

# EPU - Informatique ROB4

## Informatique Système

### Cours Introductif

**Sovannara Hak, Jean-Baptiste Mouret**  
[hak@isir.upmc.fr](mailto:hak@isir.upmc.fr)

Université Pierre et Marie Curie  
Institut des Systèmes Intelligents et de Robotique (CNRS UMR 7222)

2014-2015



# Plan de ce cours

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

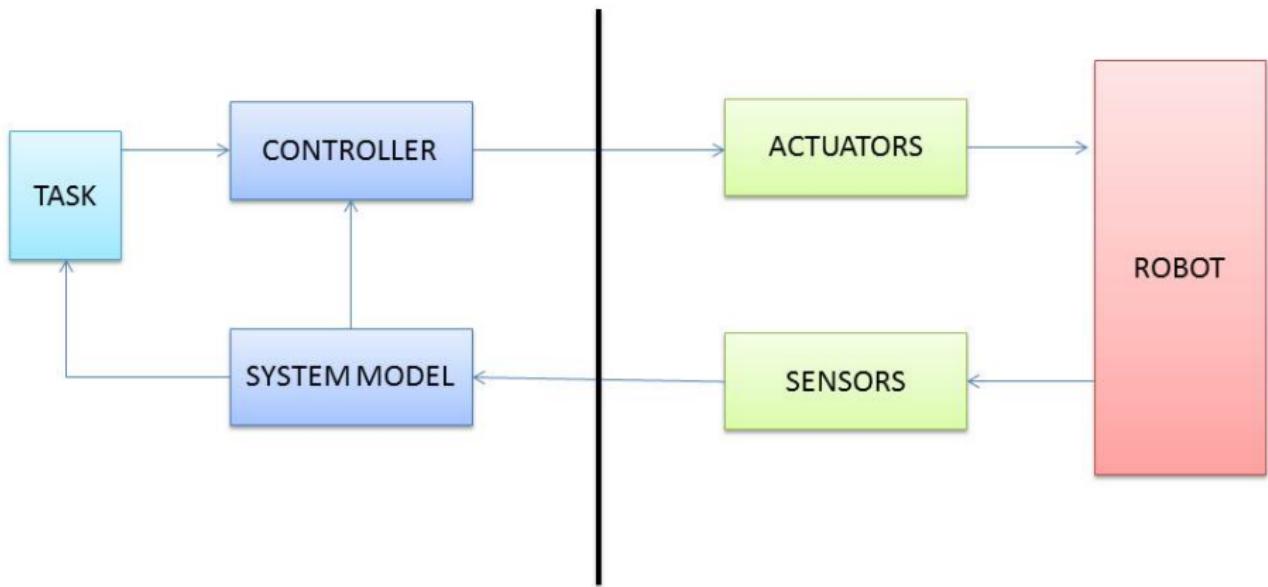
## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

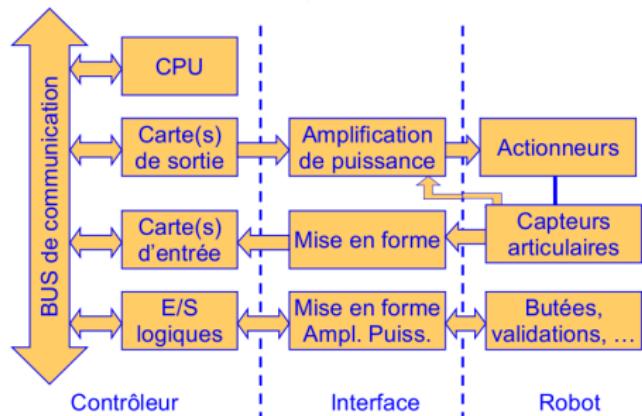
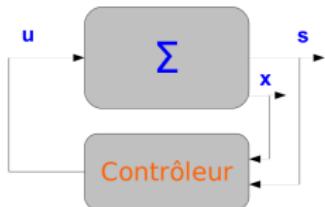
# A classical control loop in robotics



## → Quel est le rôle du contrôleur en Robotique ?



©Honda Research



# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- **Synthèse matérielle et logicielle du contrôleur en Robotique**

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

## → Comment synthétiser le contrôleur ?

### Premières solutions : DSPs et $\mu$ contrôleurs

- Nombre d'E/S (relativement) limité
- Types de protocole de communication limités
- Capacité de calcul "faible"

### Seconde solutions : architectures "PC" Intel ix86, PowerPC, Motorola 68K, ...

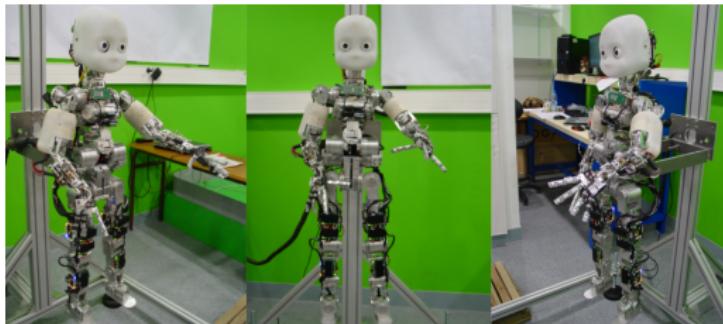
- Solutions versatiles et adaptables
- Un grand nombre de configurations matérielles possibles : PC classique, "Gumstix" PC, PC104 | multi-processeurs, mémoire partagée ...
- Systèmes d'Exploitations (temps-réel) dédiés : VxWorks, QNX, RTlinux, Xenomai ...

## → Comment synthétiser le contrôleur ?

### Solutions hybrides

- Contrôleurs locaux (au niveau des actionneurs) basés sur des DSPs ou des  $\mu$ contrôleurs
  - ↪ Déport d'une partie des calculs (asservissement bas-niveau, traitement des signaux des capteurs).
- Architecture de type PC au niveau central
  - ↪ Centralisation des informations et calculs haut-niveau (coordination des mouvements, fusion données capteurs).
- Communication entre le haut et le bas niveau au travers de bus avec protocoles standardisés.

### ↪ Approche justifiée pour des robots à grand nombre de ddl



# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ **Système d'exploitation (Operating System, OS) : ensemble de programmes à l'interface entre le matériel de l'ordinateur et les utilisateurs.**

### Définition générale

- Prise en charge de la gestion des ressources (processeur, mémoire et périphériques) et de leur partage
- Machine virtuelle, "au-dessus du matériel", facile d'emploi et conviviale (requêtes standardisées indépendantes du type de matériel).

### Rôles principaux

- Partage du processeur : parmi tous les programmes chargés en mémoire centrale, lequel doit s'exécuter à un instant donné ?
- Allocation de la mémoire centrale aux différents programmes : comment assurer la protection des zones mémoires entre différents programmes en cours d'exécution ?
- Traitements des requêtes d'accès (E/S) aux différents périphériques ?

→ **L'OS doit assurer l'équité d'accès à la machine physique et aux ressources matérielles entre les différents programmes et la cohérence de ces accès.**

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS**
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

## → Quels types d'OS pour quels types de besoin ?

### Les "ancêtres" : OS à traitement par lots (batch)

- Maximisation de l'utilisation du processeur
- Pas d'interactions possibles avec l'utilisateur au cours de l'exécution.

### OS interactifs (Windows, Unix, Linux, BSD,...)

- Interaction entre le(s) utilisateur(s) et le système même au cours de l'exécution d'un programme
- Utilisation multi-utilisateurs " transparente "
- Partage du temps processeur entre les différents processus et les différents utilisateurs (système en temps partagé).

### OS temps-réel (VxWorks, RTlinux, QNX, Xenomai ...)

- Réagissent en un temps adapté aux évènements externes ;
- Fonctionnent en continu sans réduire le débit du flot d'information à traiter ;
- Temps de calcul connus et modélisables pour permettre l'analyse de la réactivité ;
- Latence maîtrisée.

## → Quels types d'OS pour quels types de besoin ?

Les "ancêtres" : OS à traitement par lots (batch)

...

OS interactifs (Windows, Unix, Linux, BSD,...)

...

OS temps-réel (VxWorks, RTlinux, QNX ...)

- Réagissent en un temps adapté aux événements externes ;
- Fonctionnent en continu sans réduire le débit du flux d'information à traiter ;
- Temps de calcul connus et modélisables pour permettre l'analyse de la réactivité ;
- Latence maîtrisée.

OS temps-réel : remarques

- Utile dans le cas de **contraintes temporelles** : " j'ai absolument besoin du résultat de cette opération à tel instant ".
- Ce n'est pas temps le temps d'exécution qui est primordial mais la capacité à fournir un résultat à un instant précis ou à un **intervalle de temps régulier** (qui peut aller de la ms à l'heure en fonction du système physique associé).
- Un **mécanisme de préemption** permet l'interruption à tout moment de l'exécution d'un processus si un processus de priorité plus élevée apparaît.

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- **Les particularités de Linux**
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

## → UNIX



Ken Thompson (assis) et Dennis Ritchie en 1972, Bell Labs, invention de Unix.

"The strange birth and long life of Unix", W. Toomey, IEEE Spectrum, Dec. 2011.

```
11/3/71          SYS SEEK (II)

NAME      seek -- move read/write pointer
SYNOPSIS (file descriptor in r0)
          sys seek offseti ptrname / seek = 19.
DESCRIPTION The file descriptor refers to a file open for
             reading or writing. The read (or write) pointer
             for the file is set as follows:
               if ptrname is 0, the pointer is set to offseti.
               if ptrname is 1, the pointer is set to its
               current location plus offseti.
               if ptrname is 2, the pointer is set to the
               size of the file plus offseti.

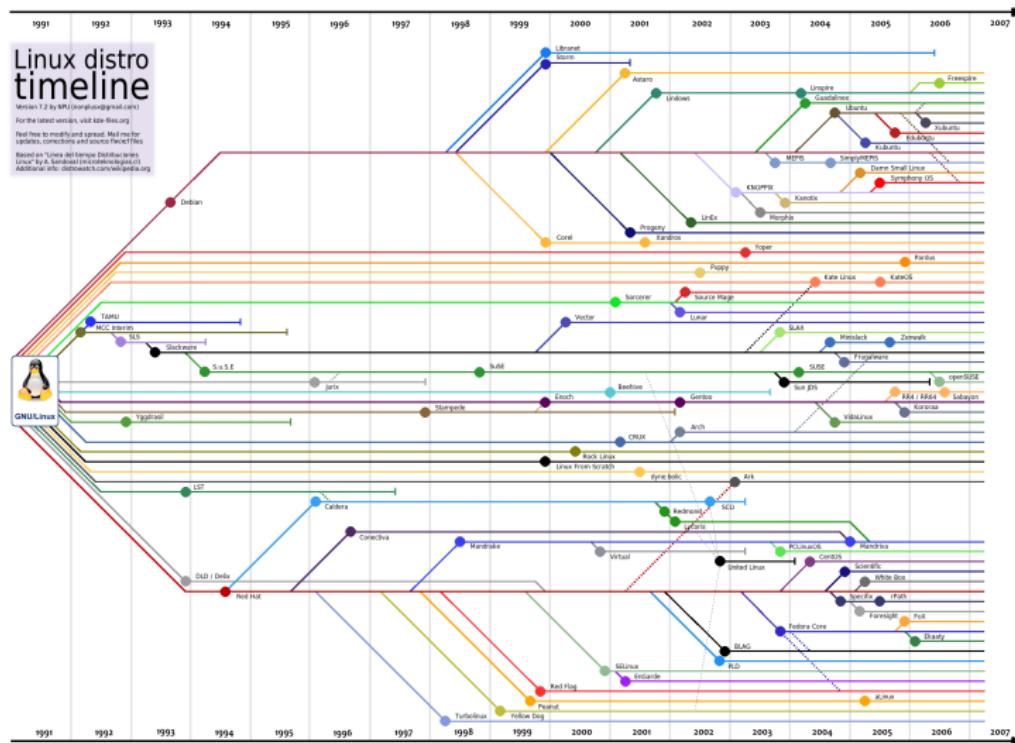
FILES      --
SEE ALSO   tell
DIAGNOSTICS The error bit (c-bit) is set for an undefined
             file descriptor.
BUGS       A file can conceptually be as large as 2**20
             bytes. Clearly only 2**16 bytes can be addressed
             by seek. This problem is most acute on the tape
             file and R/W and R/F. Something is going to be
             done about this.
OWNER      ken, dmr
```

Une page du premier manuel Unix or *man* (Novembre 1971), par Thompson (ken) et Ritchie (dmr).

## → Quelques faits à propos de Linux (que vous utiliserez en TP) :

- Système compatible avec la norme POSIX pour les OS.
- Système né en 1991 (Linus Torvalds) et membre de la famille des systèmes de type Unix.
- Code source libre d'accès (licence publique GNU) et donc système vivant s'enrichissant constamment.
- Programmé dans un langage puissant et " haut niveau " : C/C++.
- Système interactif (multi-utilisateurs, multi-tâches) avec des aspects compatibles avec la problématique du temps réel faiblement contraint.
- Le système s'organise autour du noyau (kernel) qui représente l'ensemble des programmes nécessaires au bon fonctionnement du système.
- Le noyau peut être vu comme un gestionnaire de processus.
- Le noyau est modulaire et peut être modifié hors ligne (recompilation) et dynamiquement.

→ Ne pas confondre Linux (le noyau) et les distributions Linux (Debian, Mandriva, Ubuntu, Gentoo, Fedora, Slackware ...).



# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- **Couches d'abstraction logicielles**

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

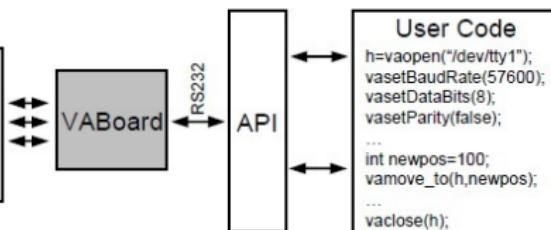
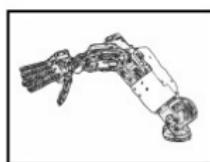
- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

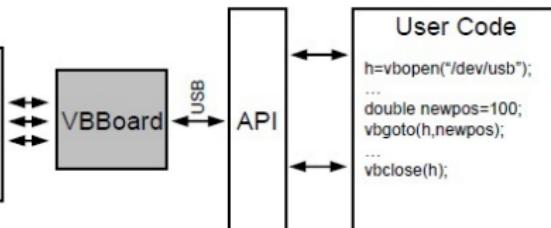
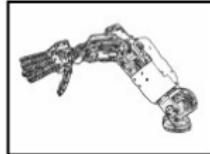
- Allocation mémoire
- Les pointeurs

## Q : Pourquoi a-t-on besoin d'une couche d'abstraction en robotiques ?

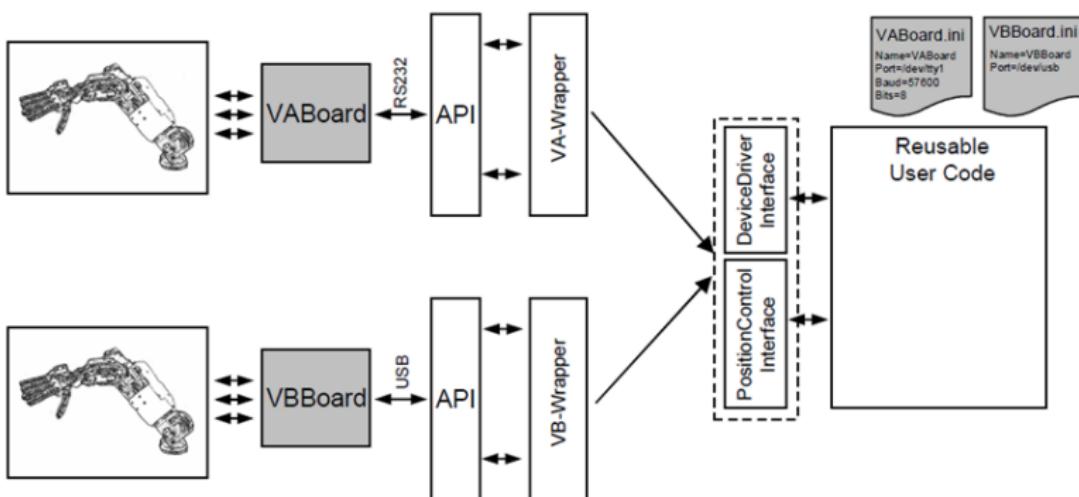
A)



B)



## R : Re-utilisation de code, abstraction du matériel, ...



## R : Support de plusieurs robots..



[Fraunhofer IPA  
Care-O-bot](#)



[Videre Erratic](#)



[TurtleBot](#)



[Aldebaran Nao](#)



[Lego NXT](#)



[Shadow Robot](#)



[Willow Garage PR2](#)



[iRobot Roomba](#)



[Robotnik  
Guardian](#)



[Merlin miabotPro](#)



[AscTec Quadrotor](#)



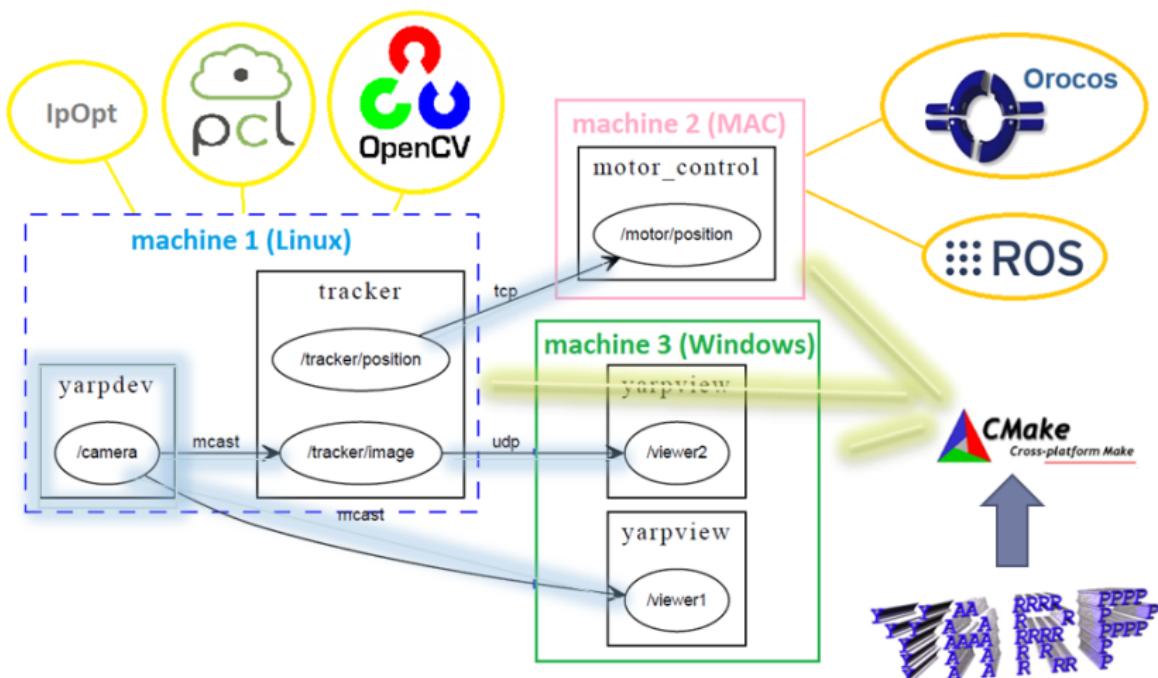
[Clearpath Robotics  
Husky](#)



[Clearpath Robotics  
Kingfisher](#)

ROS ([www.ros.org](http://www.ros.org))

## R : interopérabilité entre plusieurs OS..



→ Au delà des OS au sens classique du terme, des couches d'abstraction ont été développées spécifiquement pour la Robotique :

- Abstraction des aspects purement matériels (drivers, interfaces) et fournissant un cadre de développement de plus haut niveau.
- Ces outils fournissent souvent des mécanismes de gestion des aspects multi-tâches, mémoire et communication. Ceci permet à l'utilisateur d'utiliser au mieux les ressources du système sans avoir à en connaître parfaitement les détails.

#### Les solutions existantes

- Player/Stage, GenOM, Yarp, Orocos, ROS
- Urbi, DevBot, Microsoft Robotics Studio (MRS)
- ...

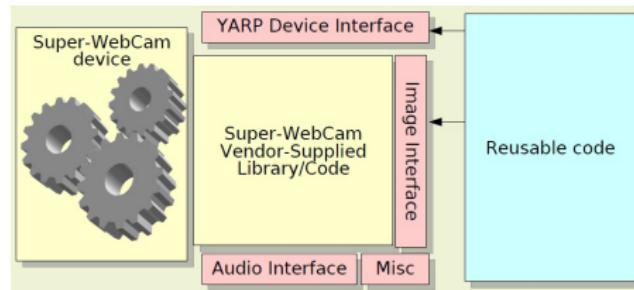
→ Ces solutions ne remplacent pas l'OS. Elles fournissent juste une couche logicielle supplémentaire spécifique aux problématiques de la Robotique.

## Exemple : YARP

*"... YARP is plumbing for robot software. It is a set of libraries, protocols, and tools to keep modules and devices cleanly decoupled. It is reluctant middleware, with no desire or expectation to be in control of your system. YARP is definitely not an operating system. ..."*

P. Fitzpatrick

- Ré-utilisation de code : rendre logiciels robotiques plus stable et durable
- abstraction du matériel : indépendance des capteurs, actionneurs, processeurs, and réseaux
- changement de matériel moins perturbant
- minimiser les problèmes d'incompatibilité d'architecture, framework, et middleware
- gratuit, opensource (LGPL)



# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ L'OS réalise une couche logicielle placée entre la machine matérielle et les applications. Il peut être découpé en plusieurs grandes fonctionnalités :

- **Gestion du processeur** : l'allocation du temps processeur aux différents programmes en cours d'exécution est réalisé par un ordonnanceur qui planifie l'exécution des programmes (il existe différentes politiques d'ordonnancement : FIFO, priorités, tourniquet...).
- **Gestion des objets externes** : la mémoire centrale est volatile et cela nécessite l'utilisation de périphériques de mémoire de masse (disques durs....). La gestion de l'allocation des mémoires de masse ainsi que l'accès aux données stockées s'appuie sur le système de gestion de fichiers.
- **Gestion des accès E/S aux périphériques** : gestion du lien entre les appels de haut niveau des programmes utilisateurs (par exemple getchar()) et les opérations de bas niveau de l'unité d'échange responsable du périphérique (ici le clavier). Ce lien est assuré par les pilotes d'E/S (drivers).
- **Gestion de la mémoire centrale** : la mémoire physique étant limitée, seuls les codes en cours d'exécution sont placées en mémoire centrale. Les autres données en mémoire sont placées en mémoire virtuelle.
- **Gestion de la concurrence** : accès cohérents aux données partagées par plusieurs processus au travers d'outils de synchronisation et de communications entre processus.
- **Gestion de la protection** : limite les accès possibles des programmes utilisateurs afin de préserver les ressources (processeur, mémoire, fichiers). Modes d'exécution utilisateur et superviseur.
- **Gestion de l'accès au réseau** : couche de protocole permettant les échanges avec des machines distantes.

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- **Le processeur**
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ Le processeur a pour rôle l'exécution des programmes. Les programmes *utilisateur* accèdent aux fonctionnalités de l'OS au travers d'appels aux *routines systèmes*.

## Caractéristiques

- Vitesse d'horloge
- Jeu d'instruction (propre au constructeur : Intel, PowerPC, Motorola...)
- Etat observable entre l'exécution de 2 instructions uniquement
- Zone mémoire dédiée (la **pile**) pour l'exécution des programmes (appel de sous-programmes, traitement d'interruptions).

## Contexte d'exécution d'un programme

- Compteur de programme
- Pointeur de pile
- Etat du processeur
- Registres généraux

## Il existe deux modes d'exécution possibles

- **Superviseur** : toutes les instructions et toute la mémoire sont accessibles ;
- **Utilisateur** : accès restreint à certaines instructions et adresses mémoire.

→ **Processus** : instance d'exécution d'un programme par le processeur.

→ **processus  $\neq$  programme**

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- **Les interruptions**
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ **Les interruptions permettent la prise en compte asynchrones d'évènements externes au processeur.**

- La prise en compte d'une interruption provoque l'arrêt (pas nécessairement définitif) du programme en cours et l'exécution du sous-programme associé à l'interruption.
- La gestion des interruptions peut être **hiérarchisée** : dans ce cas les interruptions de priorité basse ne peuvent pas interrompre les interruptions de priorité plus élevée.
- Certaines interruptions sont **masquables**, *i.e.* elles doivent être ignorées par certains programmes.
- La table de pointeurs vers les sous-programmes de traitement associés aux interruptions est appelée **vecteur d'interruption**.
- Les déroutements sont des interruptions non masquables internes au processeur. Ils permettent la prise en compte d'erreurs du type : violation de mode d'exécution, violation d'accès mémoire, erreur d'adressage, erreur arithmétique (division par 0, débordement de registre...),...

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- **Les Entrées/Sorties (IO)**

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ **La gestion des I/O se fait au travers d'un bus. Le rôle du bus est d'assurer la connexion des périphériques au processeurs.** Il existe différents protocoles de bus, les plus standards étant :

- PCI : bus parallèle 32 ou 64 bits, standardisé par Intel. Nombreuses variantes : PCI-Express, Compact-PCI, Mini-PCI,...
- USB : Universal Serial Bus. Bus d'I/O série grand public.
- I2C : Inter-Integrated Circuit bus, standardisé par Philips. Bus série 2 fils simple pour interconnexion de périphériques. Utilisé pour capteurs de température, information sur les moniteurs,...
- CAN Controller Area Network. Conçu par l'industrie automobile. Communication temps-réel entre  $\mu$ contrôleurs et capteurs/actionneurs.

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

## Langage C

### Histoire

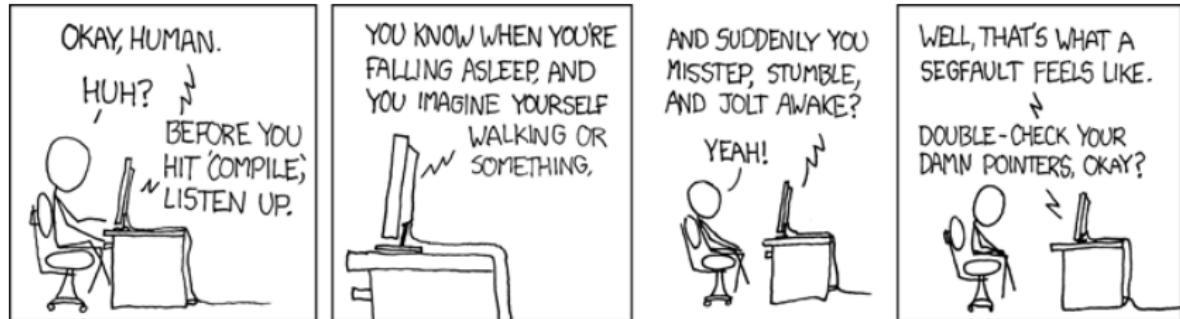
- 1972 : Dennis Ritchie - AT&T Bell Labs
- 1978 : "The C programming language" published
- 1989 : C89 standard, aka ANSI C or standard C
- 1990 : C90 is adopted by ISO
- 1999 : C99 standard
- 2007 : new C standard C1X announced

### Très utilisé

- OS, like Linux
- $\mu$ controllers
- embedded processors
- DSP processors

### Particularités

- Avantages : peu de mots clefs, structures, pointeurs (memoire, tableaux, ..), librairies externe, code rapide
- Inconvénient : pas d'exceptions, pas de programmation orienté objet, pas de polymorphisme



(original comics on [xkcd.com](http://xkcd.com))

## Attention ! C est intrinsèquement peu sûr !

- Bas niveau
- Pas de gestion d'exceptions
- Pas de vérification des limites mémoires
- Pas de vérification de types à l'exécution

## Ecrire du code C

Edition : hello.c

```
/* hello.c - the simplest program */
#include<stdio.h>
int main(int argc, char* argv[])
{
    printf("Hello!");
    return 0;
}
```

Compilation :

```
gcc -Wall hello.c -o hello.o
```

## Ecrire du code C

### Quelques fonctions pratiques

Dans #include <stdlib.h> :

- Conversion d'une chaîne en entier : int atoi(const char \*nptr);
- Conversion d'une chaîne en double : double atof(const char \*nptr);

Dans #include <string.h> :

- Copie de n caractères d'une chaîne vers une autre :  
`char *strncpy(char *dest, const char *src, size_t n);`
- Comparaison de deux chaînes : int strcmp(const char \*s1, const char \*s2);

→ Prototype générique de la fonction main : int main(int argc, char\* argv[]);

- La fonction main d'un programme peut prendre des arguments en ligne de commande.
- Exemple : » cp fichier1.tex fichier2.tex
- La récupération des ces arguments se fait au moyen des arguments argc et argv.
- argc : nombre d'arguments passés au main + 1
- argv : tableau de pointeurs sur des chaînes de caractères :
  - le premier élément argv[0] pointe sur la chaîne qui contient le nom du fichier exécutable du programme.
  - les éléments suivants, argv[1], argv[2], ..., argv[argc-1] pointent sur les chaînes correspondants aux arguments passés en ligne de commande.

↪ Les arguments du main offrent de nombreuses possibilités.

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- **La chaîne de compilation**
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ La production d'un programme exécutable à partir d'un code écrit en C répond à plusieurs règles.

- Un et un seul des fichiers sources utilisés contient une fonction `main()` ;
- Cette fonction est la fonction principale du programme ;
- C'est à partir de `main()` que l'ensemble des traitements sera effectué soit directement, soit par appel à d'autres fonctions ;
- Ces autres fonctions peuvent être :
  - des fonctions standards et en général portables du C ;
  - des fonctions définies par le programmeur.

## Trois étapes

- ① préprocesseur
- ② compilation
- ③ édition de liens

→ Les étapes de compilation et d'édition de liens sont précédée d'un passage du préprocesseur. Le préprocesseur permet des manipulations post compilation et édition de liens.

- Inclusion de fichiers à l'aide de l'instruction `#include` :
  - Cette instruction permet l'inclusion de fichiers d'en-tête `.h`;
  - Ces fichiers permettent notamment de faire référence à des fonctions dont le symbole ne sera résolu qu'au moment de l'édition de liens;
  - Ils permettent aussi la définition de types, de structures...;
- Remplacement de morceaux de code par des constantes ou des macros en utilisant l'instruction `#define` ;

```
#define PI 3.14159
#define norm(x,y) (sqrt(pow(x,2.0)+pow(y,2.0)))
#define myfabs(x) ((x < 0) ? -x : x)
```

- la prise en compte conditionnelle de morceaux de code à la compilation avec `#ifdef...#endif` ou `#ifndef...#endif` ;

```
#ifndef CODROSLINKDOTH
#define CODROSLINKDOTH
....bloc pris en compte de manière conditionnelle...
#endif
```

## → La production du programme passe par une étape de *compilation*.

- Elle permet, à partir d'un fichier source .c, de produire un fichier objet .o ;
- Le fichier objet contient "une traduction" du code source C en un code compréhensible par le processeur (assembleur) ;



- GCC (GNU C COMPILER) projet démarré en 1985 par Richard Stallman
- `gcc -c foo.c` produit le fichier objet `foo.o` ;
- Les options classiquement passées à `gcc` à l'étape de compilation sont les suivantes :
  - `-g` pour rendre le débogage possible ;
  - `-Wall` pour afficher tous les avertissements ;
  - `-O0` pour imposer la non optimisation du code (par défaut)
  - ...

## → La compilation doit être suivie d'une *édition de liens*.

- Elle permet le rassemblement de l'ensemble des fichiers objet qui composent le programme ;
- Son rôle est la résolution de l'ensemble des références aux symboles non définis des différents fichiers objets ;
- L'éditeur de liens GNU est le programme `ld` qui est en général appelé de manière implicite via `gcc` ;
- `gcc -o mon_exec main.o titi.o tutu.o tata.o` produit l'exécutable `mon_exec` ;
- Les fichiers objets peuvent être regroupés en bibliothèques statiques `.a` à l'aide des commandes `ar` et `ranlib` :

```
ar cr libtoto.a titi.o tutu.o tata.o  
ranlib libtoto.a
```

- L'appel d'une bibliothèque à l'édition de liens se fait comme suit :  
`gcc -o mon_exec main.o -L ./lib -ltoto`
- L'appel à la bibliothèque standard de math se fait par `-lm`, `libm.a` étant le nom de cette bibliothèque ;
- L'option `-L chemin_de_la_lib` permet d'indiquer les chemins de bibliothèques qui ne seraient pas des bibliothèques standards du C (`/usr/lib`) ;
- Il existe aussi des bibliothèques qui sont liées dynamiquement (`.so` ou `.dll` sous Windows) ;

## Options communes :

### • Générique

- -E stop after preprocessing, do not compile
- -o *file* place output in file *file* (default *a.out*)

### • Optimisation

- -O0 no optimization
- -O1 optimize and minimize compile time
- -O2 more costly optimization
- -Os optimize for code size

### • langage C

- -ansi , -std=c90 , -std=iso9899:1990 to select the standard C
- -pedantic for all the diagnostics required by the specific standard
- -pedantic-errors to treat warning as errors

### • Debug et erreurs

- -g enables generation of debugging information, that can be used by gdb (works with -O0)
- -Werr make all warnings into errors
- -Wall enables all (main) warnings
- -Wextra enables extra warnings not enabled by -Wall (for example -Wsign-compare)

### • Threads

- -pthread support multithreading using POSIX threads library

### • Edition de liens/Linking

- -l*library* search the library named *library* when linking (default lib*library*.a), for example -lm links the math library
- -L*dir* add *dir* to the list of directories to be searched for -l

### Using the GNU Compiler Collection

For GCC version 4.1.1

(GCC)

Richard M. Stallman and the [GCC Developer Community](#)

## TEST 1

test.c

```
/* a test */
#include<stdio.h>
int main(int argc, char* argv[])
{
    puts("Hello students!"); //output text to stdout and end line
    return 0;
}
```

définition suivante :

```
#define DMSG "Hello students!"
```

Comment réécrire test.c ?

## TEST 2

```
#define DMSG = "Hello students!"
```

```
#include <stdio.h>;
```

```
int function (void arg)
{ return arg-1; }
```

Trouver l'erreur et la corriger :

## TEST 3

test.c

```
/* a test */
#include<stdio.h>
int main(int argc, char* argv[])
{
    puts("Hello students!"); //output text to stdout and end line
    return 0;
}
```

Comment compiler et obtenir le binaire test.o ?

## → Exemple simple

- Création de l'exécutable `my_exec`
- Fonction `main()` déclarée dans `main.c`
- `main()` appelle les fonctions `klatu()`, `verata()` et `nictu()` respectivement déclarées dans `klatu.c`, `verata.c` et `nictu.c`
- `klatu()` et `verata()` appellent `nictu()`

→ Comment générer l'exécutable `my_exec` ?

→ Avec ou sans bibliothèque ?

## → Exemple simple

- Création de l'exécutable `my_exec`
  - Fonction `main()` déclarée dans `main.c`
  - `main()` appelle les fonctions `klatu()`, `verata()` et `nictu()` respectivement déclarées dans `klatu.c`, `verata.c` et `nictu.c`
  - `klatu()` et `verata()` appellent `nictu()`
- Comment générer l'exécutable `my_exec` ?  
→ Avec ou sans bibliothèque ?

## Solution basique

### Compilation

```
» gcc -g -c klatu.c  
» gcc -g -c verata.c  
» gcc -g -c nictu.c  
» gcc -g -c main.c
```

### Édition de liens

```
» gcc -g -o my_exec main.o klatu.o verata.o nictu.o
```

## ↪ Exemple simple

### Solution basique

#### Compilation

```
» gcc -g -c klatu.c  
» gcc -g -c verata.c  
» gcc -g -c nictu.c  
» gcc -g -c main.c
```

#### Édition de liens

```
» gcc -g -o my_exec main.o tutu.o tata.o titi.o
```

### Solution utilisant une bibliothèque

#### Compilation

```
» gcc -g -c klatu.c  
» gcc -g -c verata.c  
» gcc -g -c nictu.c  
» gcc -g -c main.c
```

#### Création de la bibliothèque

```
» ar cr libgroovy.a klatu.o verata.o nictu.o  
» ranlib libgroovy.a
```

#### Édition de liens

```
» gcc -g -o my_exec2 main.o -L . -lgroovy
```

## Exemple simple ... mais quand-même ...

### Compilation

```
» gcc -g -c klatu.c  
» gcc -g -c verata.c  
» gcc -g -c nictu.c  
» gcc -g -c main.c
```

### Création de la bibliothèque

```
» ar cr libgroovy.a klatu.o verata.o nictu.o  
» ranlib libgroovy.a
```

### Édition de liens

```
» gcc -g -o my_exec2 main.o -L . -lgroovy
```

## Remarques

- Long et fastidieux au delà de 2 fichiers à compiler (souvent un projet peut en compter plusieurs centaines) !
- Quand un fichier source est modifié, il n'est pas nécessaire de tout recompiler : par souci d'efficacité il faut donc connaître les **dépendances** et ne recompiler et lier ce qui est nécessaire. Ceci n'est pas humainement faisable au delà de 5 fichiers sources.

→ **Il existe des outils qui permettent d'automatiser le processus de création de programme : make et pour les projets plus importants CMake, autotools, ...**

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- **Make et Makefile**
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ make est un programme permettant d'exécuter de manière automatisée l'ensemble des opérations nécessaires à l'obtention d'un exécutable

- Les instructions exécutées par make sont placées dans un fichier Makefile
- En utilisant les dates d'accès aux fichiers, make est capable de déterminer quels fichiers doivent être recompilés ou non → **gestion des dépendances**
- La seule connaissance strictement nécessaire pour l'écriture de fichiers Makefile "simples" est celle de l'écriture des règles :
  - CIBLE : DEPENDANCES  
COMMANDÉ
  - ...
  - CIBLES peut représenter le nom d'un fichier à produire (un fichier .o, un exécutable...) ou de manière plus générale, un nom quelconque (question2 par exemple)
  - DEPENDANCES représente l'ensemble des règles qui doivent avoir été exécutées et/ou l'ensemble des fichiers qui doivent exister afin de pouvoir lancer la commande COMMANDÉ.
- make est en fait beaucoup plus général que cela. Par exemple, ce document a été réalisé en utilisant make. (cadre général de la compilation en C).

→ Un bon document de référence sur make peut être trouvé à cette adresse :

<http://www.laas.fr/~matthieu/cours/make>

→ Make automatise la chaîne de compilation...CMake, Autotools et certaines IDE automatisent la production de Makefile (ou de tout autre mécanisme standardisé de description efficace de la chaîne de compilation)

## → Exemple simple

- L'exemple qui suit produit un exécutable `monprog` à partir des fichiers `main.o` et `fichier1.o`.
- Ces deux fichiers `.o` dépendent de leurs fichiers source respectifs. De plus `main.o` dépend de `fichier1.h`.
- La règle `clean` permet d'effacer les fichiers `.o`. La règle `vclean` applique la règle `clean` puis efface l'exécutable `monprog`.

```
monprog: main.o fichier1.o
        gcc -o monprog main.o fichier1.o

fichier1.o: fichier1.c
        gcc -c fichier1.c

main.o: main.c fichier1.h
        gcc -c main.c

clean:
        rm -f *.o

vclean: clean
        rm -f monprog
```

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- **Outils de débogage**

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

→ Des erreurs syntaxiques ou conceptuels peuvent faire échouer la phase de compilation et/ou d'édition des liens. Par ailleurs, une fois l'exécutable généré, son comportement peut ne pas être celui attendu : résultats faux, comportement non conforme, erreurs de segmentation...

→ Nécessite le recours à des outils de débogage :

- Le plus simple : la fonction `int printf(const char *format, ...)` qui permet notamment d'afficher à l'écran des chaînes de caractères.
- La fonction `void assert(scalar expression)` qui termine le programme en cas d'échec (0) du test.
- Le débogueur `gdb` (et éventuellement son interface graphique `ddd`) qui permet l'exécution d'un programme pas à pas tout en visualisant l'évolution des valeurs des variables.

```
» gdb ./my_exec  
(gdb) run arguments_du_main
```

...

- L'outil `valgrind` qui permet notamment de vérifier que la mémoire allouée dynamiquement est bien libérée par le programme (il existe un grand nombre d'autres possibilités offertes par `valgrind`).

```
» valgrind ./my_exec run arguments_du_main ...
```

- Les outils `gdb` et `valgrind` nécessitent la compilation et l'édition de liens avec l'option `-g`.

# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

## → A propos d'allocation mémoire...

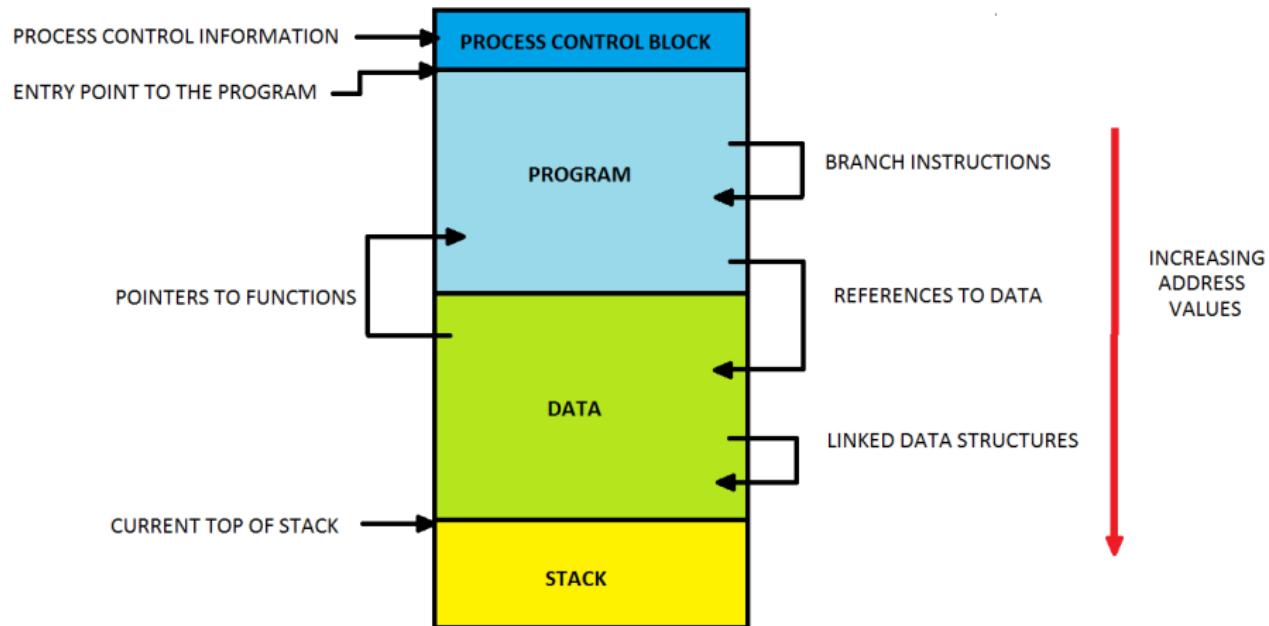
### Différentes méthodes d'allocation

- De manière **statique**, au lancement du programme ; C'est le cas des variables globales ;
- De manière **automatique**, dans la pile (*stack*) d'exécution du processus correspondant au programme en cours ; C'est le cas des variables locales aux fonctions dont la taille est connue et spécifiée au moment de l'écriture du programme ; C'est aussi le cas de la mémoire allouée pour le passage des arguments à une fonction.
- De manière **dynamique**, dans le tas (*heap*) autrement dit dans une zone mémoire du système non spécifiquement dédiée au programme en cours d'exécution.

### Quelques remarques

- L'allocation dynamique permet une grande souplesse de programmation car elle n'impose pas l'allocation de morceaux arbitrairement grands de mémoire au lancement du programme ;
- Elle reste cependant relativement coûteuse en temps d'exécution, peut parfois échouer (difficile à rattraper) et est la source d'un grand nombre de bugs.
- Dans la mesure du possible, il est préférable d'éviter les instructions d'allocation dynamique dans des zones critiques du programme ;
- On évitera par exemple d'allouer dynamiquement de la mémoire dans la partie du code qui réalise une boucle d'asservissement à une fréquence élevée.

## → Un processus en mémoire



# Rappel du plan

## 1 Introduction

- Robotique et Programmation Système
- Synthèse matérielle et logicielle du contrôleur en Robotique

## 2 Le système d'exploitation

- Définition
- Les différents types d'OS
- Les particularités de Linux
- Couches d'abstraction logicielles

## 3 Mécanismes gérés par l'OS

- Vue générale
- Le processeur
- Les interruptions
- Les Entrées/Sorties (IO)

## 4 Technique de production de programmes

- Rappels de C
- La chaîne de compilation
- Make et Makefile
- Outils de débogage

## 5 Mémoire et pointeurs en C

- Allocation mémoire
- Les pointeurs

## pointeurs sont des adresses mémoires

symbole	exemple	signification
&	&x	l'adresse de x
*	*p	l'objet pointé par p



```
int* a,b;      déclare des pointeurs a,b vers des entiers
int c;         déclare un entier c
a = &c;          a est l'adresse de c
*b = *a;        l'objet pointé par b est le même que celui pointé par a
```

→ Un pointeur est une variable contenant une adresse mémoire.



- Notation : `type *p;` -> "p est un pointeur sur type"  
Exemples : `char *p, int *p, POINT *p, struct data *p, etc.`
- Accès à la donnée pointée par p : `*p;`
- Accès au champ d'une structure pointée par p :  
`p->champ ou (*p).champ;`
- Accès à l'adresse de la variable var : `&var;`
- `void *p` : pointeur sur un type indéfini (cf. `malloc()`);
- `p = &data;` stocke dans p l'adresse de data ;
- `*p = data;` copie dans la variable désignée par p la valeur de la variable data ;
- Arithmétique sur les pointeurs (sauf `void *`) :
  - Addition/soustraction d'un entier : déplace le pointeur en mémoire de la taille du désigné ;  
Exemples :
 

```

int *p; /* p est un pointeur sur entiers */
p = &i; /* p reçoit l'adresse de i */
p++; /* p pointe sur l'entier suivant i en mémoire */
          
```
  - Soustraction de pointeurs vers même type : retourne le nombre d'éléments entre les deux adresses ;
  - ! Toute autre opération est interdite car dénuée de sens !

## → Allocation dynamique (dans le tas = heap) en C (#include <stdlib.h>)

- Allocation :

- void \*malloc(size\_t taille);
- void \*calloc(size\_t nmemb, size\_t taille);
- void \*realloc(void \*ptr, size\_t size);

- Désallocation :

- void free(void \*ptr)

```
typedef struct point {  
    double x, y;  
} POINT;  
POINT *p;  
...  
p = (POINT *)malloc(sizeof(POINT));  
p->x = 0.0;  
p->y = 0.0;  
...  
free(p);
```

## ↪ Equivalence tableau / pointeur

- La notation [ ] peut à la fois être utilisée pour un tableau ou pour une zone désignée par un pointeur.

```
int* p;  
p = (int*) malloc(sizeof(int)*100);
```

**p[i] est équivalent à \*(p+i) et p est équivalent à &p[0]**

- Inversement, un tableau peut être manipulé comme un pointeur (sur la première case du tableau).

```
int tab[100];
```

**\*(tab+i) est équivalent à tab[i] et tab est équivalent à &tab[0]**

- Attention :  $\text{sizeof}(p) \neq \text{sizeof}(\text{tab})$ . p a la taille d'une adresse en mémoire : 4 octets pour un processeur 32 bits, 8 octets pour un processeur 64 bits. tab a la taille de 100 entiers : 400 octets.

## → Pointeurs sur des pointeurs

- Il est souvent utile de définir des pointeurs sur des pointeurs sur de type.
- Exemple : Stocker une image de  $10 \times 10$  pixels codés en niveau de gris par des entiers.

```
int** p;
/* Allocation */
p = (int **) malloc(sizeof(int*) * 10);
for (i = 0; i < 10; i++) {
    *(p+i) = (int *) malloc(10 * sizeof(int));
}
...
/* Désallocation */
for (i = 0; i < 10; i++) {
    free(*(p+i));
}
free(p);
```

## TEST 4

Soit :

```
int n = 4;  
double p = 3.14;
```

Comment obtenir l'adresse des variables *n, p* ?

Comment obtenir 7.14 en utilisant les pointeurs *pn, pp* ?

## TEST 5

Soit :

```
int a = 5, b = 7;
```

on veut écrire une fonction pour permuter les entiers de sorte qu'en appelant :

```
swap(&a, &b);
```

on obtienne  $a = 7$ ,  $b = 5$ .

Quel est le prototype de la fonction swap ?

Comment faire la permutation ?

## TEST 6

Soit :

```
int a [] = { 11, 24, 37 };  
int *p;  
p=a;
```

Quels sont les résultats de ces opérations ?

```
(p+2) - p =  
*p =  
*(p+1) =  
*(p+2) - *p =
```

Réponse :

## TEST 7

Soient les chaînes de caractères suivantes de longueurs différentes :

```
char* str1 = "hello";
char* str2 = "death star";
char* str3 = "obi wan";
```

déclarer un tableau de chaînes de caractères qui les contient :

## TEST 8

```
#include <stdio.h>
char* getMessage()
{
    char msg[] = "Pointers are fun";
    return msg;
}
int main ()
{
    char* string = getMessage();
    puts(string);
    return 0;
}
```

Qu'est ce qui est faux dans ce code ?

## TEST 9 (pas facile)

```
#include<stdio.h>
int main()
{
    static int i,j,k;
    int* (*ptr) [];
    int* array[3]={&i,&j,&k};
    ptr=&array;
    j=i+++k+10;
    ++(**ptr);
    printf("Output = %d",***ptr);
    return 0;
}
```

Que retourne ce code ?

## TEST 9 (pas facile)

```
static int i,j,k;  
int *(*ptr)[];  
int *array[3]={&i,&j,&k};  
ptr=&array;
```

## TEST 9 (pas facile)

```
j=i+++k+10;
```

```
++(**ptr);
```

```
printf("Output = %d",***ptr);
```

## Questions ?

