Allocation mémoire — Fonctions usuelles

M2 SETI B4 / MS SE SE758

Guillaume Duc
guillaume.duc@telecom-paris.fr
2020–2021

Fonctions d'allocation mémoire usuelles

- Allouent de la mémoire dans la zone réservée au noyau
  - Allouent un nombre arbitraire d'octets ou d'une ou plusieurs pages
- Renvoient la première adresse virtuelle de la zone allouée (ou `NULL` en cas d'erreur)
  - Il est très important de vérifier cette valeur de retour, le déréférencement d'un pointeur `NULL` par le noyau pouvant le planter complètement)

Drapeaux

- La plupart des fonctions d'allocation mémoire prennent un argument de type `gfp_t` qui contient des drapeaux permettant de contrôler leur fonctionnement
- La liste des valeurs possibles pour ces drapeaux est donnée dans `include/linux/gfp.h`
- Les valeurs usuelles sont
  - `GFP_KERNEL` : cas le plus courant, la fonction d'allocation est autorisée à mettre le processus courant en veille le temps de réaliser l'allocation
  - `GFP_ATOMIC` : la fonction d'allocation n'est pas autorisée à mettre le processus en veille. À utiliser dans les contextes où il n'est pas permis de bloquer (gestionnaire d'interruption par exemple)

Fonctions classiques (allocation)

```
#include <linux/slab.h>
```

- `void *kmalloc(size_t size, gfp_t flags)` : Alloue `size` octets et retourne l'adresse virtuelle du début de la zone
- `void *kzalloc(size_t size, gfp_t flags)` : Le contenu de la zone mémoire est mis à zéro
- `void *kcalloc(size_t n, size_t size, gfp_t flags)` : Alloue de la mémoire pour un tableau de `n` éléments de `size` octets chacun
- Toutes ces fonctions retournent `NULL` si l'allocation a échoué

Fonctions classiques (libération)

```
#include <linux/slab.h>
```

- `void kfree(const void *p)` : Libère une zone mémoire allouée par `kmalloc`
- `void kzfree(const void *p)` (noyaux $<$ 5.10) ou `void kfree_sensitive(const void *p)` (noyaux $>=$ 5.10) : Efface le contenu de la zone mémoire avant la libération (utilisée principalement lorsque la zone contient des données sensibles)

Fonctions équivalentes `devm_`

`#include <linux/device.h>`

```
void *devm_kmalloc(struct device *dev, size_t size, gfp_t gfp)
void *devm_kzalloc(struct device *dev, size_t size, gfp_t gfp)
void *devm_kcalloc(struct device *dev, size_t n, size_t size, gfp_t flags)
```

- La mémoire allouée par ces fonctions est automatiquement libérée lorsque le périphérique représenté par la structure `struct device *dev` disparaît
- La mémoire peut également être libérée explicitement à l'aide de `void devm_kfree(struct device *dev, void *p)`
- GPL uniquement

Allocation d'une ou plusieurs pages

`#include <linux/gfp.h>`

- Lorsque la zone à allouer est large, ou que l'on a besoin d'une zone contiguë en mémoire physique, il est préférable d'utiliser les fonctions suivantes qui permettent d'allouer une ou plusieurs pages
- `unsigned long get_zeroed_page(gfp_t gfp_mask)` : alloue une page, la remplit avec des zéros et retourne la première adresse virtuelle
- `unsigned long __get_free_page(gfp_t gfp_mask)` : alloue une page (sans réinitialiser son contenu)
- `unsigned long __get_free_pages(gfp_t gfp_mask, unsigned int order)` : alloue 2^{order} pages contiguës en mémoire physique
- Ces fonctions peuvent échouer et renvoyer `0`

Libération d'une ou plusieurs pages

```
#include <linux/gfp.h>
```

- `void free_page(unsigned long addr)` : libère une page précédemment allouée avec `get_zeroed_page` ou `__get_free_page`
- `void free_pages(unsigned long addr, unsigned int order)` : libère les 2^{order} pages allouées précédemment avec `__get_free_pages`
  - Attention, il faut libérer exactement le même nombre de pages que celui qui a été alloué

Allocation d'une ou plusieurs pages

```
#include <linux/device.h>
```

- `unsigned long devm_get_free_pages(struct device *dev, gfp_t gfp_mask, unsigned int order)`
- `void devm_free_pages(struct device *dev, unsigned long addr)`
- Ces deux fonctions sont équivalentes à `__get_free_pages` et `free_pages` mais la mémoire est libérée automatiquement quand le périphérique disparaît

Fonctions diverses

`#include <linux/kernel.h>`

- `char *kasprintf(gfp_t gfp, const char *fmt, ...)` : équivalent de la fonction `asprintf`, construit une chaîne de caractères à partir de la chaîne de format `fmt` (de la même manière que `snprintf`) mais en allouant dynamiquement le tampon de destination
- La mémoire doit être libérée à l'aide de `kfree`
- La variante `char *devm_kasprintf(struct device *dev, gfp_t gfp, const char *fmt, ...)` existe également (GPL uniquement)

Remarques générales

- Toutes les fonctions d'allocation peuvent échouer (même avec le drapeau `GFP_KERNEL`)
  - Il est donc indispensable de vérifier la valeur renvoyée
- N'oubliez pas de libérer la mémoire allouée, sinon vous créez une fuite mémoire qui ne pourra être corrigée qu'au redémarrage
  - Les fonctions `devm_*` constituent un filet de sécurité, les fuites potentielles étant limitées à la durée de vie d'un périphérique