Cours de Système d'Exploitation

Mamadou M. Kanté

Université Blaise Pascal - LIMOS, CNRS

Bibliographie: Tanenbaum, Cours de SE suivi en 2004 à Univ Bordeaux 1, Cours de F. Pellegrini (LaBRI) et autres.

Organisation

- 14h de CM en L3 et 10h de CM en L2.
- 16h de TP en L3, et 20h de TP en L2.

Organisation

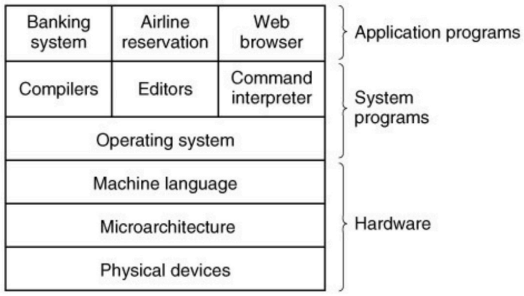
- 14h de CM en L3 et 10h de CM en L2.
- 16h de TP en L3, et 20h de TP en L2.
- Évaluation
  - 1 Examen Terminal,
  - 1 TP noté.
- Note finale :

$$\text{moy}(70\%\text{Partiel} + 30\%\text{TP})$$

Plan

Plan

Un ordinateur



(Tanenbaum)

Le matériel n'est qu'une coquille vide : il faut affecter des fonctionnalités.

OS ?

Un OS (Operating System)

- le logiciel qui cache la complexité de l'architecture matérielle et la rend facilement exploitable :

OS ?

Un OS (Operating System)

- le logiciel qui cache la complexité de l'architecture matérielle et la rend facilement exploitable :

- Donne aux développeurs une base (/bin)
- Rend les choses simples, uniformes et cohérente (/media)

OS ?

Un OS (Operating System)

- le logiciel qui cache la complexité de l'architecture matérielle et la rend facilement exploitable :

- Donne aux développeurs une base (/bin)
- Rend les choses simples, uniformes et cohérente (/media)

OS = machine virtuelle plus facile à programmer et gestionnaire de ressources

OS ?

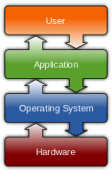
Le système d'exploitation est

- Une machine virtuelle : on n'a pas à se soucier de comment invoquer les commandes des périphériques
  - Les spécificités matérielles sont cachées aux programmeurs.
- Un gestionnaire de ressources : Gestion d'accés aux différents périphériques (clavier, écran, imprimante, interfaces réseaux, etc.)

Le système d'exploitation se charge

- Des fichiers
- Des processus
- De la mémoire
- Des E/S
- Des utilisateurs

OS ? II



Un OS (Operating System) ce n'est pas :

- L'interprète de commandes (logiciel système)
- L'interface graphique
- Les utilitaires (*cp, chmod, ls,...*)
- Le BIOS

OS ? III

Un OS (Operating System) c'est :

Une machine virtuelle

- Vue Uniforme des entrées/sorties
- Une mémoire virtuelle et partageable
- Gestions Fichiers et Répertoires
- Gestion droits d'accès, sécurité et traitement d'erreurs
- Gestions Processus (Ordonnancement, Communication,...)

Un gestionnaire de ressource

- Fonctionnement des Ressources
- Contrôle l'accès aux Ressources
- Gestion des erreurs
- L'évitement des conflits (ressource critique)

Pourquoi suivre cette UE ?

- C'est la BASE, le b.a.-ba, le cambouis primordial...
- C'est votre ami : il gère vos programmes
- Les problèmes rencontrés sont classiques

Types d'OS

Différents types d'OS pour différentes fonctionnalités :

- Mono utilisateur : votre téléphone par exemple
- Temps réel (Nucléaire, Chimie) : réactif
- Général (Linux, Windows, Android, Mac OS,...) Multi-tâches et Multi-utilisateurs

Concepts d'un OS

- Les programmes en exécution sont encapsulés dans des processus.
  - Gestion multi-programmation.
  - Gestion des communications entre programmes.
  - Gestion des attentes de réponse des programmes.
  - Droits associés aux restrictions fixés aux utilisateurs.

Concepts d'un OS

- Les programmes en exécution sont encapsulés dans des processus.
  - Gestion multi-programmation.
  - Gestion des communications entre programmes.
  - Gestion des attentes de réponse des programmes.
  - Droits associés aux restrictions fixés aux utilisateurs.
- (Systèmes de) Fichiers
  - Les données stockées dans des objets appelés fichiers et on s'abstrait des disques.
  - Gestion des droits d'accés aux fichiers.

Concepts d'un OS

- Les programmes en exécution sont encapsulés dans des processus.
  - Gestion multi-programmation.
  - Gestion des communications entre programmes.
  - Gestion des attentes de réponse des programmes.
  - Droits associés aux restrictions fixés aux utilisateurs.
- (Systèmes de) Fichiers
  - Les données stockées dans des objets appelés fichiers et on s'abstrait des disques.
  - Gestion des droits d'accés aux fichiers.
- Mémoire
  - Gestion de la mémoire d'exécution des programmes par espaces d'adressage.
  - Mémoire transformée en mémoire virtuelle.

Concepts d'un OS

- Les programmes en exécution sont encapsulés dans des processus.
  - Gestion multi-programmation.
  - Gestion des communications entre programmes.
  - Gestion des attentes de réponse des programmes.
  - Droits associés aux restrictions fixés aux utilisateurs.
- (Systèmes de) Fichiers
  - Les données stockées dans des objets appelés fichiers et on s'abstrait des disques.
  - Gestion des droits d'accés aux fichiers.
- Mémoire
  - Gestion de la mémoire d'exécution des programmes par espaces d'adressage.
  - Mémoire transformée en mémoire virtuelle.
- Un ensemble de fonctions systèmes avec des super-droits et une sémantique d'appel particulière appelée appels systèmes.
  - L'interpréteur Shell est l'exemple type utilisant beaucoup les appels systèmes.

Plan

Principales Fonctionnalités d'un système

Attention : 1 seule instruction par temps CPU

1. Proposer des services pour accéder au matériel: fonctions systèmes
   - read, open, fork, dup, etc.
2. Traiter les erreurs matérielles des processus
   - division par zero, seg fault, etc.
3. Traiter les interruptions matérielles
   - erreur de lecture disque, ecran, souris, clavier, etc.
4. Entretien global: accés au processeur, allocation de mémoire, etc.

Principales Fonctionnalités d'un système

Attention : 1 seule instruction par temps CPU

1. Proposer des services pour accéder au matériel: fonctions systèmes
   - read, open, fork, dup, etc.
2. Traiter les erreurs matérielles des processus
   - division par zero, seg fault, etc.
3. Traiter les interruptions matérielles
   - erreur de lecture disque, ecran, souris, clavier, etc.
4. Entretien global: accés au processeur, allocation de mémoire, etc.

Par le mécanisme des appels systèmes

Problématiques

1. Protection matérielle.
2. Certaines instructions sont réservées (pour la protection par exemple) ou accés à certaines parties de la mémoire.

Problématiques

1. Protection matérielle.
2. Certaines instructions sont réservées (pour la protection par exemple) ou accés à certaines parties de la mémoire.

Mémoire virtuelle et 2 modes d'exécution: utilisateur et noyau

Problématiques

1. Protection matérielle.
2. Certaines instructions sont réservées (pour la protection par exemple) ou accés à certaines parties de la mémoire.

Mémoire virtuelle et 2 modes d'exécution: utilisateur et noyau

Mémoire virtuelle

1. Les adresses mémoire des programmes ne peuvent référencer les adresses physiques.
2. Les processus ont des espaces d'adressage virtuel
3. Lors du chargement les adresses virtuelles sont traduites en adresses physiques (changement de contexte)
   - Un circuit Memory Management Unit fait la conversion à l'aide de registres
   - Une table de conversion pour chaque processus

Modes d'Exécution

Utilisateur

1. Processus peut accéder uniquement à son espace d'adressage et à un sous-ensemble du jeu d'instructions.
   $\Longrightarrow$ pas de corruption du système
2. L'accés à l'espace noyau est protégé et on y accède par une instruction protégée.

Modes d'Exécution

Utilisateur

1. Processus peut accéder uniquement à son espace d'adressage et à un sous-ensemble du jeu d'instructions.
   $\Longrightarrow$ pas de corruption du système
2. L'accés à l'espace noyau est protégé et on y accède par une instruction protégée.

Noyau

1. Accès à tous les espaces : noyau et utilisateur
   - Code et données du SE accessible seulement en mode noyau: les segments mémoire sont inclus seulement lors du passage en mode mémoire.
2. Accés à toutes les instructions protégées (qui ne peuvent exécutées qu'en mode noyau)
   - Instructions de modification segments de mémoire: un processus ne peut pas modifier ses droits d'accés à la mémoire.
   - Accés aux périphériques: E/S, réseaux, allocation mémoire, etc.

Noyau?

Machine virtuelle

1. Vue uniforme des E/S
2. Gestion de la mémoire et des processus, réseau
3. Système de fichiers

Gestionnaire de ressources

1. Fonctionnement des ressources (processeur, délais, ...)
2. Contrôle d'accés aux ressources (Allocation CPU, disque, mémoire, canal de communication réseau, ...)
3. Gestion des erreurs
4. Gestion des conflits

Modes d'exécution

Mode noyau $\neq$ mode root

1. Mode noyau = gestion par le matériel via des interruptions (matérielle et logicielle)
2. Mode root = gestion logicielle (par le code du SE) et est souvent en mode utilisateur.

Mode noyau par le matériel

1. non connaissance lors de la compilation des segments de mémoire où se trouvent les fonctions systèmes.
2. Raisons: maintenabilité et portabillité du SE

Standard POSIX: Portable Operating System Interface

1. Est une interface de programmation système.
   - Un ensemble de fonctions disponibles sur tous les SE *IX et pratiquement implémentées par tous.
   - Un ensemble de types : time_t, size_t, dev_t, ...
2. Beaucoup de fonctions libc sont des wrappers: font juste appel à la fonction système (ex: time, E/S, etc.)
   - Stocker les arguments dans les bons registres
   - Invoquer l'appel système
   - Interpréter la valeur de retour et si possible positionner la variable `errno`.

Principe Exécution Appels Systèmes (POSIX)

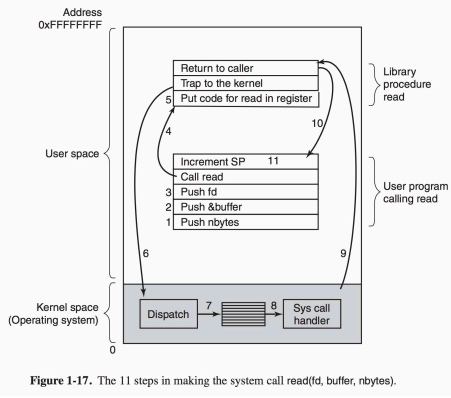


Figure 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

(Tanenbaum)

Principe Exécution Appels Systèmes (POSIX)

1. Lors de l'initialisation on installe les codes des appels systèmes dans une table Interrupt handler et à chaque fonction système on associe un numéro interrupt
2. Dans le code de l'appel système on a une instruction de passage en mode noyau (sous Linux: int) qui prend en arguments le numéro de la fonction système et les différents arguments de la fonction.
3. Depuis le mode noyau
   1. On appelle le gestionnaire d'exception trap handler: sauvegarde du contexte et transfert des données vers espace noyau.
   2. Ce dernier à son tour appelle la vraie fonction système (indexée par son numéro)
   3. Après calcul, transmission valeur de retour au trap
   4. transmission de la valeur de retour et des données et retour en mode utilisateur (encore instruction protégée) après restauration du contexte

Types d'Appels Systèmes

Appel bloquant. Le processus appelant ne pourra continuer son travail que lorsque l'appel système a terminé (lorsque les données demandées sont prêtes par exemple). Ex: appels système: open, read, write

Appel non bloquant. On fixe un délai δ. La main est redonnée automatiquement au processus appelant si au bout de δ temps l'appel système n'a pas terminé. Ex: read, write. Il existe des fonctions pour passer d'un mode bloquant à un mode non bloquant ou inversement.

Attente active. Le processus simule lui-même un mode bloquant sur un appel non bloquant. Ex: while (1) { r= read (...); if ($r \geq 0$) break;}

Gestion des erreurs

1. Une variable globale `errno` dans errno.h qui permet de transmettre les erreurs des fonctions systèmes aux codes utilisateurs
2. Un appel système qui réussit et alors le retour de la wrapper est un entier ≥ 0
3. Un appel système qui échoue et alors le retour de la wrapper est un entier < 0 et un positionnement de la variable `errno` : numéro de l'erreur

   Les fonctions `strerror(int)` et `perror(string)` pour avoir/afficher le texte associé à l'erreur : perror("ouverture") affiche

   "ouverture:"+message associé à `errno`

Gestion des erreurs

1. Une variable globale `errno` dans errno.h qui permet de transmettre les erreurs des fonctions systèmes aux codes utilisateurs
2. Un appel système qui réussit et alors le retour de la wrapper est un entier ≥ 0
3. Un appel système qui échoue et alors le retour de la wrapper est un entier < 0 et un positionnement de la variable `errno` : numéro de l'erreur

   Les fonctions `strerror(int)` et `perror(string)` pour avoir/afficher le texte associé à l'erreur : perror("ouverture") affiche

   "ouverture:"+message associé à `errno`

man 2 intro pour la liste des valeurs possibles de `errno`

Plan

Plan

Pas d'abstraction de la Mémoire

- Mémoire physique correspond à la mémoire du processus.
- Un seul processus à la fois en mémoire.
- Multi-processing lourd = copies dans le disk de l'image d'un processus avant de switcher un nouveau.

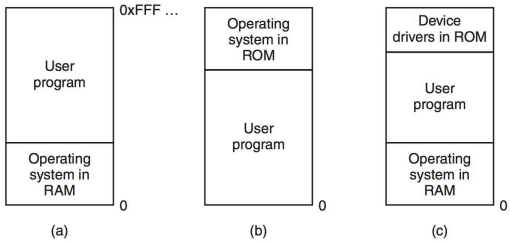


Figure 3-1. Three simple ways of organizing memory with an operating system and one user process. Other possibilities also exist.

Pas d'abstraction de la Mémoire

Chaque processus a son propre espace d'adressage dans la mémoire

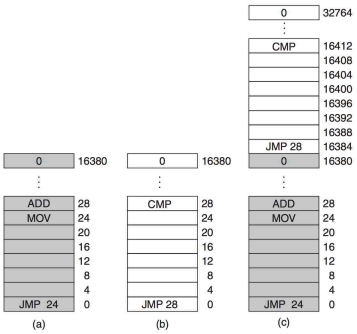
2 problèmes : protection mémoire des processus et accés mémoire

Pas d'abstraction de la Mémoire

Chaque processus a son propre espace d'adressage dans la mémoire

2 problèmes : protection mémoire des processus et accés mémoire

Solution 0 (IBM). Le processus a une adresse de base. Mais pose problème.



Pas d'abstraction de la Mémoire

Chaque processus a son propre espace d'adressage dans la mémoire

2 problèmes : protection mémoire des processus et accés mémoire

Solution 1. Le processus a une adresse de base et peut-être une taille limite.

- Protection (IBM): on peut donner des codes aux processus et coder dans les mots ces codes des processus.
- Accés mémoire : on recalcule les adresses à partir des addresses de base.
  - Logiciel : à la compilation on détermine les accés adresses et on ajoute l'adresse de base. Qui est adresse?

Pas d'abstraction de la Mémoire

Chaque processus a son propre espace d'adressage dans la mémoire

2 problèmes : protection mémoire des processus et accés mémoire

Solution 1. Le processus a une adresse de base et peut-être une taille limite.

- Protection (IBM): on peut donner des codes aux processus et coder dans les mots ces codes des processus.
- Accés mémoire : on recalcule les adresses à partir des addresses de base.
  - Logiciel : à la compilation on détermine les accés adresses et on ajoute l'adresse de base. Qui est adresse?
  - Matériel : 1 registre qui transforme les accés adresse en ajoutant l'adresse de base.

Pas d'abstraction de la Mémoire

Chaque processus a son propre espace d'adressage dans la mémoire

2 problèmes : protection mémoire des processus et accés mémoire

Solution 1. Le processus a une adresse de base et peut-être une taille limite.

- Protection (IBM): on peut donner des codes aux processus et coder dans les mots ces codes des processus.
- Accés mémoire : on recalcule les adresses à partir des addresses de base.
  - Logiciel : à la compilation on détermine les accés adresses et on ajoute l'adresse de base. Qui est adresse?
  - Matériel : 1 registre qui transforme les accés adresse en ajoutant l'adresse de base.
    - Comment protéger les autres processus ?

Pas d'abstraction de la Mémoire

Chaque processus a son propre espace d'adressage dans la mémoire

2 problèmes : protection mémoire des processus et accés mémoire

Solution 1. Le processus a une adresse de base et peut-être une taille limite.

- Protection (IBM): on peut donner des codes aux processus et coder dans les mots ces codes des processus.
- Accés mémoire : on recalcule les adresses à partir des addresses de base.
  - Logiciel : à la compilation on détermine les accés adresses et on ajoute l'adresse de base. Qui est adresse?
  - Matériel : 1 registre qui transforme les accés adresse en ajoutant l'adresse de base.
    - Comment protéger les autres processus ? Adresse limite

Pas d'abstraction de la Mémoire

Chaque processus a son propre espace d'adressage dans la mémoire

2 problèmes : protection mémoire des processus et accés mémoire

Solution 1. Le processus a une adresse de base et peut-être une taille limite.

- Protection (IBM): on peut donner des codes aux processus et coder dans les mots ces codes des processus.
- Accés mémoire : on recalcule les adresses à partir des addresses de base.
  - Logiciel : à la compilation on détermine les accés adresses et on ajoute l'adresse de base. Qui est adresse?
  - Matériel : 1 registre qui transforme les accés adresse en ajoutant l'adresse de base.
    - Comment protéger les autres processus ? Adresse limite
    - Lenteur : addition et comparaison avant chaque instruction d'adressage.
    - Extension des processus ? Comment fixer la taille des processus ?
    - Certains processus nécessitent plus que la mémoire physique.

Va et Vient

Discuter au tableau de la gestion de la mémoire

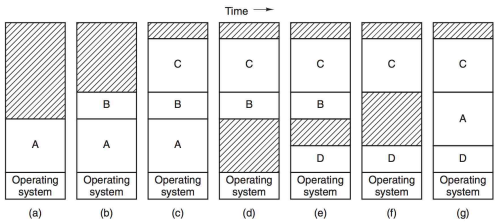


Figure 3-4. Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory.

Va et Vient

Discuter au tableau de la gestion de la mémoire

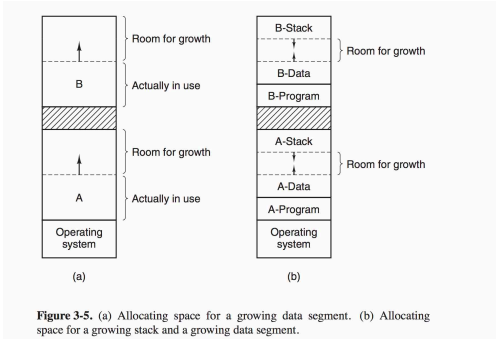


Figure 3-5. (a) Allocating space for a growing data segment. (b) Allocating space for a growing stack and a growing data segment.

Gestion Mémoire Libre

Diviser la mémoire en blocs

Questions. Taille des blocs ? Gestion des blocs utilisés ? libres ?

- Par bitmap : tradeoff taille bitmap/taille bloc ? Discuter recherche mémoire.

Gestion Mémoire Libre

Diviser la mémoire en blocs

Questions. Taille des blocs ? Gestion des blocs utilisés ? libres ?

- Par bitmap : tradeoff taille bitmap/taille bloc ? Discuter recherche mémoire.
- Par listes chaînées : discuter de différentes possibilités et de comment optimiser le choix pour un processus.

Mémoire Virtuelle par Pagination

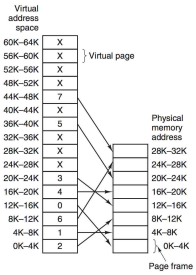
Taille des processus trop grands pour la mémoire

- Diviser le processus en pages = un bloc contigu d'adresses du processus.
- Ces pages virtuelles sont mises en correspondance avec des parties contigües de la mémoire physique.

Mémoire Virtuelle par Pagination

Taille des processus trop grands pour la mémoire

- Diviser le processus en pages = un bloc contigu d'adresses du processus.
- Ces pages virtuelles sont mises en correspondance avec des parties contigües de la mémoire physique.
- Expliquer au tableau : table des pages, mapping par TLB, changement de page.



MMU

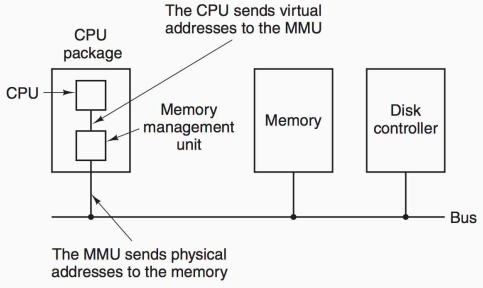


Figure 3-8. The position and function of the MMU. Here the MMU is shown as being a part of the CPU chip because it commonly is nowadays. However, logically it could be a separate chip and was years ago.

Plan

Ramasse-Miettes

- On alloue de la mémoire statiquement (tableaux) ou dynamiquement (les pointeurs ou tableaux dynamiques).
- A certains moments certains ne sont pas utilisés et il faut libérer l'espace (sinon on risque de manquer d'espace).
- Une libération par le programmeur peut ne pas être complète et dans ce cas on a des fuites de mémoire.

Ramasse-Miettes

- On alloue de la mémoire statiquement (tableaux) ou dynamiquement (les pointeurs ou tableaux dynamiques).
- A certains moments certains ne sont pas utilisés et il faut libérer l'espace (sinon on risque de manquer d'espace).
- Une libération par le programmeur peut ne pas être complète et dans ce cas on a des fuites de mémoire.
- Solution. Gérer au niveau système (OS ou machine virtuelle utilisateur) la libération de la mémoire

Ramasse-Miettes

- On alloue de la mémoire statiquement (tableaux) ou dynamiquement (les pointeurs ou tableaux dynamiques).
- A certains moments certains ne sont pas utilisés et il faut libérer l'espace (sinon on risque de manquer d'espace).
- Une libération par le programmeur peut ne pas être complète et dans ce cas on a des fuites de mémoire.
- Solution. Gérer au niveau système (OS ou machine virtuelle utilisateur) la libération de la mémoire Ramasse-Miettes.

Ramasse-Miettes

- On alloue de la mémoire statiquement (tableaux) ou dynamiquement (les pointeurs ou tableaux dynamiques).
- A certains moments certains ne sont pas utilisés et il faut libérer l'espace (sinon on risque de manquer d'espace).
- Une libération par le programmeur peut ne pas être complète et dans ce cas on a des fuites de mémoire.
- Solution. Gérer au niveau système (OS ou machine virtuelle utilisateur) la libération de la mémoire Ramasse-Miettes.

Principe.

- Déterminer les objets qui ne sont plus (ou peuvent plus être) utilisés par le programme.

Ramasse-Miettes

- On alloue de la mémoire statiquement (tableaux) ou dynamiquement (les pointeurs ou tableaux dynamiques).
- A certains moments certains ne sont pas utilisés et il faut libérer l'espace (sinon on risque de manquer d'espace).
- Une libération par le programmeur peut ne pas être complète et dans ce cas on a des fuites de mémoire.
- Solution. Gérer au niveau système (OS ou machine virtuelle utilisateur) la libération de la mémoire Ramasse-Miettes.

Principe.

- Déterminer les objets qui ne sont plus (ou peuvent plus être) utilisés par le programme.
  Non faisable à la compilation, par contre on peut identifier ceux non référencés pendant l'exécution.

Ramasse-Miettes

- On alloue de la mémoire statiquement (tableaux) ou dynamiquement (les pointeurs ou tableaux dynamiques).
- A certains moments certains ne sont pas utilisés et il faut libérer l'espace (sinon on risque de manquer d'espace).
- Une libération par le programmeur peut ne pas être complète et dans ce cas on a des fuites de mémoire.
- Solution. Gérer au niveau système (OS ou machine virtuelle utilisateur) la libération de la mémoire Ramasse-Miettes.

Principe.

- Déterminer les objets qui ne sont plus (ou peuvent plus être) utilisés par le programme.
  Non faisable à la compilation, par contre on peut identifier ceux non référencés pendant l'exécution.
- Libérer la mémoire utilisée par ces objets.

Algorithmes de Ramasse-Miettes

Plusieurs familles

Algorithmes de Ramasse-Miettes

Plusieurs familles

- Comptage de références : A tout bloc alloué est associé un compteur qui compte le nombre de références de ce dernier.

Algorithmes de Ramasse-Miettes

Plusieurs familles

- Comptage de références : A tout bloc alloué est associé un compteur qui compte le nombre de références de ce dernier.
- Algorithmes traversants : Les blocs de la mémoire forment un graphe orienté où chaque objet a 2 arêtes sortantes au plus :

Algorithmes de Ramasse-Miettes

Plusieurs familles

- Comptage de références : A tout bloc alloué est associé un compteur qui compte le nombre de références de ce dernier.
- Algorithmes traversants : Les blocs de la mémoire forment un graphe orienté où chaque objet a 2 arêtes sortantes au plus :
  - Un bloc référence au plus un bloc non utilisé et au plus un bloc utilisé.

32

Algorithmes de Ramasse-Miettes

Plusieurs familles

- Comptage de références : A tout bloc alloué est associé un compteur qui compte le nombre de références de ce dernier.
- Algorithmes traversants : Les blocs de la mémoire forment un graphe orienté où chaque objet a 2 arêtes sortantes au plus :
  - Un bloc référence au plus un bloc non utilisé et au plus un bloc utilisé.
  - Ce graphe évolue : on déplacement de pointeur.

32

Algorithmes de Ramasse-Miettes

Plusieurs familles

- Comptage de références : A tout bloc alloué est associé un compteur qui compte le nombre de références de ce dernier.
- Algorithmes traversants : Les blocs de la mémoire forment un graphe orienté où chaque objet a 2 arêtes sortantes au plus :
  - Un bloc référence au plus un bloc non utilisé et au plus un bloc utilisé.
  - Ce graphe évolue : on déplacement de pointeur.
  - L'algorithme consiste à faire un parcours à partir d'objets racines.

32

Algorithmes de Ramasse-Miettes

Plusieurs familles

- Comptage de références : A tout bloc alloué est associé un compteur qui compte le nombre de références de ce dernier.
- Algorithmes traversants : Les blocs de la mémoire forment un graphe orienté où chaque objet a 2 arêtes sortantes au plus :
  - Un bloc référence au plus un bloc non utilisé et au plus un bloc utilisé.
  - Ce graphe évolue : on déplacement de pointeur.
  - L'algorithme consiste à faire un parcours à partir d'objets racines.
- Générationnels : On hiérarchise les données suivant leur durée de vie et le libérateur commence souvent par les plus jeunes.

Algorithmes de Ramasse-Miettes

Plusieurs familles

- Comptage de références : A tout bloc alloué est associé un compteur qui compte le nombre de références de ce dernier.
- Algorithmes traversants : Les blocs de la mémoire forment un graphe orienté où chaque objet a 2 arêtes sortantes au plus :
  - Un bloc référence au plus un bloc non utilisé et au plus un bloc utilisé.
  - Ce graphe évolue : on déplacement de pointeur.
  - L'algorithme consiste à faire un parcours à partir d'objets racines.
- Générationnels : On hiérarchise les données suivant leur durée de vie et le libérateur commence souvent par les plus jeunes. Exemple : celui de .Net

Exemple d'Algorithme Traversant (Mark and Sweep)

Algorithme de Dijkstra et al., CACM 21(11), 1978

```
marking phase:
begin {there are no black nodes}
   "shade all roots" {P1 and there are no white roots};
   i:= 0; k:= M;
marking cycle:
   do k > 0 → {P1 and there are no white roots}
      ⟨c:= color of node nr. i⟩;
      if c = gray → k:= M;
         C1: ⟨shade the successors of node nr. i and make node nr. i
             black⟩
      [] c ≠ gray → k:= k − 1
      fi;
      i:= (i + 1) mod M
   od
end {P1 and there are no white roots and no gray nodes, hence—as is
   easily seen—all white nodes are garbage};
appending phase:
begin i:= 0;
   appending cycle:
      do i < M → {all nodes with a number <i are nonblack; all nodes
            with a number ≥i are nongray, and are garbage, if white}
         ⟨c:= color of node nr. i⟩;
         if c = white → ⟨append node nr. i to the free list⟩
         [] c = black → ⟨make node nr. i white⟩
         fi;
         i:= i + 1
      od {there are no black nodes}
end
```

Mutator : modifications des références (typiquement lors de déréférencement et référencement)

Collector : Marquage (Mark) et Libération (Sweep).