

# Système de Gestion de la Mémoire Physique

Ensimag 2A, édition 2020-2021

Ensimag - Grenoble INP

## Déroulement des 3 TPs

- En binôme (1+1). Les mêmes binômes pour les 3 TPs (pas de divorce en cours de route)
- Rendu : le code source en C complet (qui compile, avec les fichiers fournis, tests...) (fichier tar obtenu par `make package_source`)
- Les informations : sujets, transparents, entrepôt git avec les squelettes de code, batterie de test, sont sur <http://ensiwiki.ensimag.fr>,
- Rendu du code source sur Teide exclusivement,
- Application de la Charte des projets de Teide sur les fraudes.

# Notation

## Notation (1/3 note de SEPC)

- Note de l'examen de TP portant sur une portion d'un des trois TPs (même environnement : cmake, tests automatiques, etc.)
- Pas de session 2
- Rendu obligatoire des 3 TPs. Note de 0 au binome si un des TP manque
- Fraude : note de 0 au binome
- Les rendus des 3 TPs servent à remonter les notes d'examen jugées très basses (ex. rendus d'examens qui ne compilent pas).

# Plan

- 1 Système de gestion de mémoire physique
- 2 Algorithmes de gestion de mémoire physique
- 3 'Buddy system'
- 4 GDB

# Système de gestion de la mémoire physique

## Rôle

- Fournir des zones de mémoire aux programmes qui en font la demande
- Gérer l'utilisation des zones de mémoires disponibles

# Système de gestion de la mémoire physique

## Rôle

- Fournir des zones de mémoire aux programmes qui en font la demande
- Gérer l'utilisation des zones de mémoires disponibles

## Deux "types" de mémoire

**Mémoire physique** : Gestion de la mémoire matérielle de la plate-forme (e.g. RAM)

**Mémoire virtuelle** : Fournir une plage mémoire plus large que celle matériellement disponible, et propre à chaque programme utilisateur

# Système de gestion de la mémoire physique

## Rôle

- Fournir des zones de mémoire aux programmes qui en font la demande
- Gérer l'utilisation des zones de mémoires disponibles

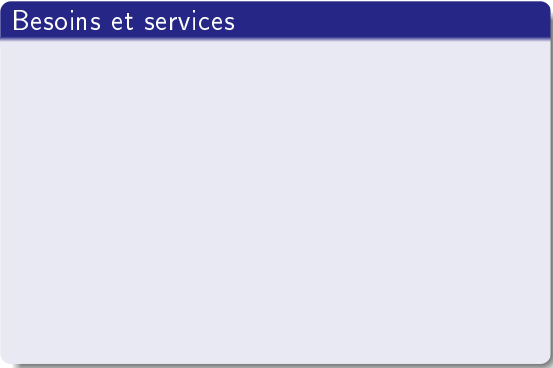
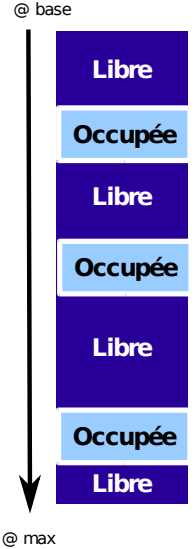
## Deux "types" de mémoire

**Mémoire physique** : Gestion de la mémoire matérielle de la plate-forme (e.g. RAM)

**Mémoire virtuelle** : Fournir une plage mémoire plus large que celle matériellement disponible, et propre à chaque programme utilisateur

# Système de gestion de la mémoire physique

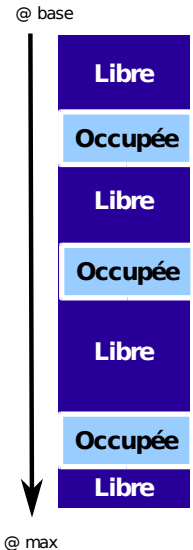
## Besoins et Services





# Système de gestion de la mémoire physique

## Besoins et Services



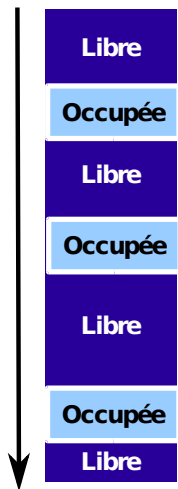
### Besoins et services

- Connaître la mémoire physique :  
@ base, @ max

# Système de gestion de la mémoire physique

## Besoins et Services

@ base



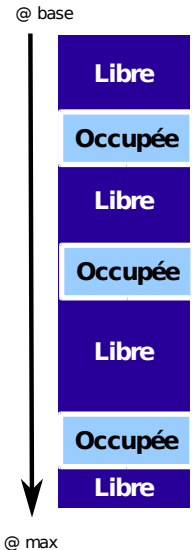
@ max

### Besoins et services

- Connaître la mémoire physique :  
*@ base, @ max*
- Connaître les zones libres et les zones utilisées : *Libre, Occupée*

# Système de gestion de la mémoire physique

## Besoins et Services

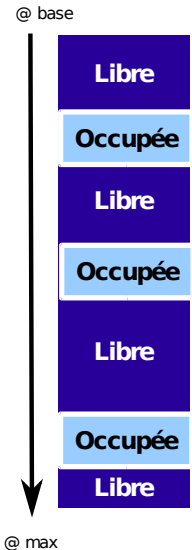


### Besoins et services

- Connaître la mémoire physique :  
*@ base, @ max*
- Connaître les zones libres et les zones utilisées : *Libre, Occupée*
- Fournir des zones libres lorsqu'un programme le demande : *Allocation*

# Système de gestion de la mémoire physique

## Besoins et Services



### Besoins et services

- Connaître la mémoire physique :  
*@ base, @ max*
- Connaître les zones libres et les zones utilisées : *Libre, Occupée*
- Fournir des zones libres lorsqu'un programme le demande : *Allocation*
- Libérer les zones "rendues" par un programme : *Libération*

# Utilisateurs du système de gestion de la mémoire

## Le système d'exploitation

C'est l'utilisateur principal !

- Utilisation pour son propre compte : structures internes, cache de fichiers sur disque, tampon de réception/envoi réseaux, etc.
- Gestion de la mémoire virtuelle : simuler une grande mémoire propre à chaque programme

# Utilisateurs du système de gestion de la mémoire

## Le système d'exploitation

C'est l'utilisateur principal !

- Utilisation pour son propre compte : structures internes, cache de fichiers sur disque, tampon de réception/envoi réseaux, etc.
- Gestion de la mémoire virtuelle : simuler une grande mémoire propre à chaque programme

## les processus / programmes utilisateurs

Uniquement en cas d'absence de mémoire virtuelle (eg. systèmes embarqués)

- Allocation mémoire pour les structures propres à un programme

# Structure de données et primitives d'utilisation

## Structures de données

### Structures de description de la mémoire et de son utilisation

- Description de la zone (@base, taille, ...)
- Descripteurs, tables (liste) des Zones Libres et/ou utilisées

# Structure de données et primitives d'utilisation

## Structures de données

### Structures de description de la mémoire et de son utilisation

- Description de la zone (@base, taille, ...)
- Descripteurs, tables (liste) des Zones Libres et/ou utilisées

## Primitives

Initialisation des structures de gestion de la mémoire

Allocation de mémoire retourne l'adresse du début d'une zone libre contiguë de taille requise

Allouer(taille)  $\Rightarrow$  pointeur de début de zone

Libération de mémoire rend la zone précédemment allouée

Libérer(@zone, taille)  $\Rightarrow$  code d'erreur



# Plan

- 1 Système de gestion de mémoire physique
- 2 Algorithmes de gestion de mémoire physique
- 3 'Buddy system'
- 4 GDB

# Stockage de l'information

## Contradiction ?

- Pour gérer la mémoire, il faut stocker de l'information sur les différentes zones,
- Cette information sur les zones doit être stockée dans la mémoire.

# Stockage de l'information

## Contradiction ?

- Pour gérer la mémoire, il faut stocker de l'information sur les différentes zones,
- Cette information sur les zones doit être stockée dans la mémoire.

## Où et que stocker ?

# Stockage de l'information

## Contradiction ?

- Pour gérer la mémoire, il faut stocker de l'information sur les différentes zones,
- Cette information sur les zones doit être stockée dans la mémoire.

## Où et que stocker ?

- On peut stocker de l'information dans les Zones Libres !

# Stockage de l'information

## Contradiction ?

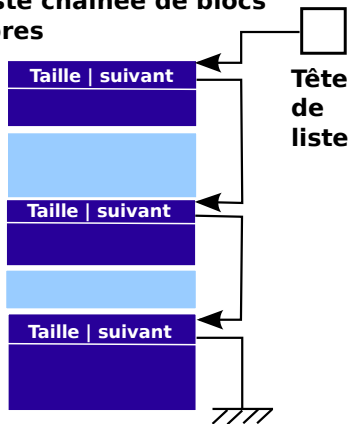
- Pour gérer la mémoire, il faut stocker de l'information sur les différentes zones,
- Cette information sur les zones doit être stockée dans la mémoire.

## Où et que stocker ?

- On peut stocker de l'information dans les Zones Libres !
- Du coup, on stocke les descriptions des Zones Libres !  
Pourquoi ? Avantages ? Inconvénients ?

# Algorithme de base : chaînage des Zones Libres (ZL)

## Liste chaînée de blocs libres



- Informations sur les ZL contenues dans les ZL (taille, adresse suivante)
- Chaînage simple ou circulaire
- Allocation
  - Parcours de la liste des ZL.
  - Choix d'une ZL en fonction de la taille demandée suivant des critères de choix variés : *best fit*, *first fit*, *worst fit*,...
- Libération et fusion éventuelle des zones libres adjacentes

# Avantages - Inconvénients

## Avantages

- Utilisation des zones libres pour stocker les informations du système de gestion de la mémoire
- Algorithme simple

## Inconvénients

- Performances : parcours linéaire
- Fragmentation : Allocation de petite taille en général

# Plan

- 1 Système de gestion de mémoire physique
- 2 Algorithmes de gestion de mémoire physique
- 3 'Buddy system'
- 4 GDB



# Plan

- 1 Système de gestion de mémoire physique
- 2 Algorithmes de gestion de mémoire physique
- 3 'Buddy system'**
- 4 GDB

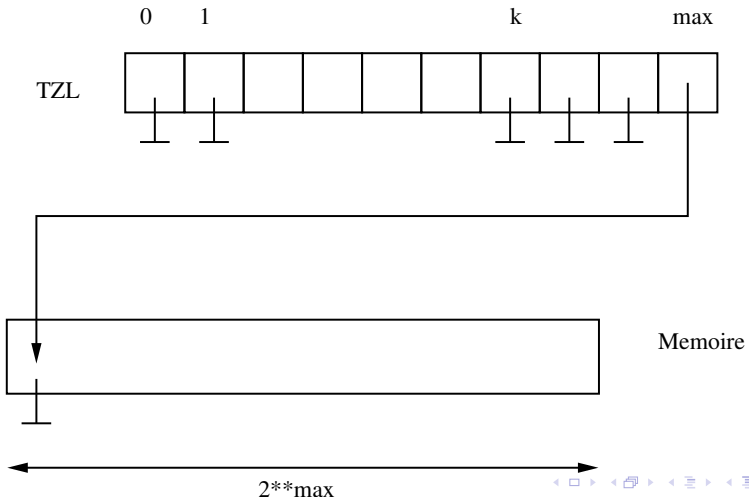
# Algorithme du compagnon (buddy system)

- Allocation par blocs de tailles prédéfinies
  - Blocs de taille  $2^k$
  - pour une totale mémoire de taille  $2^{max}$
- Principe d'allocation (dans la Table des zones libres)
  - Recherche d'un bloc de taille  $\geq 2^k$ .
  - Découpage des blocs libres en 2 blocs de taille inférieure (2 'compagnons') si nécessaire
- Principe de libération
  - Recherche du 'compagnon' du bloc libéré
  - Fusion (récursive) des 'compagnons' si possible pour rendre un seul bloc libre de la taille maximum possible

# Allocation dans le 'buddy system'

État initial

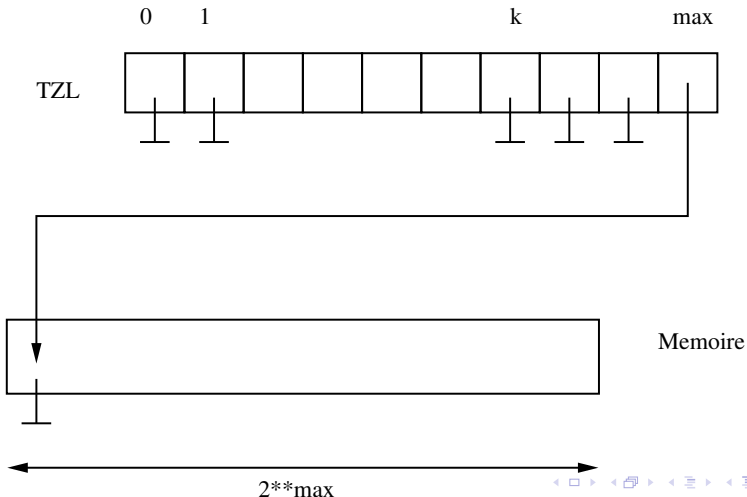
Dans l'état initial, il n'y a qu'un bloc de taille  $2^{max}$



# Allocation dans le 'buddy system'

Allocation de  $2^k$

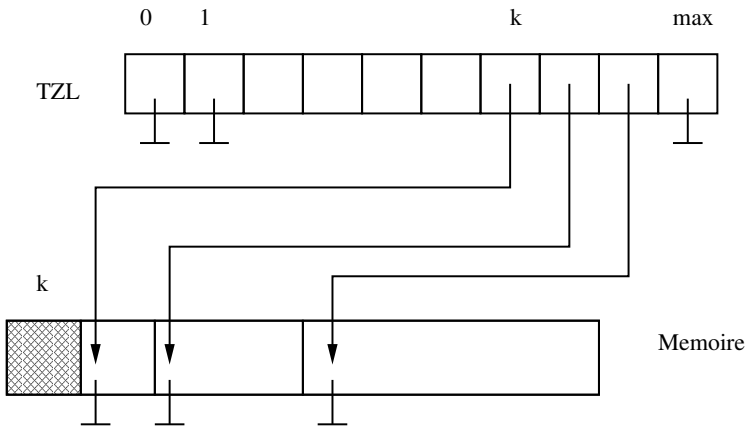
Pour allouer un bloc de taille  $2^k$  il faut trouver un bloc de taille supérieure ou égale et le découper récursivement si nécessaire.



# Allocation dans le 'buddy system'

## Allocation de $2^k$

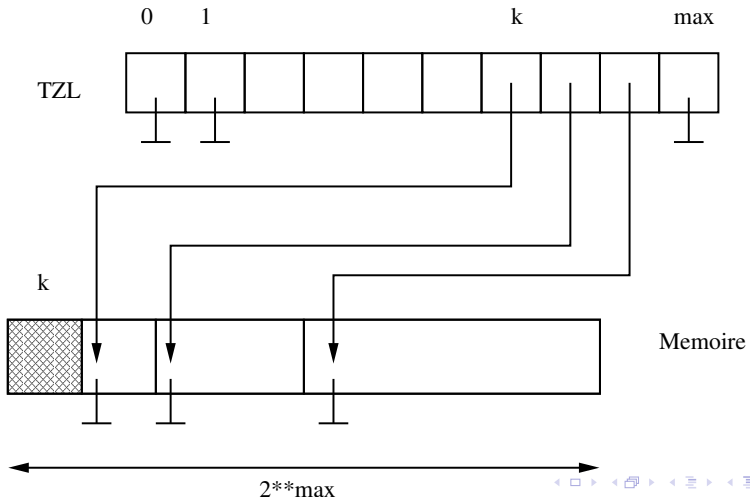
Pour allouer un bloc de taille  $2^k$  il faut trouver un bloc de taille supérieure ou égale et le découper récursivement si nécessaire.



# Libération dans le 'buddy system'

## Fusion

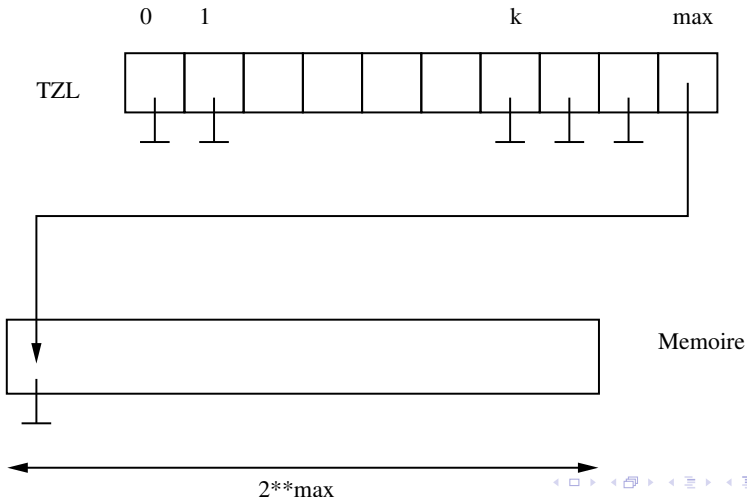
Il faut fusionner un bloc libéré de taille  $2^k$  avec son compagnon si ce dernier est libre.



# Libération dans le 'buddy system'

## Fusion

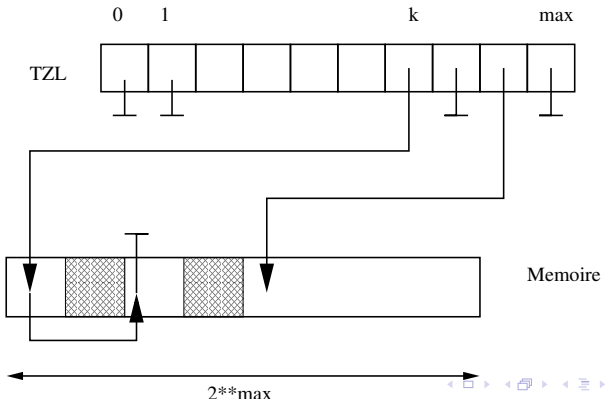
Il faut fusionner un bloc libéré de taille  $2^k$  avec son compagnon si ce dernier est libre.



# Question sur le 'buddy system'

## Question

Proposer une séquence courte d'allocation/désallocation produisant l'état suivant (Lors de la découpe d'un bloc pour une allocation, c'est le bloc de gauche qui est utilisé)





# Recherche du compagnon

## Calcul du compagnon

Les adresses des blocs sont des multiples de leur taille. Les compagnons correspondent à deux paires successives.

### Question

Pour des blocs de taille 2, les blocs d'adresses 4 et 6 sont compagnons. Explicitiez les valeurs de tailles et d'adresses en base 2 et déduisez-en comment calculer l'adresse du compagnon d'un bloc libéré.

# Recherche du compagnon

## Calcul du compagnon

Les adresses des blocs sont des multiples de leur taille. Les compagnons correspondent à deux paires successives.

### Question

Pour des blocs de taille 2, les blocs d'adresses 4 et 6 sont compagnons. Explicitiez les valeurs de tailles et d'adresses en base 2 et déduisez-en comment calculer l'adresse du compagnon d'un bloc libéré.  $ADR \oplus TAILLE == ADR\_COMPAGNON$ .

## Compléments : Allocation en nombre de Fibonacci

Une allocation en puissance de 2 entraîne parfois des "pertes" car le système utilise un grand nombre d'objets de taille  $2^n + k$  avec  $k$  petit.

## Compléments : Allocation en nombre de Fibonacci

Une allocation en puissance de 2 entraîne parfois des "pertes" car le système utilise un grand nombre d'objets de taille  $2^n + k$  avec  $k$  petit.

### Approche de FreeBSD

- Semblable à l'algorithme du Buddy
- La zone initiale est de taille  $\text{fibonacci}(n)$
- Chaque zone  $\text{fibonacci}(k)$  est coupée en zones de taille  $\text{fibonacci}(k - 1)$  et  $\text{fibonacci}(k - 2)$
- À la fusion, on cherche à savoir si le compagnon est le bloc  $\text{fibonacci}(k - 1)$ , "avant", ou  $\text{fibonacci}(k + 1)$  "après" (un seul des deux est valide!). On recalcule le découpage de façon identique à la création.

# Compléments : Allocation en weighted buddy

# Compléments : Allocation en weighted buddy

## Compagnon pondéré

- Semblable à l'algorithme du Buddy, mais avec les tailles intermédiaires en plus. (On perdra au pire 1/4 au lieu de 1/2).
- La zone initiale est de taille  $2^n$
- chaque zone  $2^k$  est coupée en 2 zones de taille  $2^{k-2}$  et  $3 \times 2^{k-2}$  (1/4 ET 3/4)
- Chaque zone  $3 \times 2^k$  ( $2^{k+1} < 3 \times 2^k < 2^{k+2}$ ) est coupée en 2 zones de taille  $2^k$  et  $2^{k+1} = 2 \times 2^k$  (1/3 et 2/3).
- À la fusion, on recalcule la découpe pour connaître la taille et la position du compagnon.
- Deux tableaux supplémentaires : SIZE[i], la taille de la i-ème entrée de la TZL et SUBBUDDY[i], l'indice dans la TZL du petit buddy de la découpe (l'indice du grand est i-1).

# GDB est votre ami !

## GDB en 1A

Vous devriez (re)faire le TP gdb de 1A du projet C (break ; print ; run ; step ; next)

gdb sait faire mille autres choses pour les curieux : API Python ; Threads ; core ; signaux ; débogage à distance ; exécuter le programme à l'envers, etc.

Le slide suivant donne quelques commandes utiles pour le TP 1

# GDB est votre ami !

même dans ddd, eclipse, qtcreator, codelite, code : :blocs, kdevelop, monodevelop, emacs

- autocompletion sur les noms
- `i=100 #` changer une variable à la volée
- `print TZL@5 #` 5 premières cases de la TZL
- `x/5xg TZL #` 5 premiers pointeurs 64bits (g) de la TZL en hexa (x)
- `break mem_alloc if n == 64 #` stop seulement pour les allocations de 64
- `watch -1 & TZL[3]->suiv #` stop si la valeur du champ suivant change
- `checkpoint et restart #` pour ne pas repartir du début
- `set $a=cour->suiv->suiv; print $a #` des variables
- `define ajout print $arg0 + $arg1 #` des fonctions