

Développement C/C++ avec Insure++ et CodeWizard



Marc Lalonde (VISI)

12 février 2002

Plan de la présentation

- Introduction - objectifs visés
- Insure++
 - Introduction - principe de base
 - Utilisation
 - Concurrence
 - Chaperon / autres gadgets
- CodeWizard
 - Introduction - principe de base
 - Règles
 - Exemple d'utilisation
- Conclusion

Introduction

- Deux outils d'analyse de code (achetés récemment) seront présentés: Insure++ (avec Chaperon) et CodeWizard;
- Insure++ et Chaperon: utiles principalement pour la détection d'erreurs de mémoire (pointeurs);
- CodeWizard: utile pour vérifier si le code source respecte des règles de codage.

Objectifs visés

- Familiariser les développeurs de la R-D (et autres intervenants soucieux de « qualité logicielle ») avec ces outils potentiellement très utiles;
- Rentabiliser l'achat en donnant l'information nécessaire à leur utilisation.

Insure++ (version 6.0)

- Produit de Parasoft (www.parasoft.com)
- Disponible pour :
 - WinNT/2000
 - Linux
 - Autres Unix (DEC, HPUX, AIX, Irix, Solaris)
- 1 licence installée sur nightowl (linux):
 - `/usr/local/parasoft/bin.linux2/insure`
- Documentation:
 - `/usr/local/parasoft/manuals/unix_manual.pdf`

Insure++: principe de base

- Principe: instrumentation du code source (i.e. modifications permettant la surveillance du programme pendant l'exécution);
- À la compilation, du code supplémentaire est inséré dans le source original, des vérifications sont faites (p. ex. variables non initialisées), etc.;
- À l'exécution, les allocations / libérations de mémoire sont « enregistrées », chaque manipulation de pointeur est vérifiée, etc.

Insure++: utilisation

- À l'utilisation (*build*), plutôt que d'invoquer le compilateur, on fait:

```
$ insure -c -g <autres options> toto.c
```

- Insure++ instrumente le code original et fait compiler le code modifié par le compilateur;
- Les erreurs relevées sont affichées dans une fenêtre;
- Une fois le code compilé et lié, on le fait exécuter; les erreurs sont affichées dans la fenêtre.

Insure++: exemples d'utilisation

- Détection de:
 - fuites de mémoire
 - utilisation de pointeurs « fous » en lecture/écriture
 - utilisation de pointeurs pendouillants (vers une zone de mémoire libérée)
 - emploi de variables inutilisées (même si initialisées)
 - Vérification des appels système
 - etc.

Insure++: autre fonctionnalité

- Possibilité de définir des interfaces:

```
● /*
   * crun_iic.c
   */
   (double a, double b, double c)
   {
       double ret;

       if(a+b+c = 10.) {
           iic_error(USER_ERROR,
                    "Sum exceeds 10 %f+%f+%f\n",
                    a, b, c);
       }
       if(a <= 0) {
           iic_error(USER_ERROR,
                    "a is negative %f\n", a);
       }
       ret = cruncher(a, b, c);
       if(ret == 0) {
           iic_error(USER_ERROR,
                    "Return zero %f,%f,%f = %f\n",
                    a, b, c, ret);
       }
       return ret;
   }
```

Insure++: concurrence

- Commercial: Rational Purify
 - Travaille à partir du code objet → identifie les instructions qui font un accès à la mémoire;
 - Disponible pour WinNT/2000 et Unix (Solaris, HP-UX) seulement.
- Open source: dmalloc (dmalloc.com)
 - Pour les erreurs d'accès à la mémoire allouée dynamiquement;
 - C'est une bibliothèque; faut travailler un peu pour s'en servir...

Chaperon

- Inconvénient de Insure: faut recompiler le code à vérifier.
- Chaperon (pour Linux seulement) élimine la contrainte en vérifiant l'exécutable directement, au prix d'une vérification plus sommaire qui ne touche que l'accès à la mémoire.
- Utilisation:

```
$ Chaperon a.out
```

Autres gadgets

- Inuse
 - Outil graphique qui permet la visualisation de l'allocation de mémoire pendant l'exécution;
- TCA (Total Coverage Analysis)
 - Insure++ permet de générer des informations de « couverture du code » pendant l'exécution de l'application. L'utilitaire « tca » analyse les informations générées pour les afficher graphiquement, produire un rapport, etc.

CodeWizard (version 4.0)

- Disponible pour
 - WinNT/2000
 - Linux
 - Autres Unix (HPUX, AIX, Solaris, etc.)
- 1 licence installée sur nightowl (linux):
`/usr/local/parasoft/bin.linux2/codewizard`
- Documentation:
`/usr/local/parasoft/manuals/cw_unix_print.pdf`

CodeWizard - principe de base

- Outil d'analyse du code source pour mettre en application des règles de codage C/C++;
- En fait, les règles ne touchent pas seulement le code comme tel mais aussi le design (« good practices »);
- Exemples de règles:
 - C++: déclaration de variables « protected »
 - C/C++: longueur fichier source > xyz lignes
 - C/C++: utilisation de « break » dans une boucle

CodeWizard - règles

- Les règles de base proviennent de:
 - Livres de S. Meyers: « Effective C++ », « More effective C++ »
 - Article de M. Klaus (DDJ, Fév. 97)
 - « Universal Coding Standards »
- Elles peuvent être désactivées au besoin (par ex. on voudrait quand même utiliser l'opérateur « ?: »), ou activées dans certaines situations (classes, fichiers, etc.).

CodeWizard - règles (2)

- Les règles ont des niveaux de priorité (allant de « *informational* » à « *severe violation* »);
- On peut aussi créer des règles personnalisées (avec RuleWizard, un outil graphique).
 - utile pour faire respecter des règles de codage à l'intérieur d'une équipe (p. ex. notation hongroise)

CodeWizard - utilisation

- Invocation:

```
$ codewizard -c -g <options> toto.c
```

- Les résultats de l'analyse s'affichent dans la même fenêtre utilisée par Insure++.

Conclusion

- Deux outils sont disponibles pour faciliter le développement d'applications C/C++ de qualité (autant pour le codage que pour le débogage)
 - Insure++
 - CodeWizard
- Gênez-vous pas pour vous en servir, c'est «
gratis »!
 - Attention au «
timeout » sur la licence!