Extension de la bibliothèque standard du C

Première partie : Interface de vérification des débordements

par Nicolas Joseph (home) (Blog)

Date de publication : 22 Septembre 2006

I - Introduction

L'annonce a été publiée le 20 Septembre sur le newsgroup compt.stc.c, un nouveau rapport technique (Technical Report, TR) a été accepté par le comité de normalisation (l'ISO). Il répondait au doux nom de TR 24731 ou *Extensions to the C Library, Part I: Bounds-checking interfaces* (Extensions de la bibliothèque C, Partie 1 : interfaces de vérification des débordements).

Cette extension se base sur la norme C99 (**Les nouveautés du C99**) et a pour but d'étendre la bibliothèque standard du C (on s'en doutait un peu :D).

Sans plus tarder, je vais vous exposer les nouveautés présentées dans cette première extension qui s'attarde sur les problèmes de débordements des tableaux.

II - But de cette extension

Voici les prétentions clairement exposées en introduction de cette extension de la norme :

- Atténuer certaines failles de sécurités : limiter les possibilités de buffer overflow, les failles de type format string, ... (1)
- Sécuriser la manipulation des chaînes de caractères : ne pas produire de chaîne de caractères sans caractère de fin ou des chaînes de caractères tronquées de manière inattendue
- Fournir une bibliothèque utile pour le code existant en réduisant les efforts nécessaires pour migrer du code existant
- Supporter du code ré-entrant (thread-safe)
- Système de nommage logique : les fonctions de ce rapport ont un nom qui se termine par _s
- Avoir un format uniforme pour les paramètres des fonctions et leur type de retour

III - Comment savoir si mon compilateur implémente ces nouveautés ?

Les compilateurs conforment à ce rapport, définissent la macro __STDC_LIB_EXT1__ correspondant à la valeur 200509L.

Ce document étant une extension de la norme, il n'y a aucune obligation pour les compilateurs de l'implémenter.

IV - Comment utiliser ces nouvelles fonctions ?

Comme toutes les fonctions de la bibliothèque standard, elles sont définies dans les fichiers d'en-tête (ce rapport n'ajoute pas de nouveau fichier d'en-tête), par contre il se peut que les fonctions ne soit pas déclarées par défaut par l'implémentation.

Pour être sûr que les fonctions soit déclarées, il faut définir la macro __STDC_WANT_LIB_EXT1__ et lui donner la valeur 1. Si vous lui donnez la valeur 0, les fonctions ne seront pas définies.

Si vous définissez cette macro pour un fichier d'en-tête dans une unité de compilation, elle doit avoir la même valeur dans le reste de votre programme si vous incluez à nouveau ce fichier d'en-tête, sinon le préprocesseurs générera une erreur.

V - errno enfin obsolète !

C'est l'une des bonnes surprises de ce TR (2) , la variable globale permettant une gestion précise des erreurs n'est plus nécessaire, puisque désormais les fonctions utilisant errno retourne une variable de type **errno_t** (un simple entier défini dans errno.h).

Afin de récupérer le message correspondant à la valeur d'une erreur, vous pouvez utiliser la fonction :

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strerror_s(char *s, rsize_t maxsize,
     errno_t errnum);
```

Et pour éviter d'obtenir un message tronqué, vous pouvez utiliser la fonction :

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
size_t strerrorlen_s(errno_t errnum);
```

Pour connaître la taille de la chaîne de caractères.

VI - Un nouveau type pour représenter les tailles

Dans le but d'harmoniser les paramètres des fonctions, un nouveau type est utilisé pour préciser la taille : **rsize_t** qui est un simple typedef de **size_t**.

A la différence de **size_t**, les paramètres de type **rsize_t** sont protégés contre les dépassements de capacité (conversion d'un nombre négatif en **unsigned int** pouvant entraîner une attaque de type buffer overflow). Cette protection s'effectue grâce à la macro RSIZE_MAX qui représente la taille limite raisonnable pour un objet.

VI - Un remplaçant sûr pour gets !

gets est un bug à elle seule puisque qu'il n'est pas possible de contrôler la taille de la chaîne fournie par l'utilisateur. Son remplacement par *fgets* n'est pas intuitif pour les débutants (la notion de fichier n'étant pas forcément connue), voici donc une version sécurisée :

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
char *gets_s(char *s, rsize_t n);
```

Qui est équivalent à :

```
fgets (s, sizeof (s), stdin);
```

VIII - Les exceptions

Oui vous avez bien lu exception ! Et ce n'est pas une image (comme le système mise en place en C99 avec les opérations sur les nombres à virgule flottante), il s'agit de véritables exceptions levées par ces nouvelles fonctions.

Par exemple, la fonction *sprintf_s*, dans le cas où la conversion dépasse la capacité du tableau de caractères passé en paramètre, plutôt que de tronquer le résultat comme *snprintf_s* le fait, elle lève une exception.

Comment gérer ces exceptions ? Le système mis en place n'a rien à voir avec ce que l'on peut rencontrer en C++, mais est plus proche du système de signaux déjà existant en C.

Il faut donc commencer par définir une fonction qui se chargera de gérer les exceptions, elle doit être de la forme :

```
typedef void (*constraint_handler_t)(
     const char * restrict msg,
     void * restrict ptr,
     errno_t error);
```

Où *msg* est un message décrivant l'exception, *ptr* est un pointeur NULL ou sur un objet défini par l'implémentation et enfin *errno* peut être renseigné si la fonction qui à levée l'exception retourne **errno_t**.

Vous pouvez bien sûr lever une exceptions grâce aux fonctions :

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void abort_handler_s(
     const char * restrict msg,
     void * restrict ptr,
     errno_t error);

void ignore_handler_s(
     const char * restrict msg,
     void * restrict ptr,
     errno_t error);
```

IX - Et c'est tout ?

Pour les grandes nouveautés, oui (je vous rappelle que ce n'est que la première partie de l'extension !), mais comme je n'ai pas envie de faire un catalogue de prototypes de fonctions (en plus le TR le fait sûrement mieux que moi), je me suis limité aux grandes lignes, mais toutes les fonctions à risques ont été revues, par exemple pour les chaînes de caractères (*strcat_s*, *strcpy_s*, *strlen_s*, ...), les fonctions appliquées aux caractères étendus (*wchar_t*, introduit en C99) et aussi les fonctions non ré-entrantes (c'est le cas d'une grande partie des fonctions de gestion des dates).

Avant de conclure, voici tout de même une liste des modifications qui ont été apportées à l'ensemble des fonctions :

- Plutôt que de renvoyer un résultat invalide (par exemple NULL pour *fopen*) les nouvelles fonctions privilégies le retour par pointeur et réserve la valeur de retour pour le code d'erreur de type **errno_t**
- La taille des tableaux modifiés est vérifiée avec un second paramètre de type **rsize_t**. Le cas le plus marquant est la famille de fonctions de *scanf_s* qui, pour les conversions de type **s**, **c** et **[** prennent une paire d'arguments : le premier est le même que pour *scanf* et le second est la taille de la variable
- En cas d'erreur (par exemple, un paramètre de type **rsize_t** supérieur à RSIZE_MAX), une exception est levée
- Les fonctions non ré-entrantes ont été revues (par exemple *asctime*) en s'inspirant des fonctions disponibles sous OpenBSD (*asctime_r* dans notre exemple)
- Utilisation intempestive du mot clés resctrict (introduit en C99)
- Pour les fonctions de la famille de *printf_s*, le spécificateur de format %n n'est plus supporté (il peut être à l'origine de faille de type **format string**).

X - Conclusion

Comme toutes les nouveautés, c'est à l'usage que l'on pourra savoir si elles sont vraiment efficaces mais de prime abord les programmes écrits en C devraient être plus sécurisés.

Cette extension venant d'être publiée, cet article est entièrement basé sur le TR. cependant le comité de normalisation a fait l'effort de se baser sur du code existant (POSIX ou les fonctions OpenBSD, par exemple) et les nouveautés étant relativement simples à implémenter (il s'agit surtout de fonctions à implémenter, contrairement au C99), on peut espérer voir apparaître rapidement des compilateurs conformes.

XI - Références

n1146.pdf

n1147.pdf

XII - Remerciements

Merci à **fearyourself** pour sa relecture attentive de cet article.

1 : Pour plus d'information à ce sujet : **Les pièges du C**
2 : même si c'est exagéré de dire que errno est obsolète puisque qu'elle est toujours utilisée par les fonctions actuelles de la bibliothèque.