1 Introduction à l'organisation mémoire des architectures de calcul

Présentation générale Un ordinateur est essentiellement composé de 2 type distincts de matériel : d'une part **les unités de calcul** et d'autre part **les unités de stockage**.

Unités de calcul Fournir des résultats aux calculs élémentaires effectués : opérations arithmétiques et logiques ; fonctions élémentaires en mathématiques, ...

Unités de stockage Permettre aux applications de mémoriser des données : données en entrées ou produites en cours et en fin de calcul.

1.1 Hiérarchie mémoire

Histoire Ordinateur composé d'unités arithmétique et logique pouvant lire et écrire des données d'une **mémoire principale**.

Architecture moderne Une **hiérarchie de mémoire**

- des mémoires rapides voir très rapides mais coûteuses (en terme d'intégration). Proches des unités arithmétiques et logiques : **registre** du processeur ou **mémoires caches**.
- une mémoire principale, appelée DRAM, de grande capacité,
- des mémoires secondaires : mémoire non volatile, mémoire flash, disques, bandes magnétiques, ...

Type de mémoire	Temps d'accès indicatif
Registre	
Cache L1 Cache L2 Cache L3	
DRAM	
SSD	
HDD	

Utilisation et gestion

registres gérés par le compilateur en fonction du programme à évaluer,

mémoires caches sauvegardes des dernières données accéedées par un programme,

mémoire principale stocke les variables du programme,

mémoires secondaires stocker les entrées et sorties, ou les données qui doivent être conservées.

1.2 Gestion de la mémoire par le système d'exploitation

Utilisation par un programme Le programme peut accéder aux données grâce à leurs **adresses** indiquant leurs positions dans la mémoire de l'ordinateur.
Le système d'exploitation **arbitre** l'accès à la ressource ≪ mémoire ≫.

Espace virtuel Le système donne à chaque programme qui s'exécute, appelé **processus**, un **espace virtuel de mémoire** ou **mémoire virtuelle**. Cet espace virtuel de mémoire est propre à chaque processus et commence classiquement à l'adresse `0x000..000` (en hexadécimal) et s'arrête à `0xFFF...FFF` (que des bits à 1).

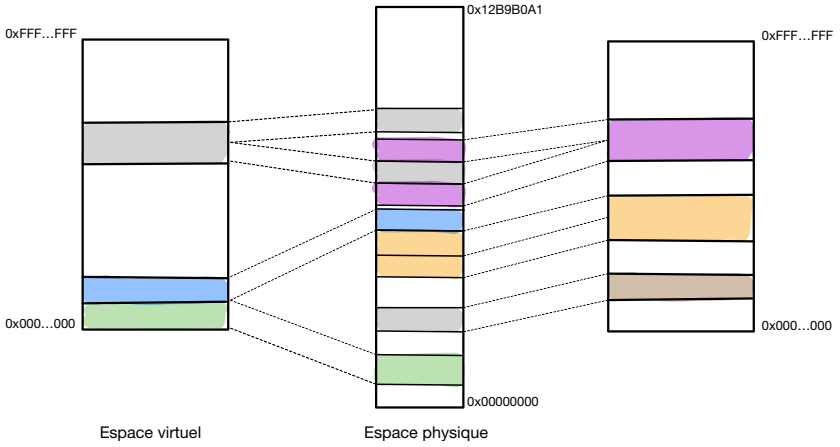
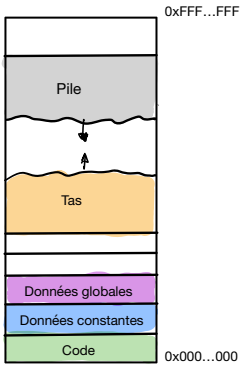


Illustration du mécanisme de projection de l'espace virtuel de mémoire de 2 processus vers une mémoire. Les zones colorées représentent les zones allouées dans l'espace virtuel et associées à des zones de la mémoire physique.

1.3 Organisation mémoire des processus

Illustration de l'organisation des différents types de mémoire d'un processus une fois créé par le système d'exploitation.



Ces quatres zones mémoires permettent de stocker différents type de données :

zone de code code exécutable du programme obtenu par compilation,

zone des données constantes données uniquement lues et jamais écrites,

zone des données globales deux sous parties : zones des données globales non initialisées, et celles initialisées,

tas données qui seront allouées dynamiquement,

pile manière très dynamique, données déclarées dans les fonctions et nécessaires à l'évaluation des expressions du programme.

2 Variables

2.1 Déclaration et définition des variables

Déclaration Donner le nom ou l'identifiant d'une variable
Définition Réserver ou allouer la zone mémoire correspondante.
Exemple `int a;` est la déclaration et la définiton d'une variable de nom `a`.

2.2 Portée d'une variable

Règle La déclaration d'une variable doit précéder son utilisation.
Portée Le nom et la zone mémoire d'une variable ne sont valides qu'après l'instruction de déclaration, et jusqu'à la fin du **bloc d'instructions** qui englobe la déclaration.

2.3 Durée de vie d'une variable

La durée de vie d'une variable est associée à la durée de sa portée dans l'exécution du programme. On parle généralement de variable :

globale variable définie en dehors de toute fonction, connue du point de sa déclaration jusqu'à la fin du fichier compilé. Allouée dans la zone des données globales du processus,

automatique variable définie dans les fonctions ou les blocs d'instructions et n'est visible que dans son propre bloc d'instructions où elle est déclarée. Allouée dans la pile d'exécution.

manuel variable allouée dynamiquement dans le tas par le programme, et libérée par le programme.

semi-manuel variable allouée dynamiquement dans la pile par le programme (`alloca`) et libérée automatiquement par le programme lorsque le **bloc d'activation** est libéré.

3 Pile d'exécution

Vocabulaire Les variables locales (et certaines valeurs temporaires) sont stockées dans la **pile d'exécution** du processus. Le compilateur génère des instructions pour empiler (pousser dans la pile) des valeurs et des instructions pour dépiler (récupérer de la pile). La valeur dépilée sera toujours la dernière valeur empilée ! Cette zone mémoire dans la pile d'exécution fait partie que l'on appelle le bloc d'activation d'un appel de fonction.

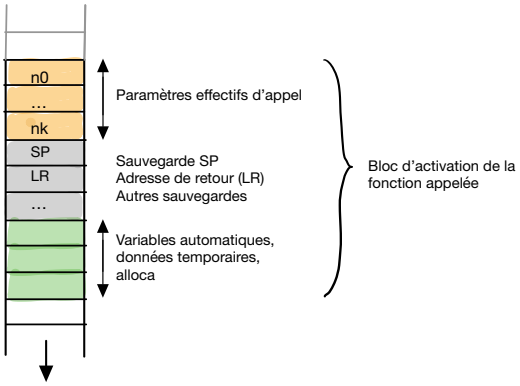
3.1 Réalisation d'un appel de fonction

Considérons une fonction `f` qui appelle une fonction `g`.

1. l'appelant (`f`) pousse sur la pile d'exécution $n_0, n_1, .., n_k$, les paramètres effectifs (les paramètres d'appel) pour `g`,
2. l'appelant sauvegarde des informations (typiquement le sommet de pile),
3. `f` appelle la fonction `g`,
4. l'appelé (`g`) peut sauvegarder d'autres informations (instruction à exécuter en retour de la fonction `g` générée lors de l'étape 3 précédente),
5. `g` alloue sur la pile un espace mémoire pour stocker ses variables locales et les données temporaires (connu à la compilation).

Pour retourner à l'appelant, `g` va restaurer les informations sauvegardées à l'étape 4 avant d'exécuter l'instruction suivante à l'appel à g dans le code de l'appelant `f`.

État de la pile pour un appel à la fonction `g`.



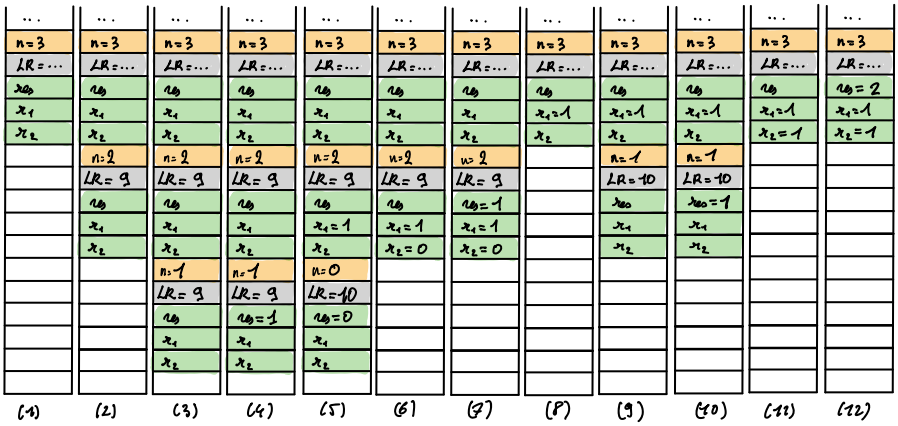
Bloc d'activation ou frame zone mémoire sur la pile d'exécution nécessaire à l'exécution de la fonction :
- déclaration et définition des variables automatiques,
- allocation des variables temporaires,
- allocation dynamique sur la pile (fonction `alloca`).

3.2 Illustration

Exemple projeté d'un code comportant deux fonctions f, illustration des blocs d'activation des fonctions.

3.3 Gestion des appels récursifs

```
1  int fibo(int n)
2  {
3    int res;
4    if (n<2) res = n;
5    else
6    {
7      int r1, r2;
8      r1 = fibo(n-1);
9      r2 = fibo(n-2);
10     res = r1+r2;
11   }
12   return res;
13  }
```



3.4 Comment est gérée la pile d'exécution ?

Gestion de la pile Le processus reçoit à sa création l'information d'une zone mémoire (un tableau) qui servira à stocker les données de la pile d'exécution. Le lanceur de processus initialise la pile lors de l'appel à la première fonction du processus.
La gestion de la pile revient à manipuler un pointeur, le pointeur **SP** pour **Stack Pointer**, qui pointe sur la prochaine adresse qui sera retournée lors de l'empilement d'une donnée.