Chapitre 1 - Introduction à l'environnement de programmation (Unix et compilation C)

Georges-Pierre BONNEAU (cours) - Mica MURPHY (note) - Antoine SAGET (note)

Lundi 1er Octobre 2018

0. Informations pratiques

- Évaluation
  - Théorique : examen final
  - Pratique : TPs et projet
  - Note finale : $\frac{2}{3}E + \frac{1}{3}\left[\frac{1}{2}(TP + P)\right]$

1. Centralisation des ressources

Unix : système d'exploitation de ressources informatiques amenées à être partagées

Nécessité de pouvoir accéder à ses fichiers, et exécuter ses programmes quelque soit la machine devant laquelle on se trouve :

- Quelques "serveurs" centralisant les moyens : `mandelbrot`, `goedel`, `turing`, `mandelbrot.e.ujf-grenoble.fr`...
- Une multitute de "clients" permettant de se connecter à ses serveurs : les PC dans toutes les salles

2. Serveurs : Unix

Environnement de travail multi-utilisateurs, partage de ressources mémoire et CPU entre les utilisateurs

```
who -q #affiche tous les utilisateurs courants
```

Accès au serveur depuis le client

3. Intéraction avec le serveur

Pas de clics, pas d'icônes, pas de glissement d'icônes ; intéraction uniquement au clavier.

```
<prog> <arg1> <arg2> ... <argn> <return>
```

Le programme `prog` est exécuté en tenant compte des arguments `arg1`, ..., `argn`. Le programme peut "afficher des résultats", i.e. émettre des caractères vers la fenêtre.

4. Système de fichiers

Les fichiers sont stockés sous forme hiérarchique par un arbre dont les feuilles désignent les fichiers, et les autres nœuds des « boites », appelées répertoires, et permettant de classer l'information.

Une de ces boites vous est réservée: elle porte le nom de votre login, dont les données sont protégées avec des droit d'accès aux répertoires et fichiers.

Commandes utiles :

- Changement de répertoire : `cd <nom du répertoire`
- Répertoire courant : `./`
- Répertoire parent : `../`
- Désignation absolue d'un répertoire/fichier : `cd /users/john`
- Désignation relative : `cd ../users/john/images`
- Création d'un répertoire : `mkdir ../mary/music`
- Listing d'un répertoire : `ls`, `ls -la`
- Connaître l'emplacement du répertoire courant : `pwd`

5. Droits d'accès aux fichiers

Droits d'accès : en lecture, écriture, exécution / pour le propriétaire, le groupe, les autres

```
-rw-r--r--  1 xia  xia   3397 sept. 11 09:22 .zshrc
```

6. Création et édition d'un fichier

```
gedit <nom de fichier> # graphique
nano <nom de fichier> # dans le terminal
```

7. Variable d'environnement PATH

```
which <nom du programme # indique le répertoire ou se trouve le programme
echo $PATH # affiche la liste des répertoires
export PATH="" # changer la valeur de la variable
```

La variable PATH indique au système où se trouvent les programmes. Elle contient une liste de répertoires dans lesquels le système recherche les programmes. Elle est réinitialisée à chaque nouveau terminal.

8. Programmation

édition $\Leftrightarrow$ source $\rightarrow$ (compilation) $\rightarrow$ exécutable $\Rightarrow$ exécution

- Édition : `nedit <programme>.c`
- Compilation : `gcc <programme>.c  -o <programme>`
- Exécution : `./<programme>`

9. Métaphore avec un système dynamique en physique

Introduction

Système dynamique continu

- État, variables d'état
  - Position, vitesse, temps
- Lois physiques faisant évoluer l'état, fonctions
  - m×a = g
  - v'(t) = a
  - p'(t) = v(t)
- Discrétisation
  - v(t+dt) = v(t) + dt×a
  - p(t+dt) = p(t) + dt×v

Système dynamique discret : le jeu d'échecs

- Position de chacune des 32 pièces définie par deux variables de types différents :
  - nom : "Roi Noir"
  - valeur : "A..H-1..8" ou "dehors"
- Actions : déplacements
- **Variable.** Association d'un nom et d'une valeur typée
- **État.** Ensemble des variables
- **Actions.** Déterminent l'évolution du système (avant/après une action)
  - *Actions élémentaires.* Un déplacement
  - *Actions composées.* Succession d'actions élémentaires
- La faisabilité d'une action dépend de l'état

Algorithme :

- État initial
- État final
- Spécification : décrit ce que l'algorithme fait, sans dire comment il le fait (état initial → état final)
- Assertion : propriété vérifiée par un état
- *Exemple.* Spécification : `3 variables a, b, t et 2 valeurs a0, b0  {ici a vaut a0, b vaut b0, t quelconque}  t <- a  {t vaut a0}  a <- b  {a vaut b0}  b <- t  {b vaut a0, a vaut b0, t vaut a0}`

Factoriel

- l'action : (état =) donnée N (entier) et résultat R (entier)
- état initial : (assertion =) N entier naturel, R indéfini
- état final : R vaut N!
- lexique : K (entier naturel) ← *variables inutiles dans la spécification de l'action*
- algorithme : `R <- 1  K <- N  v {propriété}  Tant que (K > 1) faire R <- R * K  K <- K - 1  {ici, R vaut N*(N - 1)*...*(K + 1)}  ^ {assertion à l'intérieur d'une itération = invariant}  {ici, (K vaut 0 et R vaut 1) ou (K vaut 1 et R vaut N!)}`
- à la sortie de l'itération $K \leq 1$
- 1er cas on est rentré au moins une fois dans l'itération
  - c'est le cas ssi $N > 1$
  - dernière étape de l'itération $K = 2$ à l'entrée, $K = 1$ à la fin. $R = N * (N-1) * \cdots * 2 = N!$
- 2ème cas : on ne rentre pas dans l'itération donc K vaut 0 ou 1 avant l'itération.

  Les assertions et les invariants se prouvent.

  Pour les invariants, ici on fait une **preuve par récurrence** comme ils se trouvent dans une boucle - vrai à la 1ère itération ? - supposer que l'assertion est vraie à une itération donnée → montrer que l'assertion est vraie à l'itération suivante (pas forcément $P(n)$, $P(n+1)$)

  Important : la signification de l'invariant à la sortie de l'itération

Algorithme de tri par insertion

Il y a une bijection entre le tableau de départ et le tableau trié.

Tableau de départ :

1	2	3	4	5
13	2	9	9	5

Tableau d'arrivée (avec indices de départ) :

3	5	1	4	2
13	2	9	9	5

Bijection $\begin{cases} 3 \to 1 \\ 5 \to 2 \\ 1 \to 3 \\ 4 \to 4 \\ 2 \to 5 \end{cases}$ de $\{1, \ldots, 5\}$ sur $\{1, \ldots, 5\}$ (= **permutations**)

Démonstration d'un invariant

- Initialisation
- Conservation

État initial

$$\begin{cases} j \text{ entier entre 2 et } N \\ T[i] \text{ vaut } T[j] \\ Clef \text{ vaut } T[j] \\ T \text{ contient } \{\overline{t_1}, \ldots, \overline{t_n}\} \end{cases}$$

Preuve de l'invariant I_2

- Initialisation :
  - L5 : comme on rentre dans la boucle, $\overline{t_i} = T[i] > Clef$
  - L6 : on modifie le tableau (voir PDF) et $i = j - 1$ (et $i + 1 = j$)
  - L7 : $i = j - 2$ (et $i + 1 = j - 1$)
  - $Clef < \overline{t_i} = \overline{t_{i+1}}$ en fin de boucle
- Conservation :
  - Voir document PDF

  Terminaison (facultatif) : que peut-on affirmer en sortie de boucle ? - on sait que l'invariant est vérifié à la dernière itération - on sait qu'on est sortis de la boucle : - soit $i = 0$ et alors : - T vaut $[\overline{t_1}, \overline{t_1}, \ldots, \overline{t_{j-1}}, \overline{t_{j+1}}, \ldots, \overline{t_N}]$ - $Clef < \overline{t_1}$ - soit $i \geq 1$ et $T[i] \leq Clef$: - T vaut $[\overline{t_1}, \ldots, \overline{t_i}, \overline{t_{i+1}}, \overline{t_{i+1}}, \ldots, \overline{t_{j-1}}, \overline{t_{j+1}}, \ldots, \overline{t_N}]$ - $Clef \geq T[1 \ \ldots i]$ et $Clef < T[i+1 \ldots j-1]$