Outils de développement et compilation

IDE, GCC/Clang, ASAN, perf, valgrind, GDB

Timothée Ravier

LIFO, INSA-CVL, LIPN

1re année cycle ingénieur STI
2013 – 2014

Plan global

Objectifs des outils présentés

- Trouver le maximum de bugs :
  - avant même d'avoir compilé le projet ;
  - lors de la compilation ;
  - lors de l'exécutions des tests.
- Comprendre pourquoi les bugs restant se produisent.

Plan

Environnement de développement intégré

- vim + pulgins / emacs + plugins ;
- KDevelop, QtCreator ;
- Eclipse, NetBeans, IntelliJ IDEA ;
- Anjuta, Code : :Blocks, eric. . .
- Voir la liste et comparaison sur Wikipedia [1].

Plan

Exploiter les options du compilateur

- `$ gcc -Wall -Wextra;`
- `$ clang -Wall -Wextra;`
- Choisir judicieusement le niveau d'optimisation en fonction de l'usage :
  - `-O2` ou `-O3` pour le code final ;
  - `-Og` pour le débug (avec `-g`) ;
  - ...

Plan

Débuggage avec GDB

- Permets d'observer et de manipuler le comportement d'un programme lors de son exécution ;
- Exécution instruction par instruction ou fonction par fonction ;
- Mise en place de point d'arrêt avec ou sans conditions ;
- Affichage des variables de la pile et du tas ;
- Manipulation de la pile d'exécution ;
- Étude d'un core dump ;
- Manipulation des pointeurs, structures...

Débuggage avec GDB

- Outil en ligne de commande avec intégration dans la plupart des IDE (sinon il y a KDbg) ;
- Possible futur remplacement : `LLDB` du projet `LLVM`. À surveiller !

GDB : exemple

- `$ gdb monprog [core_dump]`

```
(gdb) breakpoint mafonctionavecbug
(gdb) run
...
(gdb) backtrace
(gdb) print p
(gdb) info locals
```

Plan

Clang

- Analyseur statique (sans exécution) intégré au compilateur ;
- Penser à activer le plus possible d'options à la compilation pour lever de potentielles erreurs ;
- Utiliser les dernières version pour profiter des améliorations de l'analyseur même si le code du projet ne sera pas nécessairement compilé avec le dernier compilateur à la fin ;
- Utiliser plusieurs compilateurs.

Coverity

- Outils d'analyse statique en ligne ;
- Gratuit pour les projets open source sinon payant ;
- Analyse le code d'un projet et y cherche des erreurs potentielles ;
- Étude récente sur les résultats obtenus avec Coverity [2].

Plan

Collection d'outils

- Machine virtuelle qui utilise la compilation à la volée (Just in time ou JIT) ;
- Aucune instruction du programme originel n'est exécutée directement sur le processeur ;
- Permet de trouver de nombreux types de bugs suivant l'outil utilisé ;
- Avantage : ne nécessite pas de recompilation d'un programme ;
- Désavantage : très lent.

Memcheck

- Utilisation de zone mémoire non-initialisées ;
- Lecture/écriture dans une zone mémoire après qu'elle ai été libérée (use after free) ;
- Lecture/écriture en dehors d'une zone mémoire allouée par malloc ;
- Fuite de mémoire ;
- ...

Autre outils

- Massif : étude de l'usage (profiling) du tas. Voir l'interface Massif Visualizer ;
- Helgrind et DRD : détecte les situations de compétition (race conditions) dans les programmes multithreadés ;
- Cachegrind : étude de l'usage des caches. Voir l'interface KCacheGrind ;
- Callgrind : analyseur de graphe d'appel de fonctions. Voir l'interface KCacheGrind ;

Autres outils (expérimental)

- exp-sgcheck : détecter les erreurs de dépassement de tableaux pour les variables globales et sur la pile qui ne peuvent pas être trouvés par Memcheck. Attention aux faux positifs ;
- exp-dhat : analyse dynamique de l'usage du tas ;
- exp-bbv : simulateur de performance d'un programme qui extrapole un résultat à partir d'un ensemble de mesures.

Plan

AddressSanitizer : a fast memory error detector

- Outil de détection d'erreurs liées à l'usage de la mémoire en C/C++ ;
- Il peut détecter :
  - Use after free (dangling pointer dereference) ;
  - Heap buffer overflow ;
  - Stack buffer overflow ;
  - Global buffer overflow ;
  - Use after return ;
  - Initialization order bugs.
- Disponible avec GCC et Clang.

Avantages & inconvénients

- Avantages :
  - Beaucoup plus rapide que Valgrind (memcheck).
- Désavantages :
  - Recompilation du programme à étudier nécessaire ;
  - Le code est légèrement modifié, on n'exécute pas exactement le même code que sans ASan. Il faut donc passer les tests deux fois au minimum.

Plan

perf

- Outils d'analyse des performances ;
- Analyse statistique du système complet (espace utilisateur et noyau) ;
- Supporte les compteurs de performance matériel, les points de traçage les compteurs de performance logiciels, les sondes dynamiques (kprobes et uprobes)…
- Ne pas oublier de compiler son programme et ses bibliothèques avec les symboles de débug.
- Avantage : impact limité sur les performances.

perf : commandes

- stat : mesure l'ensemble des évènements produits par un programme ou le système entier sur une durée donnée ;
- top : vue similaire à l'outil top des fonctions les plus consommatrices en temps d'exécution ;
- record : mesure et enregistre des échantillons de résultats pour un programme précis ;
- report : analyse un fichier de résultats généré par `perf record`. Possibilité de générer des graphe d'appel de fonctions.
- annotate : lis les données enregistrées `perf record` et annote le code source d'un programme ;

perf : commandes

- sched : informations sur l'ordonnanceur et la latence ;
- list : liste les évènements supportés sur un système.

Plan

Intégration continue (Continuous Integration)

- Maintient d'un dépôt central pour le projet ;
- Tous les développeurs travaillent avec cette version du projet et intègrent leur changements régulièrement ;
- Automatisation des étapes de la construction du projet ;
- Automatisation des tests du projet et lancement automatique à la suite de chaque construction ;
- (Optionnel mais vivement recommandé) Tous les commits sont validés avant d'être intégrés ;
- Parmi ces validations, il faut que le projet soit construit et testé pour chaque nouveau commit avant leur inclusion dans la branche principale ;

Intégration continue (Continuous Integration)

- La construction du projet doit rester une étape rapide pour garder le cycle modification / construction / test / debug le plus court possible ;
- Si possible, les tests doivent être fait dans un environnement reproduisant ce qui est utilisé en production ;
- Il doit être facile de récupérer la dernière version construite du projet (mise à dispositions d'archives ou de paquets « nightly ») ;
- Tout le monde à accès aux résultat des dernières construction du projet ;
- Le déploiement en production doit être automatisé (inclue les procédures de sauvegarde de l'environnement).

Avantages

- Les bugs et les erreurs rencontrées à l'exécution des tests sont rapportées avant l'inclusion définitive du code dans la branche principale du projet ;
- Les développeurs détectent et fixent les bugs continuellement et le plus tôt possible, ce qui facilite significativement le travail puisque le code est frais ;
- Évite le syndrome du commit du vendredi soir qui casse un projet ;
- Test immédiat de tous les changements ;
- Disponibilité permanente d'une version construite du projet pour les tests, les démos ou même la mise en production ;

Avantages

- Retour rapide sur la qualité du code et son fonctionnement et l'impact des changements effectués ;
- Encourage l'évolution progressive et continue d'un projet ;
- Permets de mesurer l'évolution du projet en terme de qualité du code.

Désavantages

- Temps de mise en place significatif nécessaire (sauf si l'on loue ce service à une autre entreprise) ;
- Les tests ne sont efficaces que si ils couvrent bien l'ensemble du projet et sont maintenus à jour.

Exemples

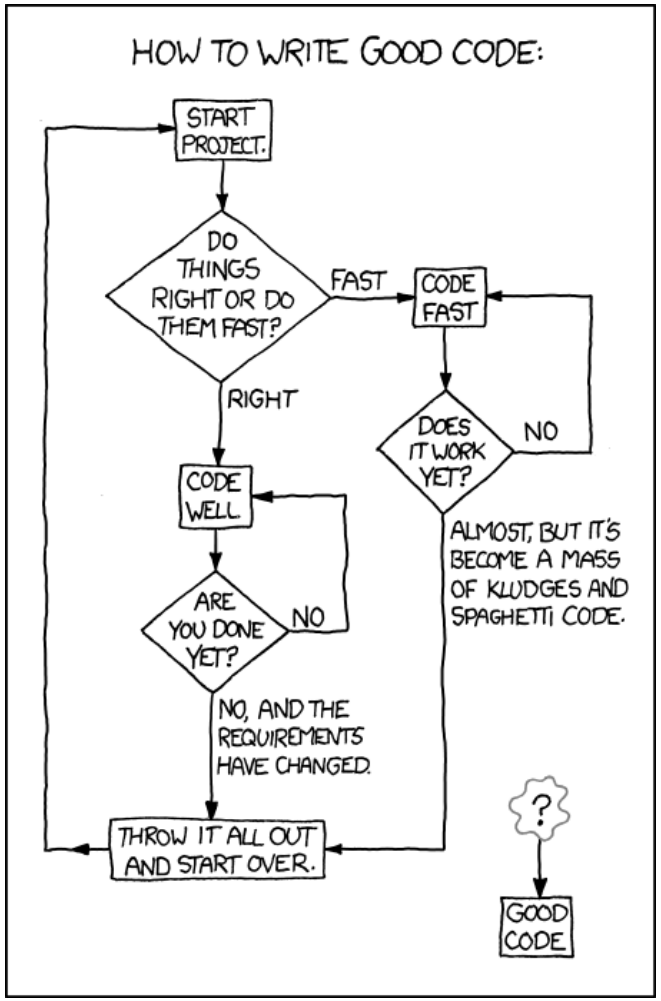
- Gerrit + Jenkins ;
- Github + Travis CI ;
- ...

Plan

Doxygen

- Générateur de documentation sous divers format ;
- S’appuie à la fois sur le code et sur des commentaires spéciaux qui seront interprétés par Doxygen ;
- Encourage la documentation du code simultanément à son écriture ;
- Peut aussi générer des graphes d’appel de fonctions.

Comment produire du bon code ?



https://xkcd.com/844/

Références I

1. Comparison of integrated development environments : http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments
2. Étude sur les erreurs trouvées par Coverity : http://www.zdnet.com/coverity-finds-open-source-software-quality-better-than-proprietary-code-7000028514/