Systèmes d'exploitation

Outils de développement

Chaîne d'outils / Toolchain

L'ensemble d'outils utilisé durant le processus de développement logiciel.

En programmation système, il y a beaucoup de façons différentes de développer, donc de nombreuses chaînes d'outils différentes !

Exemples d'outils:

- Système de compilation (Build System)
- Libraries (Peut dépendre du système d'exploitation)
- Débuggeur
- Outils d'analyse (Profiler, Memory Leak Detector, …)
- IDE (Integrated development environment)

Système de compilation / Build System

L'ensemble d'outils responsible pour l'assemblage de programmes éxecutables.

- Préprocesseur
- Compilateur
- Éditeur de liens (Linker)

Souvent simplement référencé comme le compilateur

- Configuration
- Empaquetage (Packaging)
- Autres activitées apart de la compilation

CMake est un outil qui facilite beaucoup ce processus !

Prétraitement / Preprocessing

Durant la phase de prétraitement, on éffectue des operations sur le code source:

- Substitution de macro
- Inclusion de fichiers
- Compilation conditionnelle

Ce que la directive **#include <fichier>** ou **#include "fichier"** fait:

- **Localisation** du fichier
  - Si on utilise `""`, on utilise le système de fichier
  - Si on utilise `<>`, on utilise les directoires du compilateur (Pour les fichiers standards)
- **Inclusion** des contenues du fichier
- **Récursion** du préprocesseur sur le contenue du fichier

On utilise de la compilation conditionnelle pour s'assurer qu'un ficher n'est pas inclues plus qu'une fois:

- **#pragma once**
- **#ifndef, #define <UNIQUE>, #endif**

Compilation

Durant la phase de compilation, on traduit le code source en code machine.

Fichiers ".c" (source) se transforment en fichiers ".o" (objets ou binaires) après la compilation.

La compilation d'un fichier source est souvent référencée comme une "unité de traduction" (translation unit).

Important: Chaque fichier source est compilé de manière isolée par rapport aux autres fichiers sources.

Liaison / Linking

Durant la phase de liaison, plusieurs fichiers objets sont combinés ensemble pour produire un exécutable final ou une librarie.

Système d'exploitation	Executable	Librarie
Windows	.exe	.lib, .dll (dynamic link library),
Linux	rien ou .out	.a (archive), .so (shared object)

Résout les références, par exemple, la définition du symbole "foo" utilisé dans un fichier objet est trouvée et liée.

Rappel: "undefined reference", "multiple definitions found"

Débuggage

Un déboggeur est un outil qui assiste a trouvé et corrigé des erreurs, bugs, ou autres problèmes avec le programme.

Voici quelques des fonctionnalités clés:

- Définition de points d'arrêt (Break points)
- Définition de points d'arrêt conditionnels (ne pas sous estimé l'importance)
- Exécution pas à pas du code
- Inspection des variables et de la mémoire
- Traçage de la pile

Analyse

Un outil d'analyse permet be mieux comprendre le code.

Statique: Il suffit de lire le code source
Dynamique: Il faut éxecuté le code

Quelques cas d'utilisation:

- Détection de bugs
- Mesure du temp d'execution
- Qualité de code (Detection de mauvaises pratiques)

Très utilisé surtout en C et autres languages système !

Valgrind

Valgrind est un outil d'analyse dynamique qui permet principalement de trouver des erreurs de mémoire.

Valgrind est supporté sur Linux et est intégré dans l'IDE CLion si vous êtes sur Linux.

Nous utilisons Valgrind lors de la correction de vos travaux pratiques.

Valgrind memcheck | CLion Documentation (jetbrains.com)

Sanitizers

Les sanitizers sont similaires à Valgrind, sauf qu'ils sont intégrés dans le compilateur.

On les active en ajoutant un flag au compilateur.

MemorySanitizer implémente un sous-ensemble de Valgrind, disponible seulement avec clang.

Quelques sanitizers importants:

- -fsanitize=address : **AddressSanitizer**, un détecteur d'erreurs de mémoire.
- -fsanitize=leak : **LeakSanitizer**, un détecteur de memory leak.
- -fsanitize=thread : **ThreadSanitizer**, un détecteur de course de données.
- -fsanitize=memory : **MemorySanitizer**, un détecteur de lectures non initialisées. Nécessite l'instrumentation de l'ensemble du code du programme.
- -fsanitize=undefined : **UndefinedBehaviorSanitizer**, un vérificateur de comportements indéfinis rapide et compatible.

Google sanitizers | CLion Documentation (jetbrains.com)

Cmake (Cross-platform make)

Système de compilation qui est indépendent de la platforme.

Souvent désigné comme un générateur de système de compilation, car il produit un système de compilation natif pour la plateforme. Exemples:

- Makefiles pour les systèmes Unix
- Solutions Visual Studio pour Windows
- …

Petite note: Lorsque vous ouvrez un projet CMake dans votre IDE, assurez-vous de l'ouvrir dans le répertoire contenant le fichier CMakeLists.txt, afin que votre IDE puisse trouver le script automatiquement.

IDE (Integrated Development Environment)

Outil qui intègre le toolchain avec l'editeur de code.

Peut contenir beaucoup d'autres outils utiles durant le processus de dévelopement.

Augment souvent la complexité du toolchain. Ce qui n'est souvent pas désirable.

Outils pour le cour

Obligatoire :

- Système : Unix (y compris macOS) ou Linux ou WSL (Windows avec Linux)
- Système de compilation : Clang ou GCC
- Générateur de système de compilation : CMake

Recommandé :

- Système : Linux ou WSL
- Valgrind (utilisé dans votre correction)
- Sanitizers (Surtout si vous utilisez macos)
- IDE CLion

Il est toujours possible d'utiliser les machines du DIRO :)

Demonstration des outils en classe

Installation – CLion – sélection de toolchain – cmake – valgrind – sanitizers