
Institut
Mines-Télécom

# INF720 – mémoire

## Première partie

Responsable : Thomas Robert (C234-4)

Auteurs : Thomas Robert

# Contenu du cours

- **Présentation des besoins et services associés : chargement & allocation / partage / protection**
- **Présentation de la répartition des responsabilité : compilation / système d'exploitation / processeur pour l'organisation des données en mémoire.**
- **Introduction à la logique de pagination pour la gestion de la mémoire**
  - **Principe**
  - **Intérêt pour le chargement / la protection / le partage des données d'un programme**
  - **Algorithme de remplacement LRU vs FIFO**
- **Description des fonctions implémentant : mapping fichier/mémoire, protection, partage et allocation**

# Les 3 grands besoins

Charger & placer
Partager
Protéger

# Besoins liés au chargement et allocation

- Chaque processus correspond à un programme ont des besoins
  - Différents en quantité de mémoire et qui peuvent varier
  - Différents en nature (code vs donnée)
- Les données d'un programme doivent être placé en mémoire
  - Emplacement connu ou calculable à l'exécution
  - Placement compatible avec les stratégies de protection / partage
  - Chacun veut pouvoir avoir sont propre espace ….
- Le contenu de la mémoire doit être "peuplé"
  - Chargement de donnée depuis un fichier "par morceaux"
  - Optimiser l'efficacité des entrelacement "lectures/écritures"

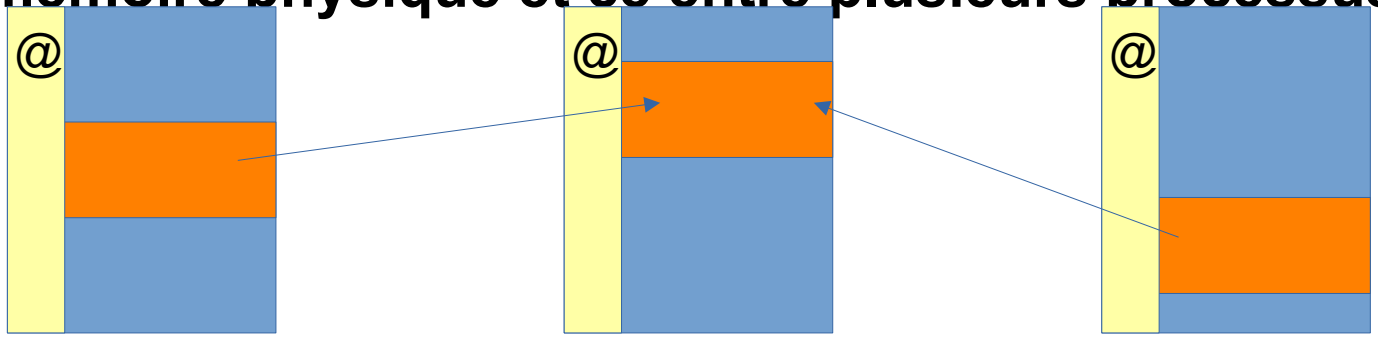
# Besoins liés au chargement et allocation

- **Services :**
  - “mapping” fichier mémoire => mmap / munmap
  - placement d’intervalles disjoints dans les zones de données => malloc
  - Optimiser  l’usage de la mémoire physique (transparent à l’utilisateur)
  - Structuration de la mémoire en “zone” par fonction (donnée, code …)

# Partager ? Pour quoi faire ?

- **Motivations :**
  - **Je lance 1000 fois “hello world !”, je charge 1000 fois le code de printf ? (pas efficace, pas utile …..)**
  - **Je veux un intermédiaire entre processus léger et lourd :**
    - **Léger : tout sauf la pile => beaucoup de data races**
    - **Lourd : rien … comment implémenter diviser pour régner ?**

- **Principe : être capable de lier un intervalle d’adresse à l’exécution du programme avec un ensemble de données en mémoire physique et ce entre plusieurs processus**



# Protéger

- Principe : contrôle d'accès pour actions RWX pour des intervalles d'adresses + mode d'exécution (kernel / user)
- Identification du sujet = le processus / l'utilisateur l'ayant lancé

- Problème : très couteux si intervalle arbitraire
  - Problème de chevauchement (difficile à détecter)
  - Pas 'unicité de la définition d'une politique de protection ([1,2] : rw [2,3]: rw ) ~ ([1,3] rw)

# Répartition des rôles entre

# Compilateur
# Processur
# Système d’exploitation

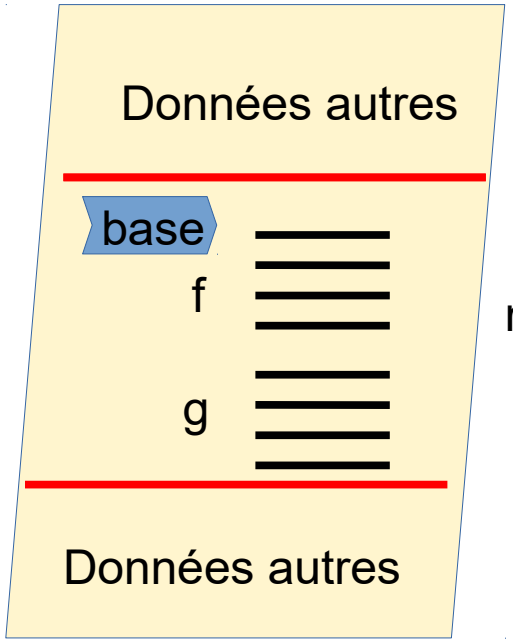
# Partage des responsabilités compilateur / système d'exploitation / processeur (I)

## Retrouver les données

- Que fait le compilateur lorsque que vous appelez une fonction ? (PLACEMENT)
  - Pb comment définit-on les adresses des fonctions
    - Interne au fichier compilé ?
    - Externe au fichier compilé ?


Binaire

- Solution naive : adresse fixe (connue à l'avance)
- Pb : comment lancer 2 fois "grep"
- 
- Solution (pré)historique
  - Utiliser une résolution à l'exécution via : 1 @ de base + 1 décalage fixe intégré au code
  - Notion de "relogeabilité"
- Si fonction interne la base est connue lors d'un appel (c"est celle de la fonction en cours d'exécution ….)




mémoire

# Partage des responsabilités compilateur / système d'exploitation / processeur (II)

## Protéger les données

- Distinguer contenu mémoire d'un programme dans l'objectif de définir ses règles d'accès.

- Organisation dès le binaire
  - Création de "sections" séquences d'octets avec un type de donnée avec un mode d'accès
    - Text : code => RX
    - Data : initialisation des variables globales => RW
    - Rodata : textes constants utilisés dans le programme => R

- A l'édition de lien (dernière étape avant production du binaire)
  - Concaténation des sections et calcul des adresses relatives aux début de section
  - Conservation de lien symbolique pour les bibliothèques chargées à l'exécution (on verra cette idée plus loin)

- Dans un exécutable (et donc au chargement) regroupement des sections dans des segments = unité de protection et allocation (3 usuellement => code, donnée et dynamique)

# Partage des responsabilités compilateur / système d'exploitation / processeur (II)

**Eviter de gaspiller la mémoire physique ...**

- **Autre motivation à morceler le placement en mémoire des données d'un programme : la vision ressource**
- **La table suivante définit les paramètres d'un scénario d'exécution de 3 processus**

| Proc | début | durée | code | data | dyn | total |
|---|---|---|---|---|---|---|
| P1 | 0 | 6 | 3ko | 1ko | 3ko | 7ko |
| P2 | 5 | 9 | 3ko | 4ko | 3ko | 10ko |
| P3 | 10 | 2 | 3ko | 2ko | 3ko | 8ko |

- **Si chargement + allocation contigue à partir d l'adresse 0, on placerait P3 après P2 en on ne pourrait pas exploiter les 7 premiers ko…**
- **Si allocation par segment / section : on placerait code et dyn à partir de l'adresse 0 => au final 19 ko au lieu de 25 nécessaire dans le cas précédant.**