Systèmes Informatiques 1

Chapitre 1 : Introduction

Section 1.1 : Introduction

Composition d'un système informatique

Le système informatique le plus simple est **un processeur** (*cpu*) et **une mémoire**. Ce prosesseur est capable de : *lire et écrire* des l'informations en mémoire et de *réaliser des calculs*.

Systèmes Unix

Unix est un nom générique donné à une famille de systèmes d'exploitations. On y retrouve MacOS, FreeBSD et Linux qui sont les plus utilisés.

Un système unix est composé de trois grands types de logiciels:
1. **Le noyau du système d'exploitation** : chargé au démarage de la machine, il se charge de toutes les interactions entre les logiciels et le matériel.
2. **Les librairies** :Nombreuses, elles facilitent l'écriture et le développement d'applications
3. **Les programmes utilitaires** : utilisés pour résoudre une série de problèmes
API signifie *Application Programming Interface*

/usr : utilitaires et librairies installées sur le système
/bin et **/sbin** : utilitaires de base nécéssaire à l'admin. syst.
/tmp : fichiers temporaires (effacé au redémarrage)
/etc : configuration du système
/home : répertoire personnel des utilisateurs
/dev : fichiers spéciaux
/root : administrateur système

En Unix une application est composée de un ou plusieurs processus.
Un processus : ensemble cohérent d'instructions qui utilisent une partie de la mémoire et sont exécutées sur un processeur.

Les processus peuvent utiliser des ressources en mémoire. lorsque le processus va se terminer, il va libérer ces ressources et retourner un entier au processus parent (0 si Ok sinon autre)

Shell

généralement appelée console ou terminal. Un **shell** est un programme qui à été spécialement conçu pour faciliter l'utilisation d'un système Unix via le clavier. Sa puissance vient de sa capacité à écrire des commandes enchaînées : <,>,>>,|
Une **pipe** ;) est une redirection de la sortie standartd d'un programme vers l'entrée std d'un autre sans passer apr un fichier intermédiaire.
mv : utilitaire pour renommer ou déplacer un fichier ou dossier.
head et **tail** : extrait le debut et la fin d'un fichier.
wc : compte le nombre de (lignes, mots, caractères).
sort : trie le fichier par ordre alphabétique
uniq : retire les doublons *(Attention : fichier trié au préalable)*
gzip/**gunzip** : compression / décompression d'un fichier .gz
cp : copie un fichier ou dossier (-r pour les dossiers).
rm : efface un fichier ou dossier.
mkdir : crée un répertoire.
rmdir : efface un répertoire vide.
cd : change le répertoire courant.
pwd : affiche le répertoire courant.
./prog : est utilisé pour lancer le programme prog.
grep : utilitaire permettant d'extraire d'un fichier les lignes qui contiennent ou non une chaine de caractère passée en argument.
echo "Blabla" >> file.txt copie le texte Blabla dans le fichier file.txt.

man :lire les pages de manuel d'un système Unix.

Un script bash commence par

```
#!/bin/bash
$# #nombre d'args
$1 #arg1 ..
$@ # liste des args
if [Cond]; then ...fi
exit 0
```

Chapitre 2 : Langage C

Section 2.1 : Le langage C

Le langage C cest un langage rapide qui compose la plus part des systèmes d'exploitations actuels.

Le langage machine : langage binaire pour le processeur.
Le langage assembleur est converti en langage machine grâce à un assebleur. Ce langage est le plus proche du processeur.
Chaque famille de processeur possède un language d'assemblage qui lui est propre

Préprocesseur

Au moment de la compilation, le compilateur va exécuter les directives préprocesseur.

```
#define <...> //ajoute les librairies au moment de la compilation
#define ZERO 0 // replace tout les ZERO par 0 au moment de la compilation
```

Constructons Syntaxiques

```
if (COND){}else{}
while(COND){} //if cond false do nothing
do {} while(COND); if cond is false then
for(INIT;COND;INCR){}
```

```
#include<stdio.h>

int main ( intarg c , char * argv [ ] ){
    printf ( "Hello , %s!\n" , NAME) ;// affiche sur la sortie standard le
    // \n c'est pour le retour à la ligne le %s c'est pour la variable
    return 0;//Un programme retourne toujours une valeur (en C : return ou exit ).
}
```

Manuel

accessible via la commande man
1. Utilitaire disponible pour tous les utilisateurs
2. Appels systèmes en C
3. Fonctions de la librairie
4. Fichiers spéciaux
5. Formats de fichiers et conventions pour certains types de fichiers
6. Jeux
7. Utilitaires de manipulation de fichiers textes
8. Commandes et procédure de gestion du système

Section 2.2 : Types de données

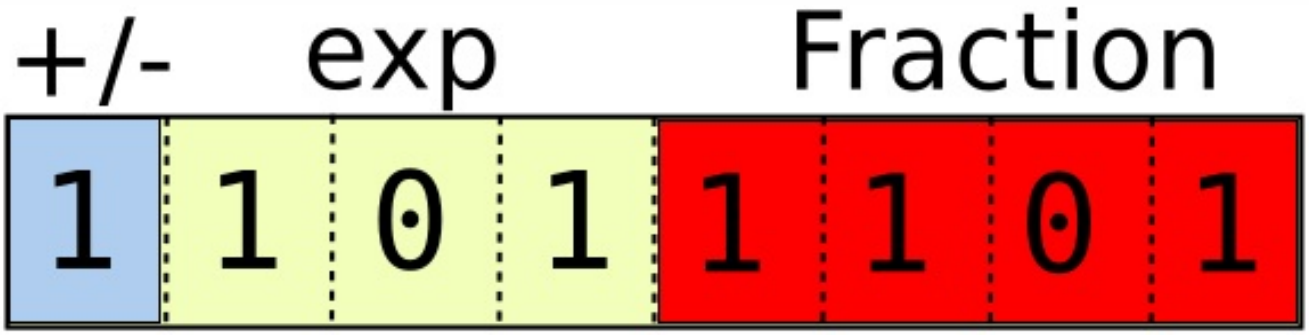
Les types de données et leur représentation en mémoire
- decimal : 123
- Binaire : **0b**1111011
- Octal : **0**173
- Hexadécimal :**0x**7B

On peut obtenir la taille en mémoire d'un type de données avec

```
sizeof(DATA_TYPE)
```

Les **nombres signés** sont représentés sous la forme : Signe (négatif si = 1) - Nombre

Standard IEEE 754



Les tableaux

Dans les anciennes version du langage langage C, les tableaux étaient de taille fixe.
Attention pas de Tab.length en C il faut donc prévoir de garder la taille du tableau en mémoire si on veut l'utiliser par la suite

```
#define N 10
float vecteur[N]
float matrice[N][N]
```

Caractères et chaines de caractères

Le langage C n'integre pas d'office les boolean et les strings. En C, les strings sont des tableaux de caractères dont le dernier élément contient la valeur '\0'.

```
char string[20]= "text";
printf("%s \n",string);
```

Les lettres étant des char (ASCII de 7 ou 8 bits) stockés sous la forme d'entiers, on peut donc effectuer des manipulations numériques avec des char.

Encodage	Particularité
ASCII	caractère Anglais
ISO8859	Latin avec Accents
UNICODE	Tous les caractères dans toutes les langues

Il n'existe pas de mécanisme d'exception en C. Celà pose des problèmes sécurité & des possibilités de Buffer Overflow.

Une chaine de caractères se termine toujours par un '\0' (equivalent à 0)

Les Pointeurs

En C, le programmeur peut interragir directement avec la mémoire où les données qu'un programme manipule sont stockés. En C, contrairement au java. Il n'y a pas de garbage collection qui retire de la mémoire les objets qui ne sont plus utilisés.

La mémoire est une zone qui est définie et accessible via son adresse.

Un pointeur est une variable contenant l'adresse d'une autre variable.

```
&var // adresse à laquelle une variable est stockée
var // variable en mémoire
*ptr // récupère la valeur à l'adresse du pointeur
```

Les structures

En C, contrairement au java (et autres langages orientés objet) on ne peut pas créer d'objets mais on peut créer des types de données (appelés structures). Les Structures n'ont pas de méthodes liés via l'encapsulation dans la classe.

Une **structure** est une combinaison de différents types de données simples ou structurés.
Dans les premières version du langage, les structures avaiant une taille fixe

```
struct NOM-STRUCTURE
    int VARIABLE1;
    int VARIABLE2;
}

struct NOM-STRUCTURE NOM-INSTANCE = {1,2}; // crée une instance
NOM-INSTANCE.VARIABLE1= 2 // accède directement à la variable en mémoire
(*ptr).x // accède à l'élèm x du ptr
ptr->x //idem supp
```

Les Alias

On peut redéfinir des noms de structures :

```
typedef int ENTIER; //redéfini int par entier
```

On peut les utiliser pour :
- la portabilité de l'app
- diminuer la taille des identifiants
- redéfinir des pointeurs (attention car un ptr reste un ptr)

Les fonctions

En C les fonctions et les pointeurs peuvent être utilisés en argument.

Manipulation bits

```
r = ~a;  //négation bit à bit
r = a & b; // conjonction bit à bit
r = a | b; // disjonction bit à bit
r = a ^ b; // xor bit à bit
a = n >> B // décale bits n de B bits
```

Section 2.3 : Declarations

Les variables sont définies par leur portée. **La portée d'une variable** peut-être définie comme la partie du programme ou la variable est accessible et où sa valaur peut-être modifiée.

Les variables globales sont des variables accessibles de partout dans le programme. Elles doivent être utilisées de façon parcimonieuses (utilisation mémoire + importante). Pour les variables locales, les premières versions du langage C imposaient leur définition au début des blocs.

Pour définir des constantes on peut :

```
#define M_PI 3.14159265 //préprocesseur
const double pi=3.14159265 // constante
```

Dans les premières versions de C on devait définir les variables au début de chaque bloc

Section 2.4 : Unions et énumérations

enum est utilisé pour définir un type de donées énumérées. càd un nombre fixe de valeurs possibles. (val stockées sous la forme d'entiers)

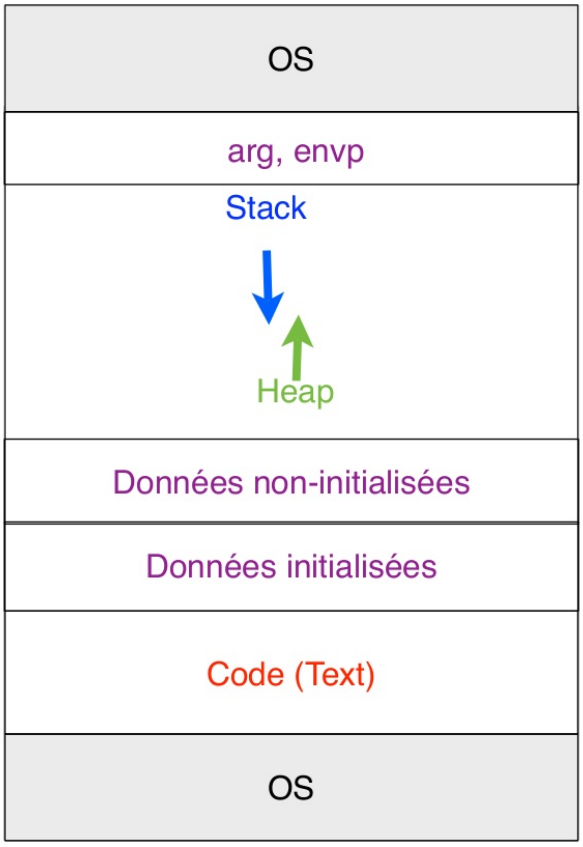
```
typedef enum{
    monday, tuesday, wednesday, thursday, friday, saturnday, sunday
}day;
day jour = monday;
```

union permet de réserver une zone en mémoire pour stocker plusieurs types de variables possibles

```
union u_t{
    int i;
    char c;
}u;
u.i = 12; // si cette variable contient un int, elle ne peut plus contenir de char sans supprimer la valeur du i
```

Section 2.5 : L'organisation de la mémoire

La mémoire peut-être divisée en six zones principales :



Le segment text

Contient toutes les instruction qui sont exécutées par le microprocesseur. (uniquement accessible en lecture)

Le segment des données initialisées

Contient l'ensemble des données et chaînes de caractères qui sont utilisées dans le programme. (il comprend l'ensemble des variables globales déjà initialisées)

Le segment des données non-initialisées

Contient les valeurs des variables non-globales

Le tas (ou heap)

C'est dans une des 2 zones dans laquelle un programme peur obtenir de la mémoire supplémentaire pour y stocker de l'information. Un programmeur peut réserver une zone permettant de stocker des données et y associer un pointeur. (brk(2) et sbrk(2) modifient la taille du heap).
En pratique, on utilise malloc(3) pour allouer de la mémoire et free(3) pour la libérer.
aloué manuellement en mémoire (variables)

Malloc, contrairement à calloc ne réinitialise pas la zone mémoire libérée.

Malloc

```
string= (char *) malloc(length*sizeof(char))//malloc retourne un ptr void qu'il faut caster ensuite
free(string);
```

Calloc

```
void *calloc(size_t num_element, size_t size); // base
char *ptr = calloc(15,size(char));//exemple
```

La pile (ou stack)

Cette zone est très importante, elle sotcke l'ensemble des variables locales mais également les variables de retour de toutes ls fonction qui sont appelées. Cette zone est gérée comme une pile.
alloué en automatiquement en mémoire (functions)

Les arguments et variables d'environnement

argc : nombre d'arguments
char* argv[] : les arguments
argv[0]: nom du programme exécuté

Cours S4

Dram : condensateur - gourmand en énergie, 50ns
Sram : consomation en continue => production de chaleur , 1ns
Dram : GB
Sram : MB
Posibilité d'avoir les deux avantages ?
on doit mettre dans la sram les données en cours d'utilisation (aussi appelé Cache)

Mémoire

Code-Données-Heap-Stack

Principe de localité :
- spatiale : si on edite un élément, il est courrant d'accéder à une variable proche en mémoire
- Si on a accédé à l'adresse X à l'instant t, il est commun d'accéder à la même adresse X à l'instant t+1

En pratique, on a une hierarchie de mémoire caches. on sépare la cache insruction de la cache données.

il existe une cache dans le processeur.

Cours S5

Cours S6

livelock = le processeur tourne mais rien n'est exécuté

Cours S7

voir slide

Cours S8

ar -> archive

2 formes de lib
-stat
on incl les librairies manuellement dans le makefile

-dynamiques
il est inutile de sauvegarder des librairies dans chaque exécutable si la librairie est présente dans tous les fichiers
référence vers une librairie en mémoire
+efficace : mémoire
Attentoion que elle soit bien présente sur le système

- Matériel
- Kernel(drivers[abstraction du matériel], interruptions, ...)
- processus système
- Applications

Image sur les slides

appels systèmes : abstraction pour intéragir avec le noyau
dnas la section 2 du manuel
* getpid : n° du processus système
* read : lire des fichiers
* kill : tuer le processus
* brk : mémoire (utilisé par malloc)

appel système :
1. Appeler le kernel
2. Quel appel système ?
3. Passer les arguments
4. Exécuter appel système
5. Retourner le résultat
6. Retour au processus

fork : copie presque identique en mémoire (pid !=) copie ses data (contexte != stack et heap ...)

père waitpid()
execve("hello"): remplace le programme par un exécutable . suicide par execve (pid =)
ensuite il fait appel à exit
préviens le père qu'il a fini****