This document explains the memory allocation mechanisms which can be meet in MutekH.

Introduction

Memory allocation in an operating system is a large spectral domain...

Glossaire

Pour enlever toutes ambiguïtés, une liste de définition de mot technique est données.

Allocateur mémoire: On appel allocateur mémoire l'additions de deux éléments. D'une part le code executé lors d'un appel à une fonction du type malloc. Et d'autre part l'ensemble des structures de données représentant une zones mémoire allouables.

Utilisateur: L'utilisateur est le code/programe qui demande de la mémoire. Par exemple, un driver qui fait une demande de mémoire au service d'allocation mémoire est l'utilisateur de ce service.

Demande de mémoire: Correspond à l'appel d'une fonction du type malloc (la fonction ne s'appel pas forcément malloc).

Libération de mémoire: Correspond à l'appel d'une fonction du type free (la fonction ne s'appel pas forcément free).

Kernel land/User land

Kernel memory managment

Cette section explique comment le noyau gère son espace mémoire.

Region based memory allocation

Algorithm use for memory allocation

Dans une région, la mémoire est divisée en bloc. Un bloc est divisé en deux partie:
-un header, contenant les informations utilisées par l'allocateur mémoire, de taille fixe.
-les "données", correspondant à l'espace destiné à l'utilisateur, de taille variable.

Un bloc peut être dans un des deux états suivant: libre ou occupé. Un bloc occupé représente une allocation mémoire en cours d'utilisation. Alors qu'un bloc libre représente une zone allouable.

Schéma bloc libre/occupé

Lorsqu'un utilisateur demande de la mémoire, l'allocateur cherche un bloc dans l'état libre, dont la taille de la partie "données" est au moins égal à la taille demandé. Dans le cas d'une recherche positive, le bloc libre trouvé est utilisé pour servir la demande d'allocation mémoire. Son état passe de libre à occupé. L'adresse de début de la partie "données" est fournie à l'utilisateur.

Dans le cas où ce bloc libre possède une partie "données" de taille supérieur à la taille demandé par l'utilisateur, et que le surplus peut contenir un bloc, alors le bloc libre est scindé en deux blocs. Le premier bloc, dont la taille de la partie données est égale à la taille demandé par l'utilisateur, est dans mis dans l'état occupé, et sa partie "données"

est fournie à l'utilisateur. Le second bloc est dans l'état libre, et occupe le surplus de mémoire du bloc initial.

Schéma Passage bloc libre->occupé, avec deux cas (pas scindé et scindé)

La technique utilisé par l'allocateur mémoire pour trouver un bloc libre est maintenant détaillé. MutekH possède deux algorithmes différents pour l'allocateur mémoire.

L'allocateur "simple", dont le but est d'être très léger au niveau du code de l'allocateur. Comme son nom l'indique, cet allocateur est simple. L'opération de libération n'est pas disponible. Il doit être utilisé dans le cas d'une application demandant de la mémoire, mais n'en libérant jamais ou que très rarement. Il s'appuie sur le mécanisme suivant: seul le dernier bloc mémoire est libre. A chaque nouvelle demande de mémoire, ce bloc est scindé en deux: le premier bloc est utilisé pour l'utilisateur, et le second bloc devient le dernier bloc libre.

L'allocateur "smart", devant répondre à des utilisations plus variées. Le paragraphe détaille les aspect techniques de l'allocateur smart.

Smart Allocator

L'allocateur "smart" s'appuie sur deux listes doublement chainées par région. Une liste (block_list) contenant l'ensemble des blocs de la region et une liste contenant l'ensemble des blocs libres (free_list).

Pour une liste, un bloc représente un maillon, et le header de ce bloc contient un pointeur vers l'élément suivant et un pointeur vers l'élément précédent. Un bloc appartient toujours à la liste block_list, et peut aussi, en même temps, appartenir à la liste free_list. Le header de ce block doit donc pouvoir contenir l'ensemble des pointeurs pour ces deux listes, soit 4 pointeurs. Si un bloc est libre, alors les 4 pointeurs sont utilisés. Dans le cas contraire, il ne fait pas parti de la liste free_list, et les deux pointeurs correspondants à cette liste sont utilisables autrement que comme pointeur. Ceci dans un but d'économie de mémoire.

Un des deux pointeurs est utilisé comme un indicateur d'état. Cet indicateur est à 0 dans le cas où le bloc est occupé. Si cet indicateur n'est pas à 0, alors il s'agit d'une pointeur de la liste free_list.

Le second pointeur est utilisé comme un pointeur vers la structure de données décrivant la région.

Un bloc est représenté par la structure de donnée suivante:

```
struct memory_allocator_header_s
{
  union
  {
    CONTAINER_ENTRY_TYPE(CLIST) free_entry;
    struct
    {
      struct memory_allocator_region_s  *region;
      void *free_null_marker;
    };
  };

  CONTAINER_ENTRY_TYPE(CLIST)   block_entry;

#ifdef CONFIG_MUTEK_MEMALLOC_CRC
  uint32_t crc;
#endif
};
```

Aspect multi zone d'une région: Une région n'est pas un bloc contigu. Elle peut être constitué de plusieurs morceaux. Chaque morceau est terminé par un bloc dont la partie "données" est vide, autrement constitué du header

seul. Ce bloc est dans l'état occupé et ne fait pas partie de la région. L'indicateur d'état est à 0, et le pointeur vers la structure de données décrivant la région est aussi à 0.

Standard API (Malloc/Free?)

API overview

Virtual memory allocation

Physical page allocation

Debug tools

-CRC

-Guard Zone

-Scramble

-MemChecker?

Todo

slab allocator debug memory allocator