Projet : développement d'un mini système d'exploitation pour PC x86

La pagination

Jérôme Ermont et Emmanuel Chaput

IRIT - Toulouse INP/ENSEEIHT

Fichiers fournis

- `kheap.h` et `kheap.c` : allocation mémoire très rudimentaire
- `mem.h` et `mem.c` : gestion des pages de la mémoire physique
- `paging.h` et `paging.c` : fichiers à compléter, gestion de la pagination

Mise en place du patch

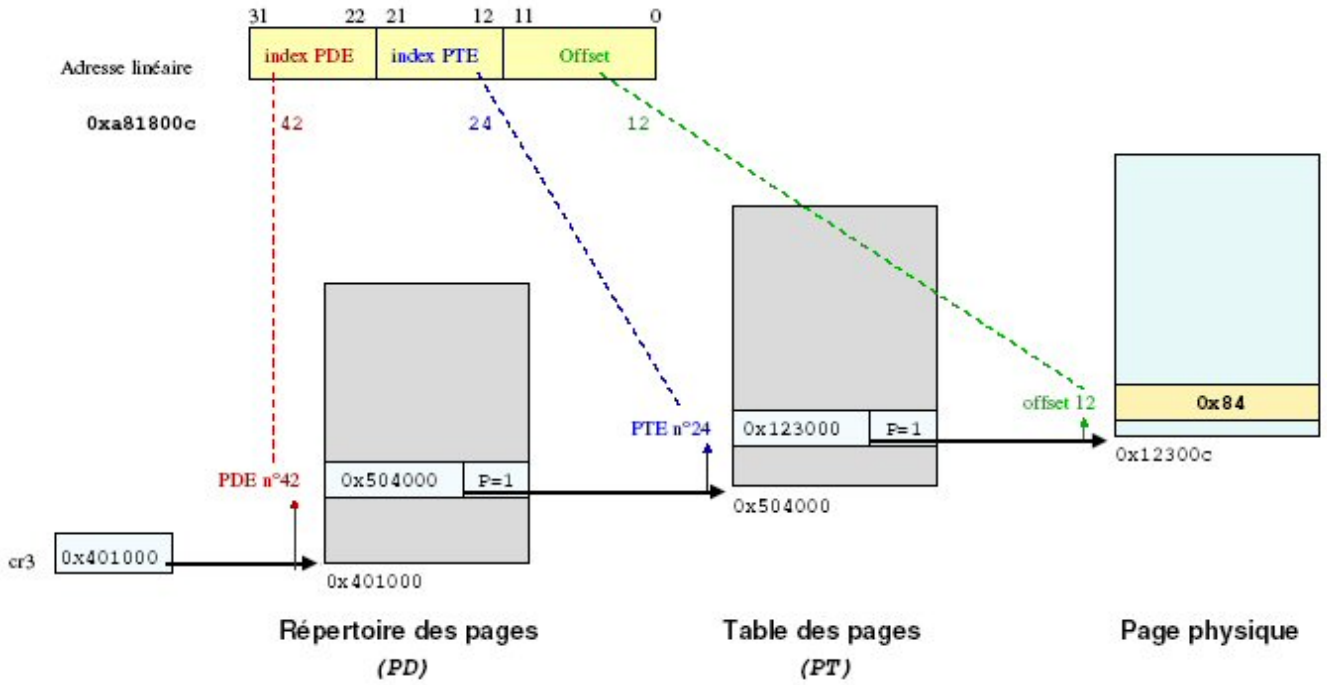
- Récupérer le patch sous Moodle :

  `https://moodle-n7.inp-toulouse.fr/mod/resource/view.php?id=84671`

- Dans le répertoire `n7OS`, appliquer le patch :

```
patch -p1 < chemin/n7OS_vmm.patch
```

La pagination sous x86

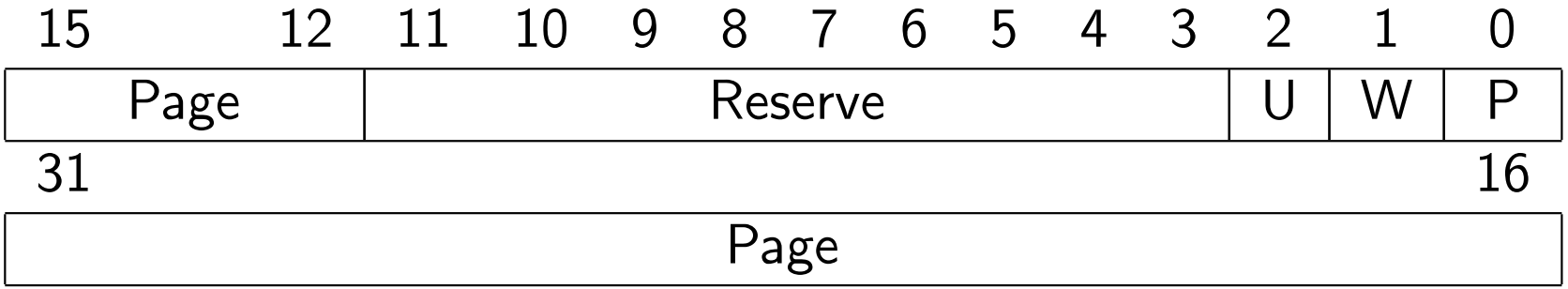


1

- Accès à une page en 3 étapes :
  - Lecture du CR3 pour connaître la position du répertoire de tables dans la mémoire
  - Recherche de la table des pages dans le répertoire de tables
  - Recherche de page dans la tables de pages

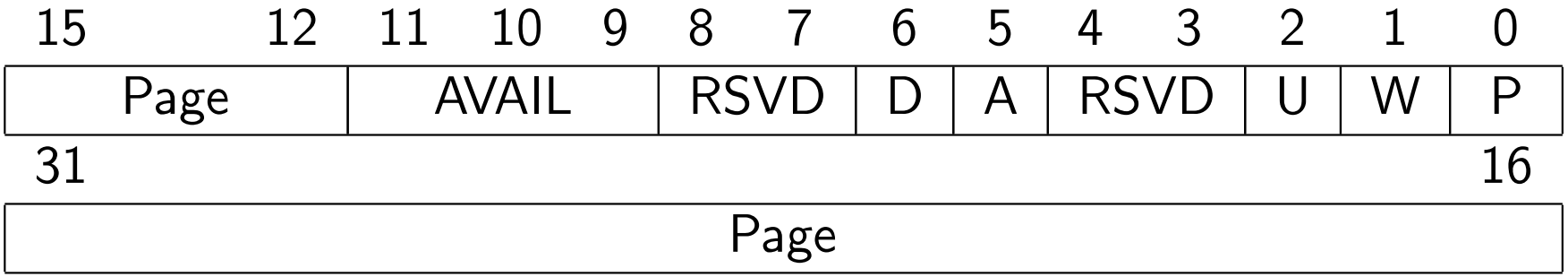
1. `http://sos.enix.org`

Format d'une entrée dans le répertoire de page



- P=1 : table de page présente en mémoire
- W=1 : table de page accessible en lecture/écriture (0 : lecture seule)
- U=1 : page utilisateur ; U=0 : page en mode noyau
- Page : Adresse de la page en mémoire physique (20 bits)

Format d'une entrée dans la table de page



- P=1 : page présente en mémoire
- W=1 : page accessible en lecture/écriture (0 : lecture seule)
- U=1 : page utilisateur ; U=0 : page en mode noyau
- A : Accessed bit
- D : Dirty bit (page modifiée)
- RSVD : bits réservés
- Page : Adresse de la page en mémoire physique (20 bits)

Répertoire de pages / Table de pages

- Le répertoire de pages
  - dans CR3
  - une entrée = 10 bits de poids fort de l'adresse virtuelle
- Les tables de pages
  - une table = une entrée dans le répertoire
  - une entrée de la table = un descripteur de page
  - position de l'entrée = 10 bit
- Une adresse virtuelle :

Les registres de contrôle du x86

- CR0 :
  - bit 31 : enable paging
  - écriture : `lmsw registre` (ou `mov registre, %cr0`)
  - lecture : `smsw registre` (ou `mv %cr0, registre`)
- CR2 :
  - adresse de la page en défaut (sur 32 bits)
- CR3 :
  - adresse du répertoire de page

Activation de la pagination

1. Ecriture de l'adresse du répertoire de page dans le CR3
2. Mise à 1 du bit 31 de CR0

Remarque :

- Peut être réalisé dans un fichier en assembleur
- En utilisant l'assembleur en ligne de GCC :

```
__asm__ __volatile__("mov␣%0,␣%%cr3"::"r"(&dir));
```

Faute de page

- Les erreurs :
  - Ecriture ou lecture d'une zone mémoire non mappée (P=0)
  - Ecriture dans une page en lecture seule
  - Lecture ou écriture d'une page mode noyau par un processus en mode utilisateur
  - Entrée dans la table de page erronée
- Provoque une IT numéro 14
- Retour = code d'erreur :
  - Bit 0 = 0 : la page n'est pas mappée
  - Bit 1 = 1 : page en lecture seule
  - Bit 2 = 1 : page en mode utilisateur
  - Bit 3 = 1 : entrée erronée
- Adresse de la page erronée : contenu de CR2

Allocation mémoire dans le noyau (kheap.h)

```
u32int kmalloc_a(u32int sz);
u32int kmalloc_p(u32int sz, u32int *phys);
u32int kmalloc_ap(u32int sz, u32int *phys);
u32int kmalloc(u32int sz);
```

Exemple d'utilisation

```
PageTable new_page_table= (PageTable)
    kmalloc_a(sizeof(Page_Table_Entry) * 1024);
```

Réserve 1024 lignes mémoire de la taille d'une entrée de table de page (cf trans. 5) et retourne l'adresse de la table de page en mémoire.

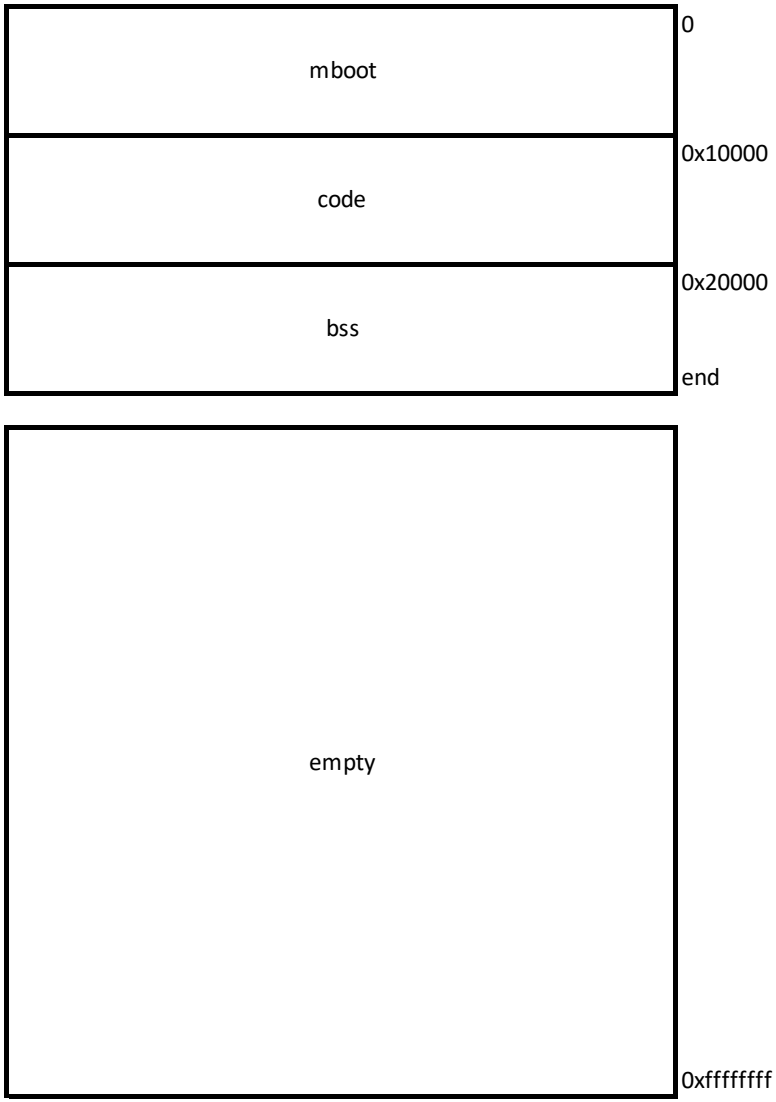
Attention

Cette allocation ne tient pas compte de la mémoire virtuelle mais est utilisée pour la mémoire virtuelle. Elle ne doit pas être utilisée pour faire de l'allocation lorsque la pagination est activée.

Travail à faire

1. Compléter les fichiers `mem.h` et `mem.c` qui gère l'allocation des pages en mémoire physique
2. Compléter les fichiers `paging.h` et `paging.c` (ils sont vides)
3. Créer une fonction `initialise_paging`
   - Alloue une zone mémoire pour stocker le répertoire de page
   - Initialise les tables de pages
   - Charge le répertoire de page dans le CR3
   - Lance la pagination
4. Créer une fonction qui mappe une page libre de la mémoire physique à une page de la mémoire virtuelle
   - Récupère la page de la mémoire virtuelle en fonction de l'adresse recherchée
   - Recherche une page de la mémoire physique libre
   - Associe la page de la mémoire physique trouvée à la page de la mémoire virtuelle dont l'adresse est donnée en argument

Espace d'adresse



Comment trouver une page de la mémoire physique libre ?

- 1ère solution :
  - Un tableau de pages libres
  - `int free_page_table[nbpages]`
  - nbpages = taille mémoire physique/taille d'une page
  - si 0 alors la page est libre

Inconvénient

- Si la taille de la mémoire physique est 16Mo et la taille d'une page est 4ko, alors la mémoire physique peut être découpée en 4096 pages.
- Dans le tableau, il faut stocker 4096 entiers.
- Un entier prend 32 bits.
- Le tableau utilisera 16kO.

Comment trouver une page de la mémoire physique libre ?

- 2ème solution :
  - Utilisation de bitmap
  - 1 bit = 1 page libre
  - Un tableau de bitmap de pages libres
  - `u32int free_page_bitmap_table[size]`
  - size = nbpages / 32
  - Par exemple :
    - nbpages = 4096, size = 128
    - taille du tableau en mémoire : 128x32 = 4kO
  - Initialement :

    `free_page_bitmap_table[0] == 0x0`
  - La page 4 est allouée :

    `free_page_bitmap_table[0] == 0x8 //0b0..01000`

    Le bit 4 est mis à 1 :

    `free_page_bitmap_table[0] = free_page_bitmap_table[0]|(1<<4)`

Exemple de plan d'adressage