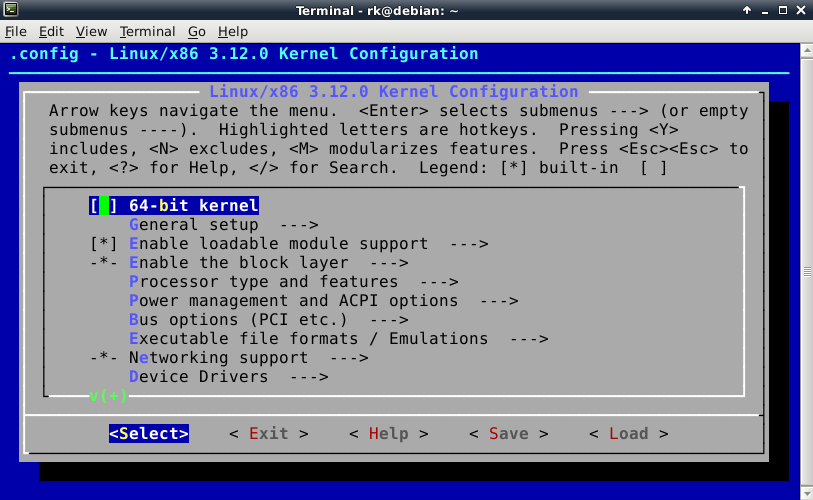
Secure Embedded System (SeS)

U-boot Valgrind Compile kernel



Auteur : Spinelli Isaia

Prof : [Schuler Jean-Roland](https://mse.hes-so.ch/consultation/master/horairesMSE.php?annee=2020&id=81&type=1)

Date : 14.09.2020

Salle : A4 – Lausanne

Classe : SeS

Table des matières

[Introduction - 2 -](#_Toc51057530)

[Références - 2 -](#_Toc51057531)

[Conclusion - 3 -](#_Toc51057532)

[Difficultés rencontrées - 3 -](#_Toc51057533)

[Compétences acquises - 3 -](#_Toc51057534)

[Résultats obtenus - 3 -](#_Toc51057535)

# Introduction

Ce rapport est composé de trois laboratoires distincts :

1. Valgrind
2. U-boot
3. Compile Kernel

# Références

Valgrind :

<http://pub.phyks.me/sdz/sdz/debuguer-facilement-avec-valgrind.html>

<https://en.wikipedia.org/wiki/Valgrind>

<https://www.valgrind.org/>

U-boot :

Compile Kernel :

## Valgrind

Valgrind est un outil de [programmation](https://fr.wikipedia.org/wiki/Programmation_informatique) [libre](https://fr.wikipedia.org/wiki/Logiciel_libre). Il permet de détecter automatiquement de nombreux bogues de gestion de la mémoire, de threading et profiler des programmes en détail.  De nombreux modules tiers ont été écrits pour satisfaire des besoins divers. Voici quelques exemples :

* *Massif :* Profileur du tas et de la pile. L'interface graphique massif-visualizer permet d'afficher les mesures issues de Massif.
* *Cachegrind :* Profileur de la cache et de prédiction de branchement. L'outil graphique KCacheGrind Permet de visualiser les mesures issues de Cachegrind.
* *Callgrind :*  Analyseur de graphe d'appel.  KCacheGrind permet aussi d'exploiter les mesures de cet outil.
* *Memcheck :* Détecteur d’erreur mémoire.

### Introduction

Durant ce chapitre l’outil Valgrind est utilisé afin de rechercher des erreurs et corriger des programmes. Le but est de se familiariser et maitriser cet outil.

### Question 1

Le premier exercice consiste à créer trois petits programmes en C contenant des erreurs afin d’utiliser l’outil Valgrind pour trouver ces erreurs.

#### Programme 1

Le 1er programme comporte une erreur de type « heap overflow » :

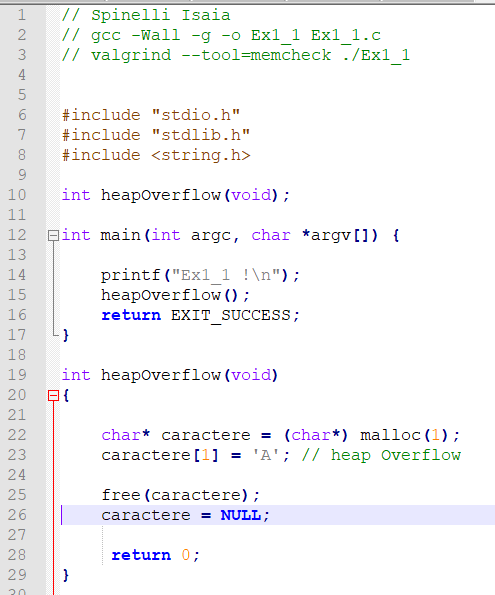


Figure 0‑1 : Premier programme

Ce programme très simple contient une erreur de type heap overflow à la ligne 23. Afin de trouver cette erreur, Valgrind à été utilisé avec l’outil « memcheck » :

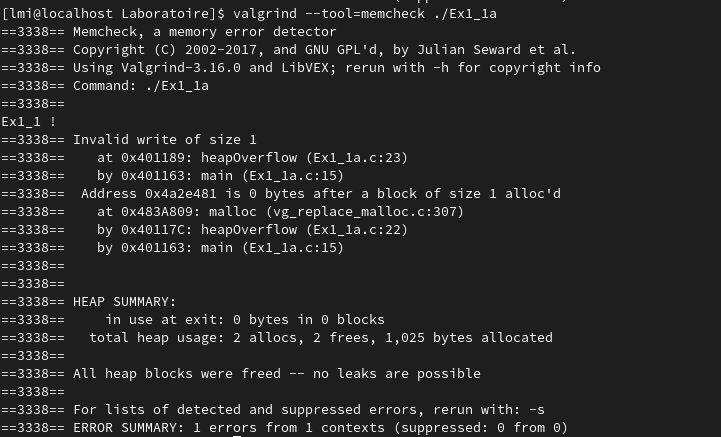


Figure 0‑2 : Sortie du premier programme

La sortie indique correctement qu’il y a une erreur. Une écriture invalide (overflow) de la taille d’un byte a été repéré à la ligne 23.

#### Programme 2

Le 2eme programme comporte une erreur de type « memory leak » :

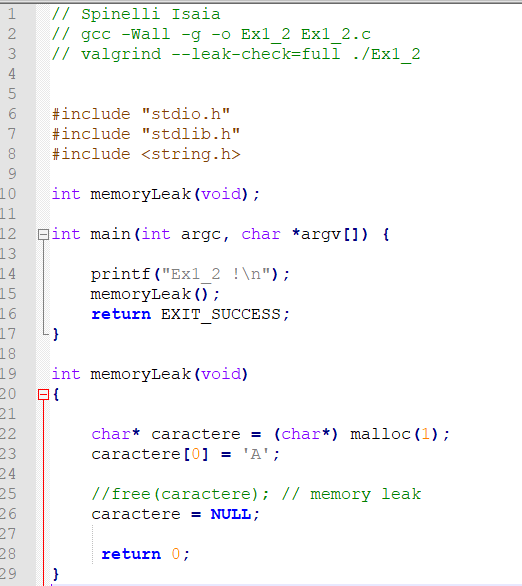


Figure 0‑3 : Deuxième programme

Ce programme est très ressemblant au premier, deux différence ont été apportées :

1. L’erreur de type heap overflow a été corrigé à la ligne 23.
2. Une erreur de type memory leak a été introduite en commentant la ligne 25.

Afin de permettre à Valgrind de détecter ce type d’erreur, l’option « --leak-check=full » de memcheck a été ajouté.

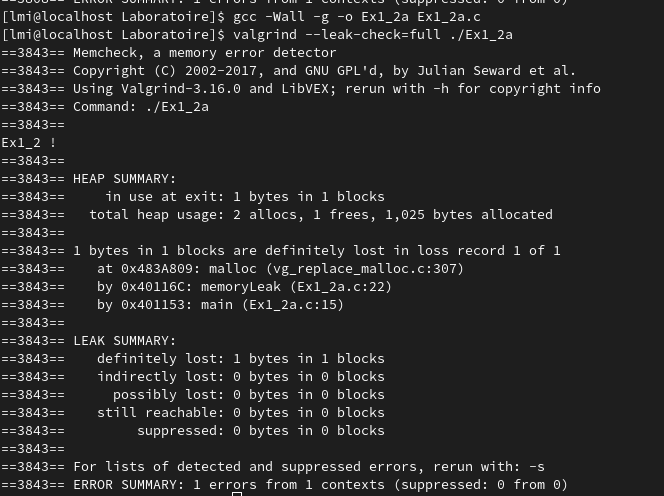


Figure 0‑4 : Sortie du deuxièeme programme

Encore une fois Valgrind a pu détecter l’erreur et indique qu’un byte a été perdu. De plus, il affiche la ligne lors de l’allocation du byte en question (ligne 22).

#### Programme 3

Le 3eme programme comporte une erreur de type « Segmentation » :

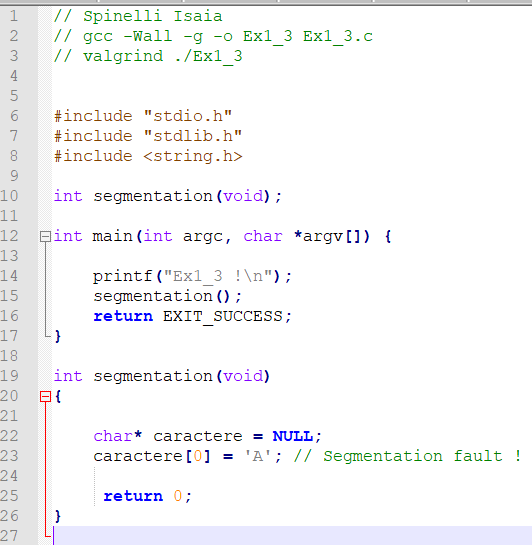


Figure 0‑5 : Troisième programme

Ce programme contient une erreur de segmentation à la ligne 23 en écrivant dans une adresse NULL. On peut voir la sortie du Valgrind ci-dessous :

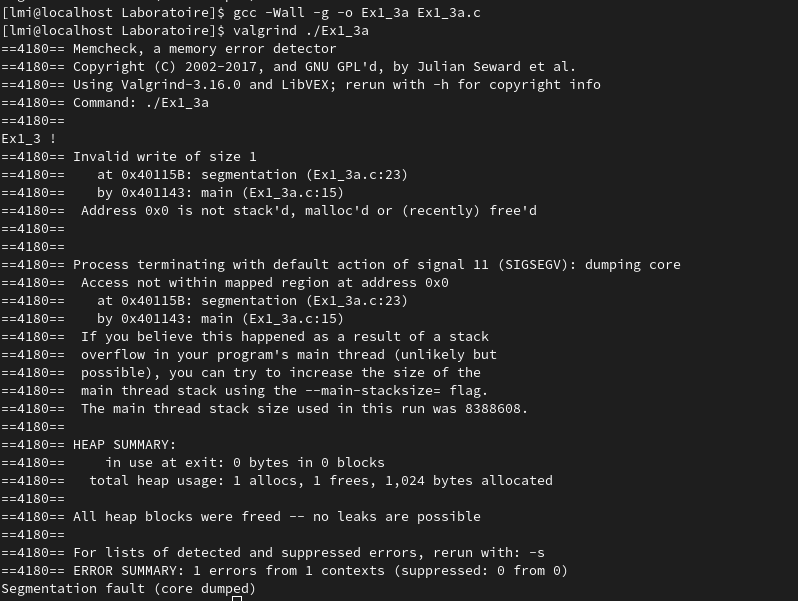


Figure 0‑6 : Sortie du troisième programme

Sans problème Valgrind a pu détecter l’erreur de segmentation tout en indiquant la ligne fautive.

### Limitations

Un quatrième programme a été réalisé afin de tester quelques limite de Valgrind :

### 

Figure 0‑7 : Quatrième programme

Ce dernier programme contient deux erreurs :

1. À la ligne 17, une erreur de dépassement d’un tableau alloué automatiquement.
2. À la ligne 22, la même erreur du programme précèdent dans une condition toujours fausse.

Ci-dessous, la sortie indiqué par Valgrind :

### 

Figure 0‑8 : Sortie du quatrième programme

Cette fois, l’outil ne détecte aucun erreur. On peut donc en conclure que Valgrind est un outil très puissant mais ne permet pas de détecter un dépassement sur un tableau alloué automatique. De plus, il ne vérifie pas les lignes de code dans des conditions toujours fausse.

## Question 2

Pour cette partie, un programme nous a été fourni. Celui-ci est spécialement mauvais est contient donc plusieurs erreurs. Le but est de contrôler et corriger ce programme en utilisant Valgrind.

Ce programme implémente en C une structure nommée « vector\_t » contenant l’adresse d’un tableau et une taille. Plusieurs fonctions permettant la manipulation de ce type de structure ont aussi été implémentées.

## U-boot

## Compile Kernel

# Conclusion

## Difficultés rencontrées

## Compétences acquises

## Résultats obtenus

Date : 02.11.20

Nom de l’étudiant : Spinelli Isaia