    perl_run(my_perl);
    perl_destruct(my_perl);
    perl_free(my_perl);
}
```

Remarquez que nous n'utilisons pas le pointeur `env`. Normalement passé comme dernier argument de `perl_parse`, `env` est remplacé ici par `NULL`, qui indique que l'environnement courant sera utilisé.
Compilons maintenant ce programme (je l'appellerai *interp.c*) :

```
% cc -o interp interp.c `perl -MExtUtils::Embed -e ccopts -e ldopts`
```

Après une compilation réussie, vous pourrez utiliser *interp* comme vous le feriez de perl lui-même :

```
% interp
print "Pretty Good Perl \n";
print "10890 - 9801 is ", 10890 - 9801;
<CTRL-D>
Pretty Good Perl
10890 - 9801 is 1089
```

ou

```
% interp -e 'printf("%x", 3735928559)'
deadbeef
```

Vous pouvez aussi lire et exécuter des expressions Perl à partir d'un fichier en plein milieu de votre programme C, en plaçant le nom de fichier dans *argv[1]* avant d'appeler *perl_run*.

2.5 Appeler un sous-programme Perl à partir de votre programme C

Pour appeler un sous-programme perl isolé, vous pouvez utiliser les fonctions *perl_call_** décrite dans *perlcall*. Dans cet exemple nous allons utiliser `perl_call_argv`.

Ceci est montré ci-dessous, dans un programme appelé *showtime.c*.

```
#include <EXTERN.h>
#include <perl.h>

static PerlInterpreter *my_perl;

int main(int argc, char **argv, char **env)
{
    char *args[] = { NULL };
    my_perl = perl_alloc();
    perl_construct(my_perl);

    perl_parse(my_perl, NULL, argc, argv, NULL);

    /*** n'utilise pas perl_run() ***/

    perl_call_argv("showtime", G_DISCARD | G_NOARGS, args);

    perl_destruct(my_perl);
    perl_free(my_perl);
}
```

où *showtime* est un sous-programme qui ne prend aucun argument (c'est le *G_NOARGS*) et dont on ignore la valeur de retour (c'est le G_DISCARD>). Ces drapeaux, et les autres, sont décrits dans *perlcall*.

Je vais définir le sous-programme *showtime* dans un fichier appelé *showtime.pl* :

```
print "Je ne devrai pas etre affiche.";

sub showtime {
    print time;
}
```

Suffisamment simple. Maintenant compilez et exécutez :

```
% cc -o showtime showtime.c `perl -MExtUtils::Embed -e ccopts -e ldopts`

% showtime showtime.pl
818284590
```

Indique le nombre de secondes écoulées depuis le 1er janvier 1970 (le début de l'ère unix), et le moment où j'ai commencé à écrire cette phrase.

Dans ce cas particulier nous n'avons pas besoin d'appeler *perl_run*, mais en général il est considéré comme étant une bonne pratique de s'assurer de l'initialisation correcte du code de la bibliothèque, incluant l'exécution de toutes les méthodes `DESTROY` des objets et des blocs `END {}` des packages.

Si vous voulez passer des arguments aux sous-programmes Perl, vous pouvez ajouter des chaînes de caractères à la liste `args` terminée par `NULL` passée à *perl_call_argv*. Pour les autres types de données, ou pour consulter les valeurs de retour, vous devrez manipuler le pile Perl. Ceci est expliqué dans le dernier paragraphe de ce document : Trifouiller la pile Perl à partir de votre programme C.

2.6 Évaluer un expression Perl à partir de votre programme C

Perl fournit deux fonctions de l'API pour évaluer des portions de code Perl. Ce sont perl_eval_sv in *perlguts* et perl_eval_pv in *perlguts*.

Ce sont sans doute les seules fonctions que vous aurez à utiliser pour exécuter des bouts de code Perl à partir de votre programme en C. Votre code peut être aussi long que vous désirez ; il peut contenir de nombreuses expressions ; il peut utiliser use in *perlfunc*, require in *perlfunc*, et do in *perlfunc* d'autres fichiers Perl.

perl_eval_pv permet d'évalué des expressions Perl, et extraire les variables pour les transformer en types C. Le programme suivant *string.c*, exécute trois chaînes Perl, extrait un `int` de la première, un <float> de la seconde et un `char *` de la troisième.

```
#include <EXTERN.h>
#include <perl.h>

static PerlInterpreter *my_perl;

main (int argc, char **argv, char **env)
{
    char *embedding[] = { "", "-e", "0" };

    my_perl = perl_alloc();
    perl_construct( my_perl );

    perl_parse(my_perl, NULL, 3, embedding, NULL);
    perl_run(my_perl);

    /** Traite $a comme un entier **/
    perl_eval_pv("$a = 3; $a **= 2", TRUE);
    printf("a = %d\n", SvIV(perl_get_sv("a", FALSE)));

    /** Traite $a comme un flottant **/
    perl_eval_pv("$a = 3.14; $a **= 2", TRUE);
    printf("a = %f\n", SvNV(perl_get_sv("a", FALSE)));

    /** Traite $a comme une chaine **/
    perl_eval_pv("$a = 'rekcaH lreP rehtonA tsuJ'; $a = reverse($a);", TRUE);
    printf("a = %s\n", SvPV(perl_get_sv("a", FALSE), PL_na));

    perl_destruct(my_perl);
    perl_free(my_perl);
}
```

Toutes les fonctions étranges comportant *sv* dans leurs noms aident à convertir les scalaires Perl en types C. Elles sont décrites dans *perlguts*.

Si vous compilez et executez *string.c*,vous pourrez voir les résultats de l'utilisation de *SvIV()* pour créer un `int`, *SvNV()* pour créer un `float` et *SvPV()* pour créer une chaîne :

```
a = 9
a = 9.859600
a = Just Another Perl Hacker
```

Dans l'exemple ci-dessus, nous avons créé une variable globale pour stocker temporairement la valeur calculée par nos expressions eval. Il est aussi possible et c'est dans la plupart des cas la meilleur stratégie de récupérer la valeur de retour à partir de *perl_eval_pv()* à la place. Exemple :

```
...
SV *val = perl_eval_pv("reverse 'rekcaH lreP rehtonA tsuJ'", TRUE);
printf("%s\n", SvPV(val,PL_na));
...
```

De cette manière, nous évitons la pollution de l'espace des noms en ne créant pas de variables globales et nous avons aussi simplifié notre code.

2.7 Effectuer des recherches de motifs et des substitutions à partir de votre programme C

La fonction *perl_eval_sv()* nous permet d'évaluer des bouts de code Perl, nous pouvons donc définir quelques fonctions qui l'utilisent pour créer des fonctions « spécialisées » dans les recherches et substitutions : *match()*, *substitute()*, et *matches()*.

```
I32 match(SV *string, char *pattern);
```

Étant donné une chaîne et un motif (ex., `m/clasp/` ou `/\b\w*\b/`, qui peuvent apparaître dans votre programme C comme "/\\b\\w*\\b/"), match() retourne 1 Si la chaîne correspond au motif et 0 autrement.

```
int substitute(SV **string, char *pattern);
```

Étant donné un pointeur vers un `SV` et une opération `=~` (ex., `s/bob/robert/g` ou `tr[A-Z][a-z]`), substitute() modifie la chaîne à l'intérieur de l'`AV` en suivant l'opération, retournant le nombre de substitutions effectuées.

```
int matches(SV *string, char *pattern, AV **matches);
```

Étant donné un `SV`, un motif et un pointeur vers un `AV` vide, matches() évalue `$string =~ $pattern` dans un contexte de tableau et remplit *matches* avec les éléments du tableau, retournant le nombre de correspondances trouvées.

Voici un exemple, *match.c*, qui les utilise tous les trois :

```
#include <EXTERN.h>
#include <perl.h>

/** my_perl_eval_sv(code, error_check)
** une sorte de perl_eval_sv(),
** mais nous retirons la valeur de retour de la pile.
**/
SV* my_perl_eval_sv(SV *sv, I32 croak_on_error)
{
    dSP;
    SV* retval;

    PUSHMARK(SP);
    perl_eval_sv(sv, G_SCALAR);

    SPAGAIN;
    retval = POPs;
    PUTBACK;

    if (croak_on_error && SvTRUE(ERRSV))
       croak(SvPVx(ERRSV, PL_na));

    return retval;
}

/** match(chaine, motif)
**
** Utilise pour faire des recherches dans un contexte scalaire.
**
** Retourne 1 si la recherche est reussie; 0 autrement.
**/

I32 match(SV *string, char *pattern)
{
    SV *command = NEWSV(1099, 0), *retval;

    sv_setpvf(command, "my $string = '%s'; $string =~ %s",
             SvPV(string,PL_na), pattern);
```

```
    retval = my_perl_eval_sv(command, TRUE);
    SvREFCNT_dec(command);

    return SvIV(retval);
}

/** substitute(chaine, motif)
**
** Utilisee pour les operations =~ qui modifient leur membre de gauche (s/// and tr///)
**
** Retourne le nombre de remplacement effectue
** Note: cette fonction modifie la chaine.
**/

I32 substitute(SV **string, char *pattern)
{
    SV *command = NEWSV(1099, 0), *retval;

    sv_setpvf(command, "$string = '%s'; ($string =~ %s)",
             SvPV(*string,PL_na), pattern);

    retval = my_perl_eval_sv(command, TRUE);
    SvREFCNT_dec(command);

    *string = perl_get_sv("string", FALSE);
    return SvIV(retval);
}

/** matches(chaine, motifs, correspondances)
**
** Utilise pour faire des recherches dans un contexte tableau.
**
** Retourne le nombre de correspondance
** et remplis le tableau correspondance avec les sous-chaines trouvees.
**/

I32 matches(SV *string, char *pattern, AV **match_list)
{
    SV *command = NEWSV(1099, 0);
    I32 num_matches;

    sv_setpvf(command, "my $string = '%s'; @array = ($string =~ %s)",
             SvPV(string,PL_na), pattern);

    my_perl_eval_sv(command, TRUE);
    SvREFCNT_dec(command);

    *match_list = perl_get_av("array", FALSE);
    num_matches = av_len(*match_list) + 1; /** assume $[ is 0 **/

    return num_matches;
}

main (int argc, char **argv, char **env)
{
    PerlInterpreter *my_perl = perl_alloc();
    char *embedding[] = { "", "-e", "0" };
    AV *match_list;
    I32 num_matches, i;
    SV *text = NEWSV(1099,0);
```

```
    perl_construct(my_perl);
    perl_parse(my_perl, NULL, 3, embedding, NULL);

    sv_setpv(text, "When he is at a convenience store and the bill comes to some amount like 76 cents, Ma

    if (match(text, "m/quarter/")) /** Does text contain 'quarter'? **/
       printf("match: Text contains the word 'quarter'.\n\n");
    else
       printf("match: Text doesn't contain the word 'quarter'.\n\n");

    if (match(text, "m/eighth/")) /** Does text contain 'eighth'? **/
       printf("match: Text contains the word 'eighth'.\n\n");
    else
       printf("match: Text doesn't contain the word 'eighth'.\n\n");

    /** Trouve toutes les occurences de /wi../ **/
    num_matches = matches(text, "m/(wi..)/g", &match_list);
    printf("matches: m/(wi..)/g found %d matches...\n", num_matches);

    for (i = 0; i < num_matches; i++)
       printf("match: %s\n", SvPV(*av_fetch(match_list, i, FALSE),PL_na));
    printf("\n");

    /** Retire toutes les voyelles de text **/
    num_matches = substitute(&text, "s/[aeiou]//gi");
    if (num_matches) {
       printf("substitute: s/[aeiou]//gi...%d substitutions made.\n",
              num_matches);
       printf("Now text is: %s\n\n", SvPV(text,PL_na));
    }

    /** Tente une substitution **/
    if (!substitute(&text, "s/Perl/C/")) {
       printf("substitute: s/Perl/C...No substitution made.\n\n");
    }

    SvREFCNT_dec(text);
    PL_perl_destruct_level = 1;
    perl_destruct(my_perl);
    perl_free(my_perl);
}
```

Affiche (les lignes trop longues ont été coupées) :

```
match: Text contains the word 'quarter'.

match: Text doesn't contain the word 'eighth'.

matches: m/(wi..)/g found 2 matches...
match: will
match: with

substitute: s/[aeiou]//gi...139 substitutions made.
Now text is: Whn h s t  cnvnnc str nd th bll cms t sm mnt lk 76 cnts,
Mynrd s wr tht thr s smthng h *shld* d, smthng tht wll nbl hm t gt bck
qrtr, bt h hs n d *wht*.  H fmbls thrgh hs rd sqzy chngprs nd gvs th by
thr xtr pnns wth hs dllr, hpng tht h mght lck nt th crrct mnt.  Th by gvs
hm bck tw f hs wn pnns nd thn th bg shny qrtr tht s hs prz. -RCHH

substitute: s/Perl/C...No substitution made.
```

2.8 Trifouiller la pile Perl à partir de votre programme C

Dans la plupart des livres d'informatique, les piles sont expliquées à l'aide de quelque chose comme une pile d'assiettes de cafétéria : la dernière chose que vous avez posée sur la pile est la première que vous allez en retirer. Ça correspond à nos buts : votre programme C déposera des arguments sur la "pile Perl", fermera ses yeux pendant que quelque chose de magique se passe, et retirera le résultat –la valeur de retour de votre sous-programme Perl– de la pile.

Premièrement, vous devez savoir comment convertir les types C en types Perl et inversement, en utilisant newSViv(), sv_setnv(), newAV() et tous leurs amis. Elles sont décrites dans *perlguts*.

Ensuite vous avez besoin de savoir comment manipuler la pile Perl. C'est décrit dans *perlcall*.

Une fois que vous avez compris ceci, intégré du Perl en C est facile.

Parce que le C ne dispose pas de fonction prédéfinie pour calculer une puissance entière, rendons l'opérateur Perl ** disponible (ceci est moins utile que ça en a l'air, car Perl implémente l'opérateur ** à l'aide de la fonction C *pow()*). Premièrement je vais créer une souche de fonction d'exponentiation dans *power.pl* :

```
sub expo {
    my ($a, $b) = @_;
    return $a ** $b;
}
```

Maintenant je vais écrire un programme C, *power.c*, avec une fonction *PerlPower()* qui contient tous les perlguts nécessaires pour déposer les deux arguments dans *expo()* et récupérer la valeur de retour. Prenez une grande respiration...

```
#include <EXTERN.h>
#include <perl.h>

static PerlInterpreter *my_perl;

static void
PerlPower(int a, int b)
{
  dSP;                            /* initialise le pointeur de pile  */
  ENTER;                        /* tout ce qui est cree a partir d'ici */
  SAVETMPS;                       /* ... est une variable temporaire */
  PUSHMARK(SP);                   /* sauvegarde du pointeur de pile  */
  XPUSHs(sv_2mortal(newSViv(a))); /* depose la base dans la pile     */
  XPUSHs(sv_2mortal(newSViv(b))); /* depose l'exposant dans la pile  */
  PUTBACK;                  /* rend global le pointeur local de pile */
  perl_call_pv("expo", G_SCALAR); /* appelle la fonction             */
  SPAGAIN;                        /* rafraichit le pointeur de pile  */
                           /* retire la valeur de retour de la pile */
  printf ("%d to the %dth power is %d.\n", a, b, POPi);
  PUTBACK;
  FREETMPS;                        /* libere la valeur de retour     */
  LEAVE;                       /* ...et retire les arguments empiles */
}

int main (int argc, char **argv, char **env)
{
  char *my_argv[] = { "", "power.pl" };

  my_perl = perl_alloc();
  perl_construct( my_perl );

  perl_parse(my_perl, NULL, 2, my_argv, (char **)NULL);
  perl_run(my_perl);

  PerlPower(3, 4);                    /*** Calcule 3 ** 4 ***/
```

```
    perl_destruct(my_perl);
    perl_free(my_perl);
}
```

Compiler et exécuter :

```
% cc -o power power.c `perl -MExtUtils::Embed -e ccopts -e ldopts`

% power
3 to the 4th power is 81.
```

2.9 Maintenir un interpréteur persistant

Lorsque l'on développe une application interactive et/ou potentiellement de longue durée, c'est une bonne idée de maintenir un interpréteur persistant plutôt que d'allouer et de construire un nouvel interpréteur de nombreuses fois. La raison principale est la vitesse : car Perl ne sera alors chargé qu'une seule fois en mémoire.

De toutes façons, vous devez être plus prudent avec l'espace des noms et la portée des variables lorsque vous utilisez un interpréteur persistant. Dans les exemples précédents nous utilisions des variables globales dans le package par défaut `main`. Nous savions exactement quel code sera exécuté, et supposions que nous pourions éviter les collisions de variables et une extension atroce de la table des symboles.

Supposons que notre application est un serveur qui exécutera occasionnellement le code Perl de quelques fichiers arbitraires. Notre serveur n'a plus moyen de savoir quel code il va exécuter. C'est très dangereux.

Si le fichier est fourni à `perl_parse()`, compilé dans un interpréteur nouvellement créé, et subséquemment détruit par `perl_destruct()` après, vous êtes protégés de la plupart des problèmes d'espace de nom.

Une manière d'éviter les collisions d'espace de nom dans ce cas est de transformer le nom de fichier en un nom de package garanti unique, et de compiler le code de ce package en utilisant eval in *perlfunc*. Dans l'exemple ci-dessous, chaque fichier ne sera compilé qu'une seule fois. Ou l'application peut choisir de nettoyer la table des symboles associée au fichier dès qu'il n'est plus nécessaire. En utilisant perl_call_argv in *perlcall*, nous allons appeler le sous-programme `Embed::Persistent::eval_file` contenu dans le fichier `persistent.pl` et lui passer le nom de fichier et le booléen nettoyer/cacher comme arguments.

Notez que le processus continuera de grossir pour chaque fichier qu'il utilisera. De plus, il peut y avoir des sous-programme `AUTOLOAD` et d'autres conditions qui peuvent faire que la table de symboles Perl grossit. Vous pouvez vouloir ajouter un peu de logique qui surveille la taille du processus, ou qui redémarre tout seul après un certain nombre de requêtes pour être sûr que la consommation mémoire est minimisée. Vous pouvez aussi vouloir limiter la portée de vos variables autant que possible grâce à my in *perlfunc*.

```
package Embed::Persistent;
#persistent.pl

use strict;
use vars '%Cache';
use Symbol qw(delete_package);

sub valid_package_name {
    my($string) = @_;
    $string =~ s/([^A-Za-z0-9\/])/sprintf("_%2x",unpack("C",$1))/eg;
    # Seconde passe pour les mots commencant par un chiffre.
    $string =~ s|/(\d)|sprintf("/_%2x",unpack("C",$1))|eg;

    # Le transformer en nom de package reel
    $string =~ s|/|::|g;
    return "Embed" . $string;
}

sub eval_file {
    my($filename, $delete) = @_;
    my $package = valid_package_name($filename);
    my $mtime = -M $filename;
```

```
    if(defined $Cache{$package}{mtime}
       &&
       $Cache{$package}{mtime} <= $mtime)
    {
       # nous avons deja compile ce sous-programme,
       # il n'a pas ete mis-a-jour sur le disque, rien a faire
       print STDERR "already compiled $package->handler\n";
    }
    else {
       local *FH;
       open FH, $filename or die "open '$filename' $!";
       local($/) = undef;
       my $sub = <FH>;
       close FH;

       #encadre le code dans un sous-programme de notre package unique
       my $eval = qq{package $package; sub handler { $sub; }};
       {
           # cacher nos variables dans ce bloc
           my($filename,$mtime,$package,$sub);
           eval $eval;
       }
       die $@ if $@;

       # le mettre en cache a moins qu'on le detruit a chaque fois
       $Cache{$package}{mtime} = $mtime unless $delete;
    }

    eval {$package->handler;};
    die $@ if $@;

    delete_package($package) if $delete;

    #Si vous voulez voir ce qui se passe
    #print Devel::Symdump->rnew($package)->as_string, $/;
}

1;

__END__

/* persistent.c */
#include <EXTERN.h>
#include <perl.h>

/* 1 = Detruire la table des symboles du fichier apres chaque requete, 0 = ne pas le faire */
#ifndef DO_CLEAN
#define DO_CLEAN 0
#endif

static PerlInterpreter *perl = NULL;

int
main(int argc, char **argv, char **env)
{
    char *embedding[] = { "", "persistent.pl" };
    char *args[] = { "", DO_CLEAN, NULL };
    char filename [1024];
    int exitstatus = 0;
```

```
    if((perl = perl_alloc()) == NULL) {
       fprintf(stderr, "no memory!");
       exit(1);
    }
    perl_construct(perl);

    exitstatus = perl_parse(perl, NULL, 2, embedding, NULL);

    if(!exitstatus) {
       exitstatus = perl_run(perl);

       while(printf("Enter file name: ") && gets(filename)) {

           /* appeler le sous-programme, passer son nom de fichier en argument */
           args[0] = filename;
           perl_call_argv("Embed::Persistent::eval_file",
                          G_DISCARD | G_EVAL, args);

           /* Verifier $@ */
           if(SvTRUE(ERRSV))
               fprintf(stderr, "eval error: %s\n", SvPV(ERRSV,PL_na));
       }
    }

    PL_perl_destruct_level = 0;
    perl_destruct(perl);
    perl_free(perl);
    exit(exitstatus);
}
```

Compilons :

```
% cc -o persistent persistent.c `perl -MExtUtils::Embed -e ccopts -e ldopts`
```

Voici un exemple de fichier script :

```
#test.pl
my $string = "hello";
foo($string);

sub foo {
    print "foo says: @_\n";
}
```

Exécutons :

```
% persistent
Enter file name: test.pl
foo says: hello
Enter file name: test.pl
already compiled Embed::test_2epl->handler
foo says: hello
Enter file name: ^C
```

2.10 Maintenir de multiples instances d'interpréteur

Quelques rares applications nécessitent de créer plus d'un interpréteur lors d'une session. Une telle application peut décider sporadiquement de libérer toutes les ressources associées à l'interpréteur.

Le programme doit s'assurer que ça ait lieu *avant* qu'un nouvel interpréteur soit construit. Par défaut, la variable globale `PL_perl_destruct_level` est positionnée à `0`, puisqu'un nettoyage supplémentaire n'est pas nécessaire lorsqu'un programme n'utilise qu'un seul interpréteur.

Positionner `PL_perl_destruct_level` à `1` rend tout plus propre :

```
PL_perl_destruct_level = 1;

while(1) {
    ...
    /* repositionner les variables globales avec PL_perl_destruct_level = 1 */
    perl_construct(my_perl);
    ...
    /* Nettoie et remet a zero _tout_ pendant perl_destruct */
    perl_destruct(my_perl);
    perl_free(my_perl);
    ...
    /* Recommencons encore et encore ! */
}
```

Lorsque *perl_destruct()* est appelé, l'arbre d'analyse syntaxique et les tables de symboles de l'interpréteur sont nettoyées, et les variables globales sont repositionnées.

Maintenant supposons que nous ayons plus d'une instance d'interpréteur s'exécutant en même temps. Ceci est faisable, mais seulement si le drapeau `-DMULTIPLICITY` a été utilisé lors de la compilation de Perl. Par défaut, cela positionne `PL_perl_destruct_level` à `1`.

Essayons :

```
#include <EXTERN.h>
#include <perl.h>

/* nous allons integrer deux interpreteurs */

#define SAY_HELLO "-e", "print qq(Hi, I'm $^X\n)"

int main(int argc, char **argv, char **env)
{
    PerlInterpreter
        *one_perl = perl_alloc(),
        *two_perl = perl_alloc();
    char *one_args[] = { "one_perl", SAY_HELLO };
    char *two_args[] = { "two_perl", SAY_HELLO };

    perl_construct(one_perl);
    perl_construct(two_perl);

    perl_parse(one_perl, NULL, 3, one_args, (char **)NULL);
    perl_parse(two_perl, NULL, 3, two_args, (char **)NULL);

    perl_run(one_perl);
    perl_run(two_perl);

    perl_destruct(one_perl);
    perl_destruct(two_perl);

    perl_free(one_perl);
    perl_free(two_perl);
}
```

Compilez comme d'habitude :

```
% cc -o multiplicity multiplicity.c `perl -MExtUtils::Embed -e ccopts -e ldopts`
```

Exécutons, exécutons :

```
% multiplicity
Hi, I'm one_perl
Hi, I'm two_perl
```

2.11 Utiliser des modules Perl utilisant les bibliothèques C, à partir de votre programme C

Si vous avez joué avec les exemples ci-dessus et avez essayé d'intégrer un script qui utilise (*use()*) un module Perl (tel que *Socket*) qui lui-même utilise une bibliothèque C ou C++, vous avez probablement vu le message suivant :

```
Can't load module Socket, dynamic loading not available in this perl.
  (You may need to build a new perl executable which either supports
  dynamic loading or has the Socket module statically linked into it.)
```

{Traduction : Ne peut charger le module Socket, le chargement dynamique n'est pas disponible dans ce perl. (Vous avez peut-être besoin de compiler un nouvel exécutable perl qui supporte le chargement dynamique ou qui soit lié statiquement au module Socket.)

Quel est le problème ?

Votre interpréteur ne sait pas communiquer avec ces extensions de son propre chef. Un peu de colle l'aidera. Jusqu'à maintenant vous appeliez *perl_parse()*, en lui passant NULL comme second argument :

```
perl_parse(my_perl, NULL, argc, my_argv, NULL);
```

C'est là que le code de collage peut être inséré pour créer le contact initial entre Perl et les routines C/C++ liées. Jettons un coup d'oeil à *perlmain.c* pour voir comment Perl fait :

```
#ifdef __cplusplus
#  define EXTERN_C extern "C"
#else
#  define EXTERN_C extern
#endif

static void xs_init _((void));

EXTERN_C void boot_DynaLoader _((CV* cv));
EXTERN_C void boot_Socket _((CV* cv));

EXTERN_C void
xs_init()
{
       char *file = __FILE__;
       /* DynaLoader est un cas special */
       newXS("DynaLoader::boot_DynaLoader", boot_DynaLoader, file);
       newXS("Socket::bootstrap", boot_Socket, file);
}
```

Explication : pour chaque extension liée à l'exécutable Perl (déterminé lors de sa configuration initiale sur votre ordinateur ou lors de l'ajout de nouvelles extensions), un sous-programme Perl est créé pour incorporer les routines de l'extension. Normalement, ce sous-programme est nommé *Module::bootstrap()* et est invoqué lors du *use Module*. Tour à tour, il passe par un XSUB, *boot_Module*, qui créé un pendant pour chaque XSUB de l'extension. Ne vous inquiétez pas de cette partie ; laissez ceci aux auteurs de *xsubpp* et des extensions. Si votre extension est chargée dynamiquement, DynaLoader créé au vol *Module::bootstrap()* pour vous. En fait, si vous disposez d'un DynaLoader fonctionnant correctement, il est rarement nécessaire de lier statiquement d'autres extensions.

Une fois que vous disposez de ce code, placez-le en deuxième argument de *perl_parse()* :

```
perl_parse(my_perl, xs_init, argc, my_argv, NULL);
```

Puis compilez :

```
% cc -o interp interp.c `perl -MExtUtils::Embed -e ccopts -e ldopts`

% interp
  use Socket;
  use SomeDynamicallyLoadedModule;

  print "Maintenant je peux utiliser des extensions!\n"'
```

ExtUtils::Embed peut aussi automatiser l'écriture du code de collage *xs_init*.

```
% perl -MExtUtils::Embed -e xsinit -- -o perlxsi.c
% cc -c perlxsi.c `perl -MExtUtils::Embed -e ccopts`
% cc -c interp.c  `perl -MExtUtils::Embed -e ccopts`
% cc -o interp perlxsi.o interp.o `perl -MExtUtils::Embed -e ldopts`
```

Consultez *perlxs* et *perlguts* pour plus de détails.

3 Intégrer du Perl sous Win32

Au moment où j'écris ceci (5.004), il existe deux versions de perl fonctionnant sous Win32. (Ces deux versions fusionnent en 5.005.) S'interfacer avec la bibliothèque Perl d'ActiveState ne se fait pas tout à fait de la même manière que dans les exemples de cette documentation, car de nombreux changements ont été effectués dans l'interface interne de programmation de Perl. Mais il est possible d'intégrer le noyau Perl d'ActiveState. Pour les détails, jetez un oeil à la FAQ Perl pour Win32 à http://www.perl.com/perl/faq/win32/Perl_for_Win32_FAQ.html.

Avec le Perl "officiel" version 5.004 ou plus, tous les exemples de ce document pourront être compilés et exécutés sans modification, même si le processus de compilation est plutôt différent entre Unix et Win32.

Pour commencer, les BACKTICKs ne fonctionne pas sous l'interpréteur de commandes natif de Win32. Le kit ExtUtils::Embed du CPAN est fourni avec un script appelé **genmake**, qui génère un simple makefile pour compiler un programme à partir d'un code source C. Il peut être utilisé de cette manière :

```
C:\ExtUtils-Embed\eg> perl genmake interp.c
C:\ExtUtils-Embed\eg> nmake
C:\ExtUtils-Embed\eg> interp -e "print qq{I'm embedded in Win32!\n}"
```

Vous pouvez vouloir utiliser un environnement plus robuste tel que Microsoft Developer Studio. Dans ce cas, exécutez la commande suivante pour générer perlxsi.c :

```
perl -MExtUtils::Embed -e xsinit
```

Créez un nouveau projet et Insérer -> Fichier dans le projet: perlxsi.c, perl.lib, ainsi que votre propre code source, par ex. interp.c. Typiquement vous pourrez trouver perl.lib dans **C:\perl\lib\CORE**, sinon, vous pourrez trouver le répertoire **CORE** relatif à `perl -V:archlib`. Le studio devra aussi connaître ce chemin pour qu'il puisse trouver les fichiers d'en-têtes Perl. Ce chemin peut être ajouté par le menu Tools -> Options -> Directories. Puis sélectionnez Build -> Build interp.exe et vous pourrez y aller.

4 MORALITE

Vous pouvez quelquefois *écrire du code plus rapide* en C, mais vous pourrez toujours *écrire plus rapidement du code* en Perl. Puisque vous pouvez utiliser l'un avec l'autre, combinez-les comme vous le désirez.

5 AUTEUR

Jon Orwant *<orwant@tpj.com>* and Doug MacEachern *<dougm@osf.org>*, with small contributions from Tim Bunce, Tom Christiansen, Guy Decoux, Hallvard Furuseth, Dov Grobgeld, and Ilya Zakharevich.

Doug MacEachern a écrit un article sur le sujet dans le Volume 1, Issue 4 du The Perl Journal (http://tpj.com). Doug est aussi l'auteur de l'intégration Perl la plus utilisée : mod_perl (perl.apache.org), qui intègre Perl au serveur web Apache. Oracle, Binary Evolution, ActiveState, et le nsapi_perl de Ben Sugars ont utilisé ce modèle pour Oracle, Netscape et les extension Perl de Internet Information Server.

July 22, 1998

6 COPYRIGHT ORIGINAL

7 TRADUCTION

7.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.005_02. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

7.2 Traducteurs

Marc Carmier *<carmier@immortels.frmug.org>*

7.3 Relecture

Julien Gilles, Gérard Delafond

8 À propos de ce document

Ce document est la traduction française du document orginal distribué avec perl. Vous pouvez retrouver l'ensemble de la documentation française Perl (éventuellement mise à jour) en consultant l'URL <http://perl.enstimac.fr/>.

Ce document PDF a été produit Paul Gaborit. Si vous utilisez la version PDF de cette documentation (ou une version papier issue de la version PDF) pour tout autre usage qu'un usage personnel, je vous serai reconnaissant de m'en informer par un petit message <mailto:Paul.Gaborit@enstimac.fr>.

Si vous avez des remarques concernant ce document, en premier lieu, contactez la traducteur (vous devriez trouver son adresse électronique dans la rubrique TRADUCTION) et expliquez-lui gentiment vos remarques ou critiques. Il devrait normalement vous répondre et prendre en compte votre avis. En l'absence de réponse, vous pouvez éventuellement me contacter.

Vous pouvez aussi participer à l'effort de traduction de la documentation Perl. Toutes les bonnes volontés sont les bienvenues. Vous devriez trouver tous les renseignements nécessaires en consultant l'URL ci-dessus.

Ce document PDF est distribué selon les termes de la license Artistique de Perl. Toute autre distribution de ce fichier ou de ses dérivés impose qu'un arrangement soit fait avec le(s) propriétaire(s) des droits. Ces droits appartiennent aux auteurs du document original (lorsqu'ils sont identifiés dans la rubrique AUTEUR), aux traducteurs et relecteurs pour la version française et à moi-même pour la version PDF.