Systèmes d'exploitations

Cours 1: Introduction

Matthieu Lemerre
CEA LIST

Année 2021-2022

Sommaire

Avant propos

Je suis toujours intéressé par des étudiants qui cherchent des stages de recherche autour de certains des sujets suivants:

- Analyse de programme, sémantique des langages de programmation, compilation;
- Méthodes formelles, vérification, interprétation abstraite;
- Cybersécurité, reverse-engineering, analyse de binaire, de malware;
- Systèmes d'exploitation, temps-réel, systèmes embarqués, concurrence et parallélisme.

But du cours de système d'exploitation

- Savoir comment lancer firefox sous windows

But du cours de système d'exploitation

- ~~Savoir comment lancer firefox sous windows~~

But du cours de système d'exploitation

- ~~Savoir comment lancer firefox sous windows~~
- Connaitre les différences entre macOS 'High Sierra' et 'Mojave'
  - Amélioration des captures d'écran
  - Le fond d'écran correspond à l'heure de la journée

But du cours de système d'exploitation

- ~~Savoir comment lancer firefox sous windows~~
- ~~Connaitre les différences entre macOS 'High Sierra' et 'Mojave'~~
  - ~~Amélioration des captures d'écran~~
  - ~~Le fond d'écran correspond à l'heure de la journée~~

But du cours de système d'exploitation

- ~~Savoir comment lancer firefox sous windows~~
- ~~Connaitre les différences entre macOS 'High Sierra' et 'Mojave'~~
  - ~~Amélioration des captures d'écran~~
  - ~~Le fond d'écran correspond à l'heure de la journée~~

- Répondre à la question: quels sont les chaînons entre un programme écrit en C, et son exécution sur le processeur?
  - Quel est le fonctionnement général du processeur et du compilateur?
  - Quel est le rôle du système d'exploitation (OS)?
  - Quels sont les principaux concepts et services fournis par un OS? (au sens large, incluant les hyperviseur)
  - Comment fonctionnent ces services?
  - Comment l'OS est architecturé pour fournir ces services?

Application de ce cours

- Améliorer votre compréhension:
  - Du language C, de l'assembleur et des langages bas-niveau;
  - De la performance des programmes
  - Du *fonctionnement* des mécanismes de programmation concurrente (threads,processus,passage de message)
  - Des concepts et fonctionalités fournies par les systèmes d'exploitation (thread,processus, contrôle d'accès, mémoire virtuelle, politique d'ordonnancement, machine virtuelle. . .)

Application de ce cours

- Améliorer votre compréhension:
  - Du language C, de l'assembleur et des langages bas-niveau;
  - De la performance des programmes
  - Du *fonctionnement* des mécanismes de programmation concurrente (threads,processus,passage de message)
  - Des concepts et fonctionalités fournies par les systèmes d'exploitation (thread,processus, contrôle d'accès, mémoire virtuelle, politique d'ordonnancement, machine virtuelle...)
- ➜ Connaissances nécessaires pour tout projet
  - Avec du C
    - comprendre l'environnement autour de votre programme C
  - Orienté performance
    - comprendre le coût des opérations que fait votre programme
  - Système
    - De la machine nue au système d'exploitation
    - Systèmes embarqués (robotique, réseau de capteurs, contrôle-commande)
  - Sécurisé
    - Notions de cloud, protection mémoire, buffer overflow...

Organisation du cours

- Cours 1 : vue d'ensemble du système d'exploitation
- Cours et TPs 2 à 6 : construction "bottom-up" d'un système d'exploitation/hyperviseur:
- Cours 7 : Examen écrit.

Les TPs représentent différents morceaux d'un système d'exploitation. Mis ensembles, ils permettent de réaliser un système d'exploitation en entier.
Certains TPs seront notés.

Sommaire

Quels OS connaissez-vous?

Quels OS connaissez-vous?

Quels OS connaissez-vous?

Quels OS connaissez-vous?

Windows Linux MacOS

Quels OS connaissez-vous?

Quels OS connaissez-vous?

Windows Linux MacOS UNIX *BSD

Quels OS connaissez-vous?

Quels OS connaissez-vous?

Windows Linux MacOS UNIX *BSD Android IOS Windows Phone

Quels OS connaissez-vous?

Quels OS connaissez-vous?

Windows Linux MacOS UNIX *BSD Android IOS Windows Phone AIX VxWorks L4 Asterios Anaxagoros Minix Spring Solaris OS/2 DOS Hurd Singularity AmigaOS GCOS Contiki MULTICS Babix ...

Idée reçue: OS = "windows/linux/macOS sur mon PC" (1)

- Parts de marché des ordinateurs de bureau:

	Date	Windows	macOS	Autre	Linux	Chrome O
Statcounter (page views)	09/2018	81.76	13.49	1.99	1.68	1.0

Idée reçue: OS = "windows/linux/macOS sur mon PC" (1)

- Parts de marché des ordinateurs de bureau:

	Date	Windows	macOS	Autre	Linux	Chrome O
Statcounter (page views)	09/2018	81.76	13.49	1.99	1.68	1.0

- Parts de marché des smartphones:

	Date	Android	iOS	Windows	Autres
Statcounter (page views)	08/2018	77.3	19.9	0.41	2.45

Idée reçue: OS = "windows/linux/macOS sur mon PC" (1)

- Parts de marché des ordinateurs de bureau:

	Date	Windows	macOS	Autre	Linux	Chrome O
Statcounter (page views)	09/2018	81.76	13.49	1.99	1.68	1.0

- Parts de marché des smartphones:

	Date	Android	iOS	Windows	Autres
Statcounter (page views)	08/2018	77.3	19.9	0.41	2.45

- Parts de marché des serveurs internets:

	Date	Linux	FreeBSD	Autre UNIX	Windows
W3techs (top 10^7)	02/2015	35.9	0.95	30.9	32.3
W3Cook (top 10^6)	05/2015	96.6	1.7	0	1.7

Idée reçue: OS = "windows/linux/macOS sur mon PC" (1)

- Parts de marché des ordinateurs de bureau:

	Date	Windows	macOS	Autre	Linux	Chrome O
Statcounter (page views)	09/2018	81.76	13.49	1.99	1.68	1.0

- Parts de marché des smartphones:

	Date	Android	iOS	Windows	Autres
Statcounter (page views)	08/2018	77.3	19.9	0.41	2.45

- Parts de marché des serveurs internets:

	Date	Linux	FreeBSD	Autre UNIX	Windows
W3techs (top 10^7)	02/2015	35.9	0.95	30.9	32.3
W3Cook (top 10^6)	05/2015	96.6	1.7	0	1.7

- Parts de marché des super-ordinateurs:

	Date	Linux
TOP 500	2018	100%

Idée reçue: OS = "windows/linux/macOS sur mon PC" (2)

- Beaucoup d'autres marchés!
  - Systèmes embarqués (avions, automobiles, robots, lave-vaisselles, objets connectés . . .)
  - Systèmes temps-réel (dur, mou, . . .)
  - Systèmes à haute sécurité (civil, militaire, ..)
  - Systèmes à haut degré de sûreté de fonctionnement (avion, nucléaire,. . .)
  - Mainframes (systèmes bancaires à haute disponibilité, . . .)
  - Hyperviseurs (pour cloud, pour la sécurité)
  - Systèmes de télécommunication (antennes, coeur de réseau. . .)
  - . . .

A quoi sert un OS?

À quoi sert un OS?

A quoi sert un OS?

À quoi sert un OS?

- Facilite l'usage de la machine
  - En particulier: facilite l'écriture de programmes

➜ Gestion du matériel

➜ Couches d'abstraction (librairies)

➜ Services

A quoi sert un OS?

À quoi sert un OS?

- Facilite l'usage de la machine
  - En particulier: facilite l'écriture de programmes
  - ➜ Gestion du matériel
  - ➜ Couches d'abstraction (librairies)
  - ➜ Services
- Permet à plusieurs tâches de coexister sur une même machine
  - ➜ Concurrence (synchronisation, communication)
  - ➜ Sécurité
  - ➜ Partage des ressources (CPU, mémoire vive, disque dur, réseau...)

A quoi sert un OS?

À quoi sert un OS?

- Facilite l'usage de la machine
  - En particulier: facilite l'écriture de programmes
  - ➜ Gestion du matériel
  - ➜ Couches d'abstraction (librairies)
  - ➜ Services partagés
- Permet à plusieurs tâches de coexister sur une même machine
  - ➜ Concurrence (synchronisation, communication)
  - ➜ Sécurité
  - ➜ Partage des ressources (CPU, mémoire vive, disque dur, réseau...)

Histoire

- Les premiers ordinateurs n'exécutaient qu'un seul programme à la fois (batch processing)

Histoire

- Les premiers ordinateurs n'exécutaient qu'un seul programme à la fois (batch processing)
- Ce programme accédait à
  - tout le processeur
  - toute la mémoire
  - tous les périphériques (lecture et écriture de cartes perforées)

➜ Pas de partage de ressources

➜ Pas besoin de système d'exploitation

Histoire

- Les premiers ordinateurs n'exécutaient qu'un seul programme à la fois (batch processing)
- Ce programme accédait à
  - tout le processeur
  - toute la mémoire
  - tous les périphériques (lecture et écriture de cartes perforées)

➜ Pas de partage de ressources

➜ Pas besoin de système d'exploitation

- Compatible time-sharing system (1961):
  - Exécution simultanée de plusieurs programmes

  ➜ Premiers systèmes d'exploitations

Histoire

- Les premiers ordinateurs n'exécutaient qu'un seul programme à la fois (batch processing)
- Ce programme accédait à
  - tout le processeur
  - toute la mémoire
  - tous les périphériques (lecture et écriture de cartes perforées)

➜ Pas de partage de ressources

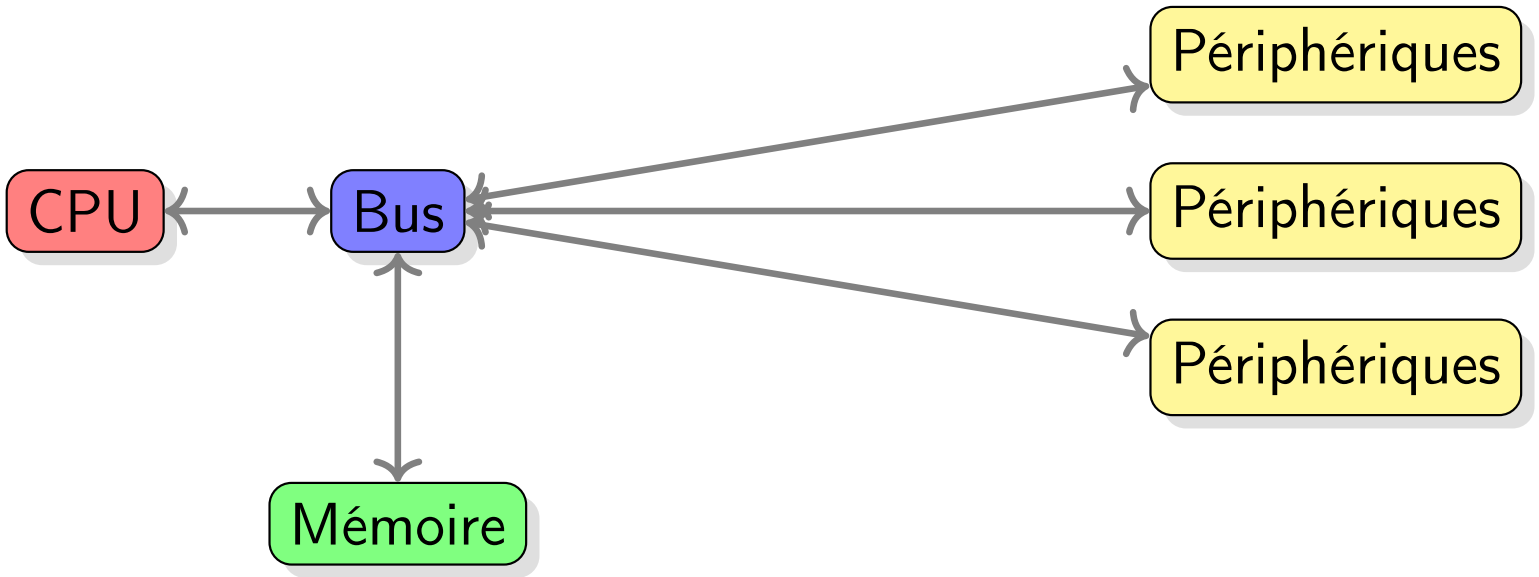
➜ Pas besoin de système d'exploitation

- Compatible time-sharing system (1961):
  - Exécution simultanée de plusieurs programmes
  - ➜ Premiers systèmes d'exploitations
- Ecore aujourd'hui: les micro-controleurs n'ont souvent pas de système d'exploitation

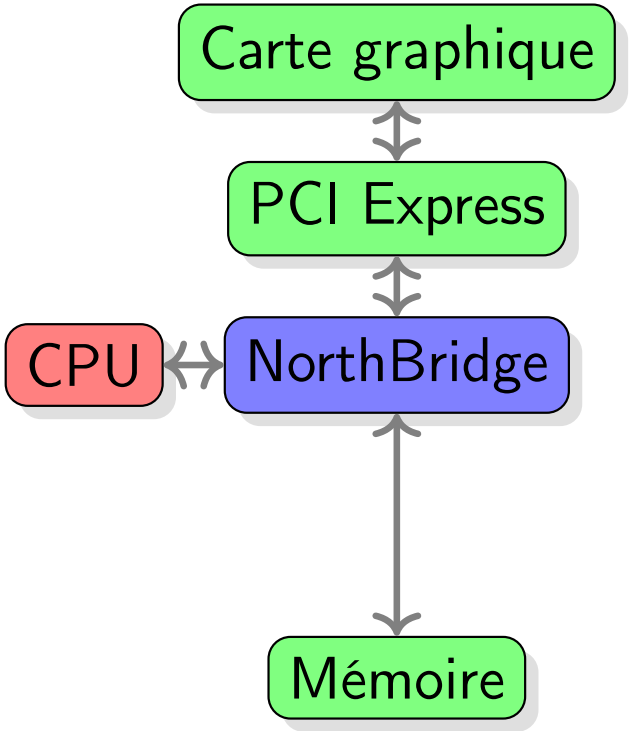
Un système d'exploitation n'est utile que lorsqu'il y a plusieurs programmes.

Sommaire

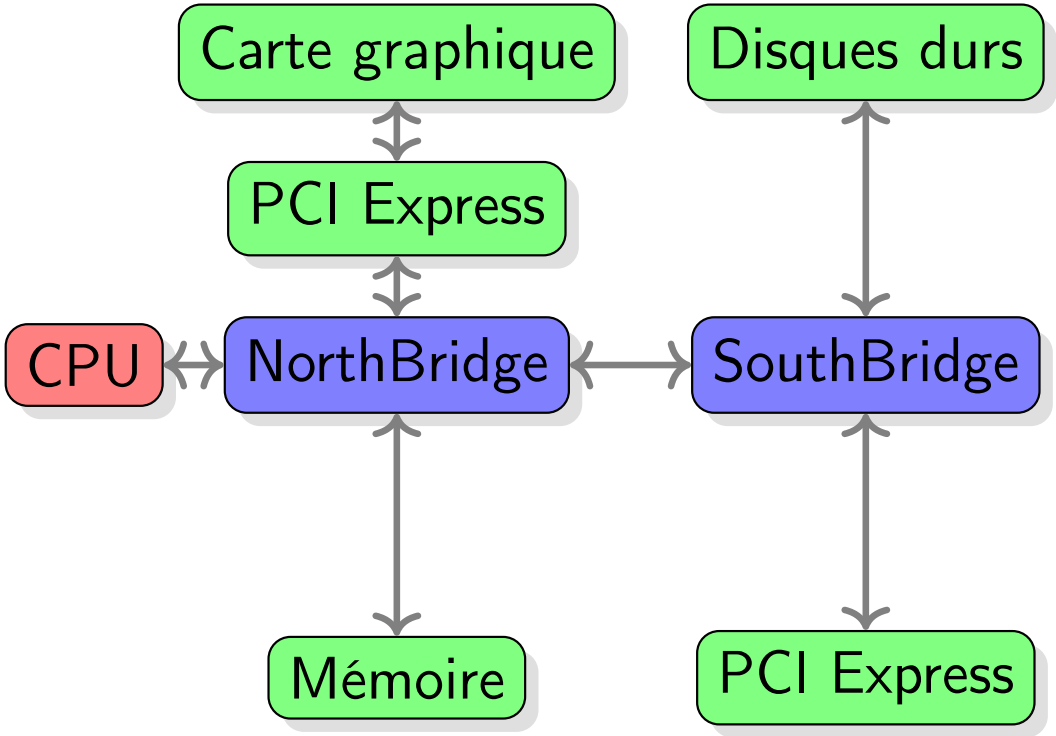
Composants d'un ordinateur



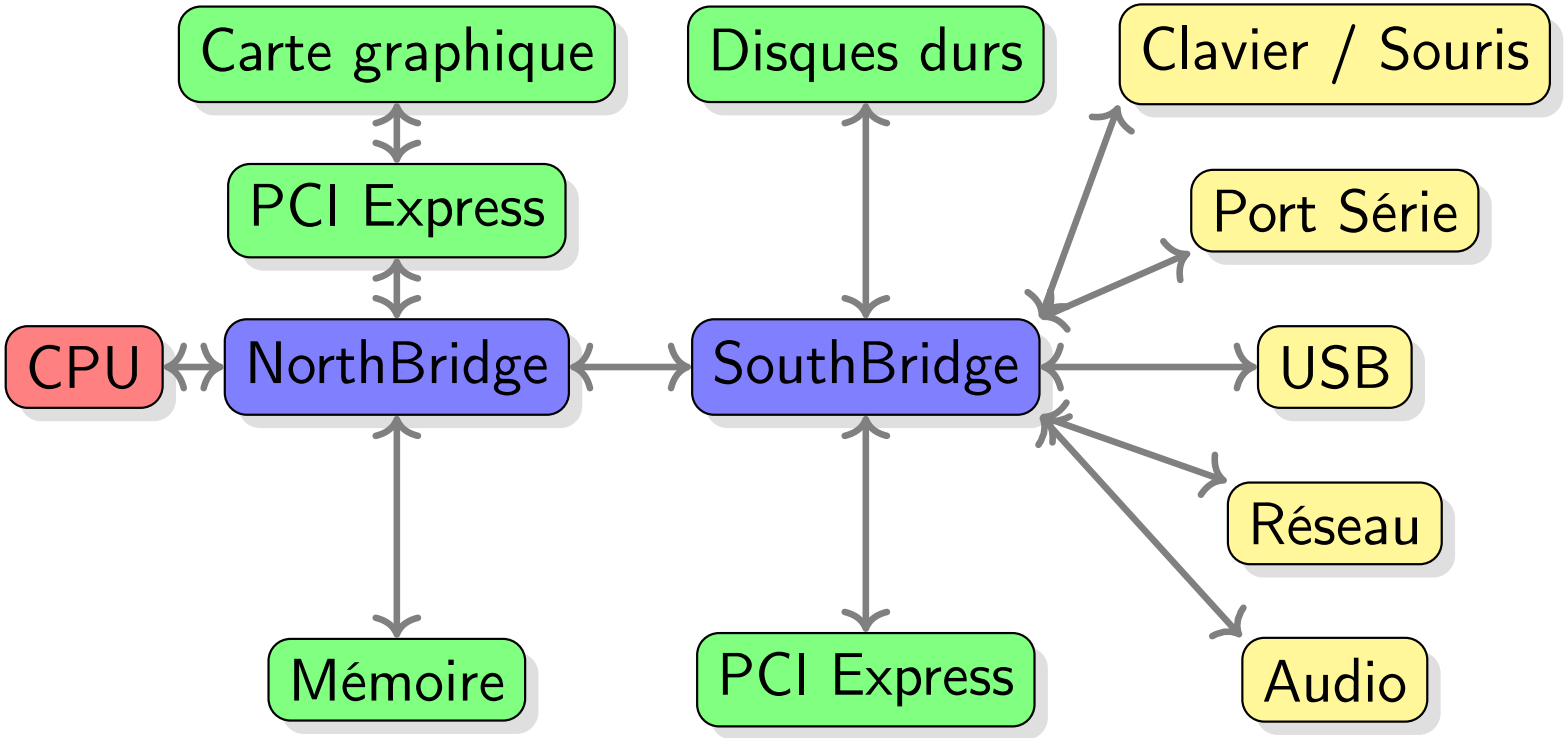
Composants dans un PC moderne



Composants dans un PC moderne

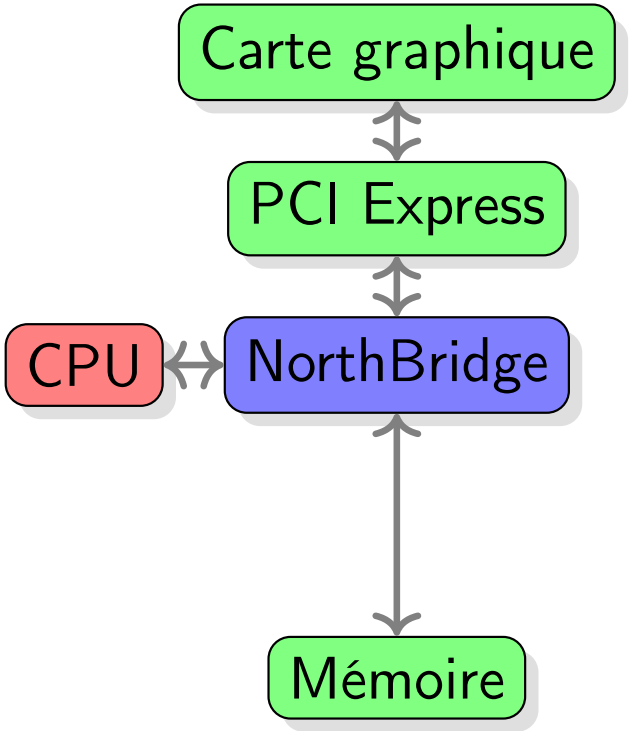


Composants dans un PC moderne



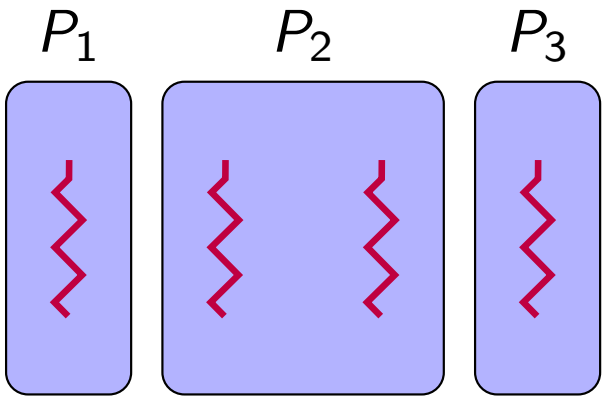
- Comment faciliter l'usage de toutes ces *ressources* matérielles?
- Comment organiser leur partage entre les différents programmes?

Composants dans un PC moderne



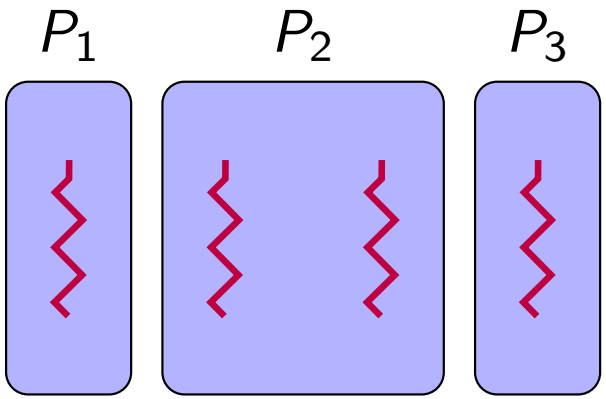
Le fonctionnement de l'architecture matérielle, et de la manière dont les programmes sont exécutés, sera vue en détail (cours 2).

Organisation et partage des accès CPU et mémoire



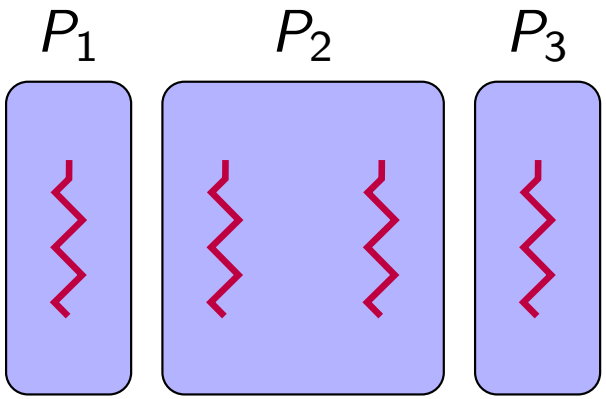
- On utilise un OS quand on veut exécuter différents programmes, ou différentes instances d'un programme.
- On appelle *processus* une instance d'un programme.
- Un processus peut être vu comme une *machine virtuelle*, disposant de:
  - Son CPU virtuel (thread)
  - Sa mémoire virtuelle (espace d'adressage)
  - Des périphériques virtuels, permettant d'accéder aux périphériques réels

Organisation et partage des accès CPU et mémoire



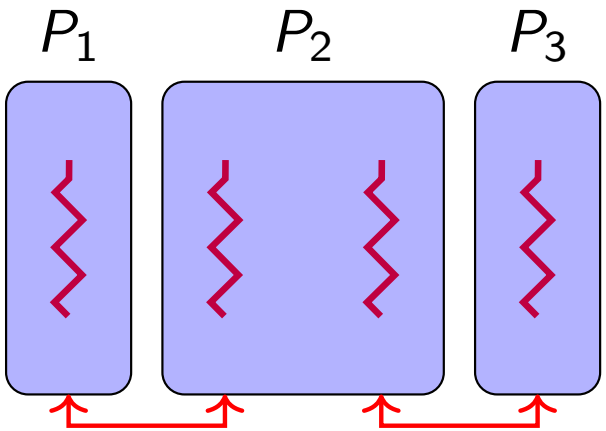
- Les threads et la mémoire virtuelle permettent d'*organiser* et *partager* l'accès au CPU et à la mémoire réelle
  - ➜ Ils sont étudiés en détail dans les cours 3 (thread), 4 (ordonnancement), et 5 (mémoire virtuelle)

Organisation et partage des accès CPU et mémoire



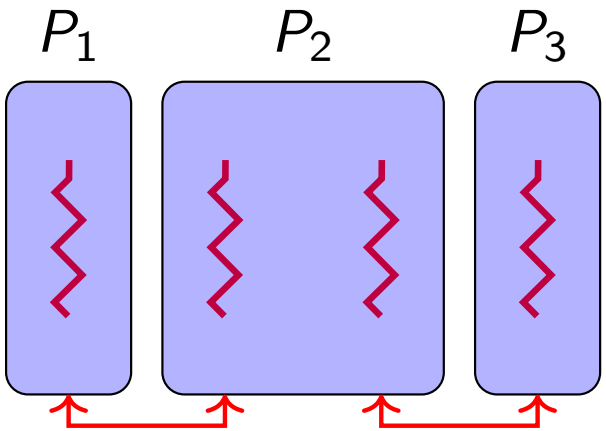
- Dans des machines réelles, les programmes sont isolés les uns des autres
  - Une erreur dans un programme n'empêche pas les autres de fonctionner
  - Un programme ne peut pas accéder aux ressources des autres machines
- De manière similaire, on va confiner les processus
  - ➜ Sécurité des systèmes d'exploitation (cours 5 et 6)

Organisation et partage des accès CPU et mémoire



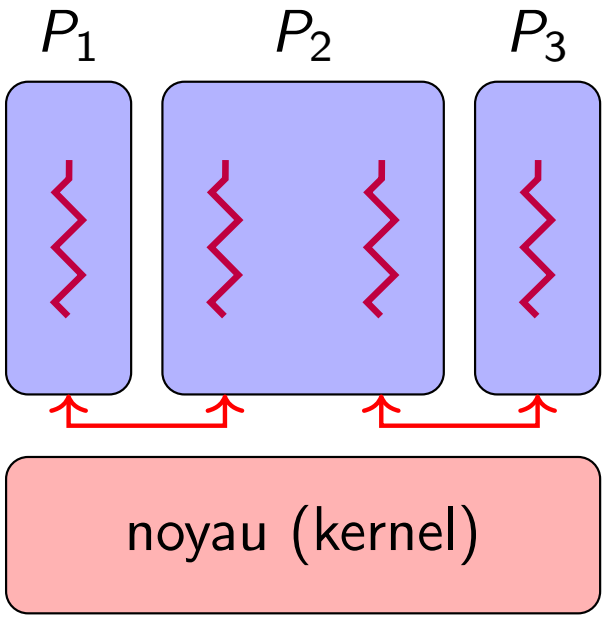
- Les machines réelles peuvent communiquer les unes avec les autres, par l'intermédiaire de liens réseau.
- De manière similaire, on va permettre aux processus de communiquer (par l'intermédiaire de mémoire partagée)
  - C'est la communication inter-processus (cours 3)

Organisation et partage des accès CPU et mémoire



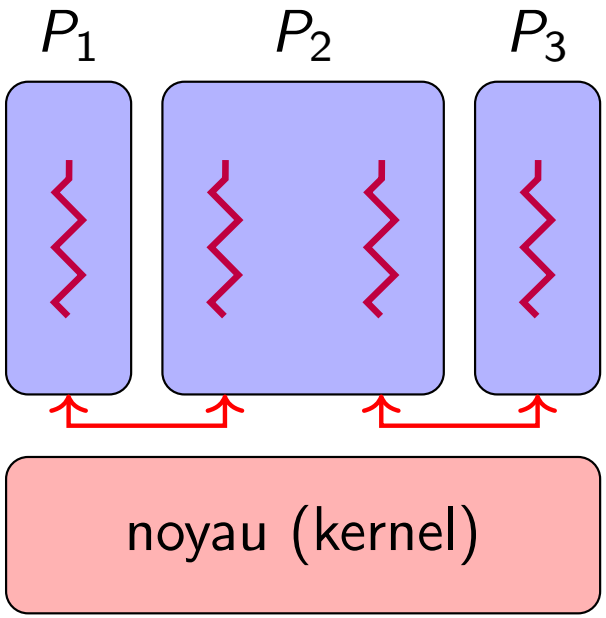
- On peut donner aux processus une interface de haut-niveau
  - Facilite l'utilisation de l'ordinateur
  - Rend le programme indépendant de la machine
- Une autre manière de faire consiste à vraiment faire croire au programme qu'il s'exécute sur une machine nue
  - ➜ *Virtualisation* (cours 6)

Organisation et partage des accès CPU et mémoire



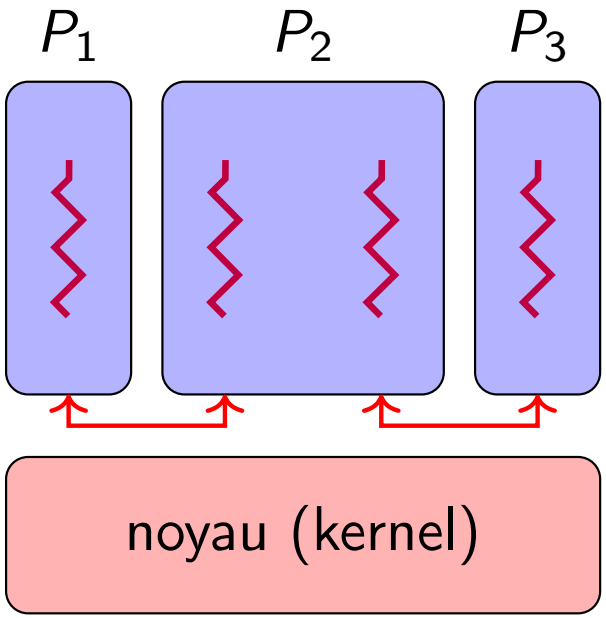
- Pour choisir:
  - Comment est répartie la mémoire (*allocation*)
  - Quel processus s'exécute à un instant t (*ordonnancement*)
  - Il faut un programme, appellé *noyau*
    - Dans le cas de machines virtuelles, on l'appelle l'*hyperviseur*
- Pour garantir l'isolation des processus, un processus normal ne doit pas pouvoir modifier l'ordonnancement ni l'allocation.
  - ➜ Le noyau est *privilégié*
  - ➜ C'est la clé de voûte de la sécurité du système.

Organisation et partage des accès CPU et mémoire



- Les *services* sont des programmes qui organisent le partage des autres ressources de la machine
  - La partie qui communique avec le matériel s'appelle un *pilote de périphérique* (driver).
  - Les services peuvent être privilégiés ou non.
  - Nous allons voir des exemples de services juste après.

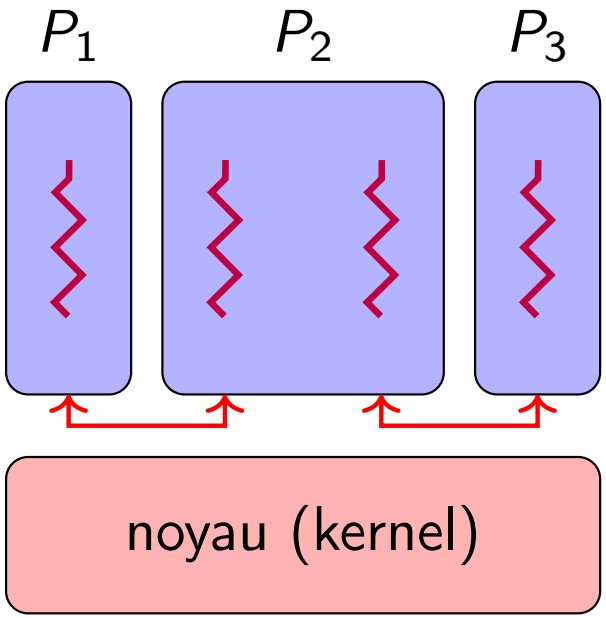
Organisation et partage des accès CPU et mémoire



Pour remplir son rôle (accès facilité et partage des ressources), le système d'exploitation est un ensemble composé *au minimum*:

- Du noyau (ex: noyau Linux)
- De services (exemple: serveur d'affichage, de son, pile réseau)
- De librairies systèmes (exemple: librairie standard C)

Organisation et partage des accès CPU et mémoire

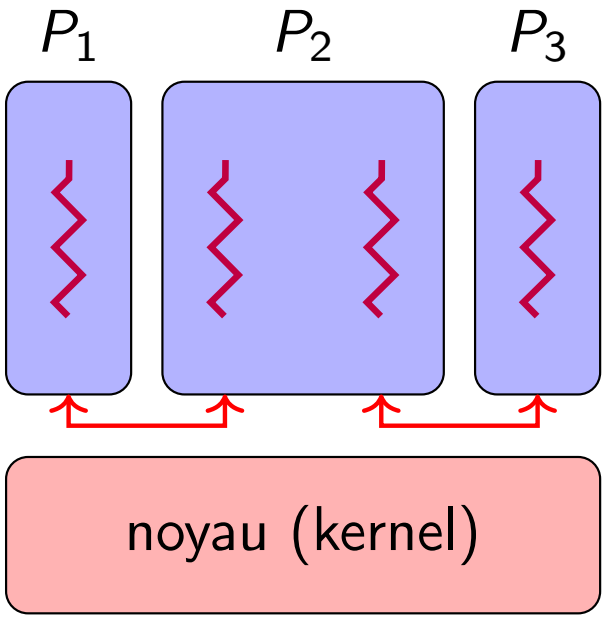


Pour remplir son rôle (accès facilité et partage des ressources), le système d'exploitation est un ensemble composé *au minimum*:

- Du noyau (ex: noyau Linux)
- De services (exemple: serveur d'affichage, de son, pile réseau)
- De librairies systèmes (exemple: librairie standard C)

Pour l'utiliser (développer et exécuter les programmes qui fonctionneront dessus), un kit de développement (software developpement kit), comprenant compilateur, documentation... est généralement nécessaire.

Organisation et partage des accès CPU et mémoire



Pour remplir son rôle (accès facilité et partage des ressources), le système d'exploitation est un ensemble composé *au minimum*:

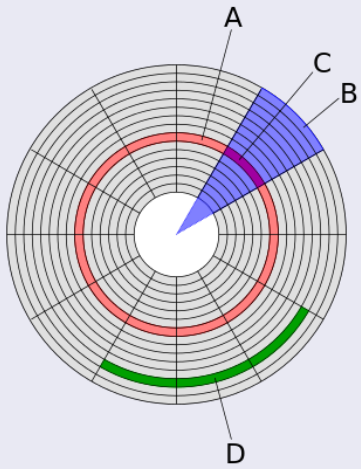
- Du noyau (ex: noyau Linux)
- De services (exemple: serveur d'affichage, de son, pile réseau)
- De librairies systèmes (exemple: librairie standard C)

Pour l'utiliser (développer et exécuter les programmes qui fonctionneront dessus), un kit de développement (software developpement kit), comprenant compilateur, documentation... est généralement nécessaire.

Des programmes additionnels pour faciliter son usage sont possibles (shell, librairie de compression, gestionnaire de fonds d'écran...)

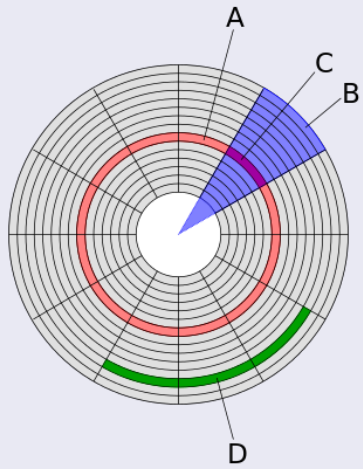
Organisation et partage du disque dur

- Un disque dur est un cylindre qui stocke des secteurs de taille 512 octets, adressés par un numéro de plateau (hauteur), de piste (A; distance), et de secteur géométrique (B; angle).
- Comment partager ces secteurs entre les différents processus?



Organisation et partage du disque dur

- Un disque dur est un cylindre qui stocke des secteurs de taille 512 octets, adressés par un numéro de plateau (hauteur), de piste (A; distance), et de secteur géométrique (B; angle).
- Comment partager ces secteurs entre les différents processus?

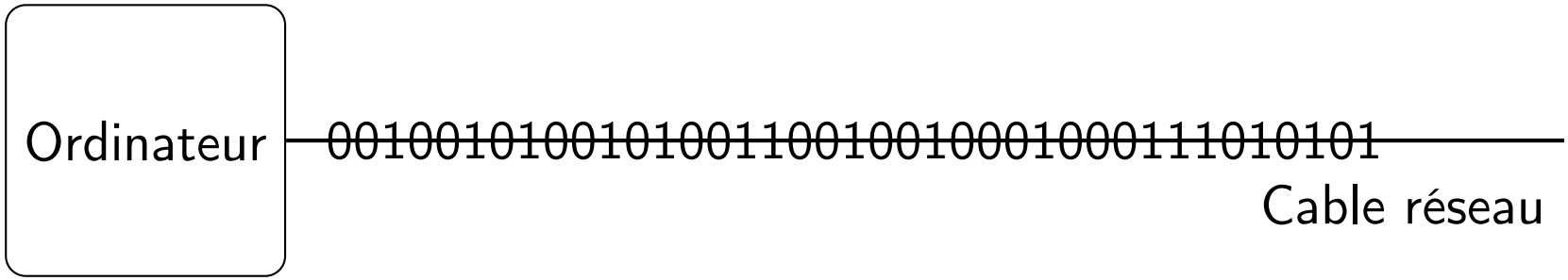


- Fichier: représente un ensemble de secteurs; adressé continuement
- Répertoires: organise les fichiers en une arborescence
  - '..' permet de remonter dans l'arbre
- Permissions: détermine quel processus peut accéder à un fichier

```
matthieu@is230028:/usr/lib/ocaml$ tree -d /dev
/dev
├── block
├── bsg
├── bus
│   └── usb
│       ├── 001
│       ├── 002
│       └── 003
├── char
├── cpu
├── disk
```

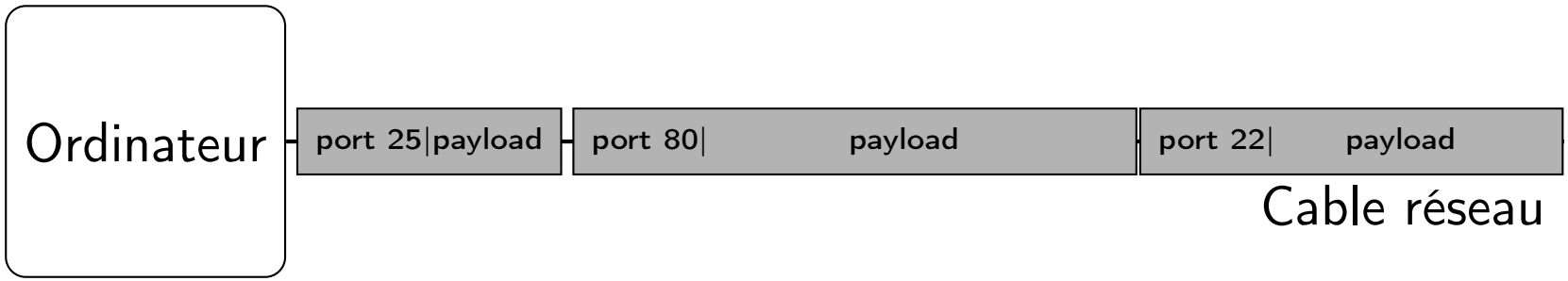
Organisation et partage du réseau: le protocole TCP/IP

Un ordinateur est connecté à un réseau par un cable. Une différence de potentiel sur une paire torsadée code pour 1, et l'absence de différence code pour 0. À quel processus transmettre les bits d'information reçus?



Organisation et partage du réseau: le protocole TCP/IP

Un ordinateur est connecté à un réseau par un cable. Une différence de potentiel sur une paire torsadée code pour 1, et l'absence de différence code pour 0. À quel processus transmettre les bits d'information reçus?



Solution: le protocole réseau (exemple: TCP/IP)

- Regroupement des bits d'information en paquets
- Chaque paquet contient un numéro de port
- Le système d'exploitation attribue chaque numéro de port ouvert à un processus.
- Attribution des ports sous UNIX:
  - ports < 1024 sont réservés au super-utilisateur
  - sinon premier arrivé permier servi

Organisation et partage des périphériques intéractifs

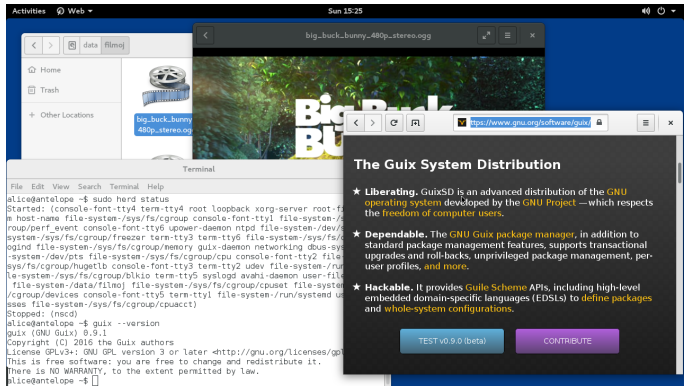
- Quel processus doit recevoir les entrées au clavier? Les clics de souris?
- Quel processus a le droit d'accéder à l'écran?

Organisation et partage des périphériques intéractifs

- Quel processus doit recevoir les entrées au clavier? Les clics de souris?
- Quel processus a le droit d'accéder à l'écran?

Système de fenêtres (graphique)

- Seule la fenêtre selectionnée doit recevoir les entrées clavier et souris
- L'utilisateur choisi comment partager son écran entre les applications

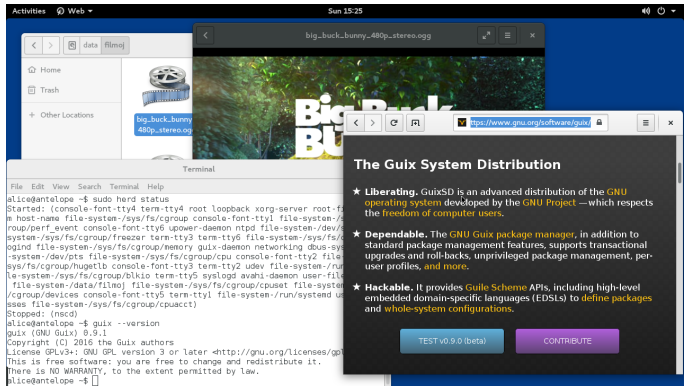


Organisation et partage des périphériques intéractifs

- Quel processus doit recevoir les entrées au clavier? Les clics de souris?
- Quel processus a le droit d'accéder à l'écran?

Système de fenêtres (graphique)

- Seule la fenêtre selectionnée doit recevoir les entrées clavier et souris
- L'utilisateur choisi comment partager son écran entre les applications



Le shell (programmes textuels)

- Seul le programme en avant-plan reçoit les entrées clavier
- Le shell affiche un mélange des sorties standard de tous les programmes

Sommaire

Conclusion

Un système d'exploitation est un ensemble de programmes qui permet:

- D'exécuter des programmes différents (processus);
- D'organiser le partage des ressources matérielles entre les processus;
- De faciliter l'usage du matériel en fournissant des abstractions de plus haut niveau;
- De protéger chaque processus de ses voisins.

L'OS rend des services; on l'utilise en lui demandant ces services.

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse 0x1004

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse 0x1004
  - `&(ptr->y)` pointe à l'adresse

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse 0x1004
  - `&(ptr->y)` pointe à l'adresse 0x1006

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse 0x1004
  - `&(ptr->y)` pointe à l'adresse 0x1006
  - `&(ptr->z)` pointe à l'adresse

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse 0x1004
  - `&(ptr->y)` pointe à l'adresse 0x1006
  - `&(ptr->z)` pointe à l'adresse 0x100a

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse 0x1004
  - `&(ptr->y)` pointe à l'adresse 0x1006
  - `&(ptr->z)` pointe à l'adresse 0x100a
  - `sizeof(struct p)` vaut

Notion pour le TP: structure et adresse mémoire

```
struct p {
  uint32_t x;
  uint16_t u;
  uint32_t y;
  uint32_t z; } __attribute__((packed));;
struct p *ptr;
```

- `packed` assure que les champs des structures sont les uns à la suite des autres (pas de *padding*).
- `uint32_t` est un mot de 32 bits
- `uint32_t` est un mot de $32/8 = 4$ octets
- Si `ptr` pointe à l'adresse 0x1000, alors
  - `&(ptr->x)` pointe à l'adresse 0x1000
  - `&(ptr->u)` pointe à l'adresse 0x1004
  - `&(ptr->y)` pointe à l'adresse 0x1006
  - `&(ptr->z)` pointe à l'adresse 0x100a
  - `sizeof(struct p)` vaut $14 =$ 0xd.

Notion pour le TP: endianness

```
char array[4] = { 0x11; 0x22; 0x33; 0x44 };
uint32_t *ptr = &array[0];
uint32_t res = *ptr;
```

Que vaut `res` à la fin?

Notion pour le TP: endianness

```
char array[4] = { 0x11; 0x22; 0x33; 0x44 };
uint32_t *ptr = &array[0];
uint32_t res = *ptr;
```

Que vaut `res` à la fin?

- Machine big endian (Sparc,MIPS,...): 0x11223344;
- Machine little endian: 0x44332211

Notion pour le TP: endianness

```
char array[4] = { 0x11; 0x22; 0x33; 0x44 };
uint32_t *ptr = &array[0];
uint32_t res = *ptr;
```

Que vaut `res` à la fin?

- Machine big endian (Sparc,MIPS,...): 0x11223344;
- Machine little endian: 0x44332211

L'endianness est une convention pour lire ou les octets dans un mot sur plusieurs octets (comme le sens d'écriture).

- Big endian: on commence par l'octet de poids fort ("de gauche à droite")
- Little endian: on commence par l'octet de poids faible ("de droite à gauche")

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 =`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 =`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`
- `0b1111 « 2 =`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`
- `0b1111 « 2 = 0b111100`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`
- `0b1111 « 2 = 0b111100`
- `0b1111 » 2 =`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`
- `0b1111 « 2 = 0b111100`
- `0b1111 » 2 = 0b11`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`
- `0b1111 « 2 = 0b111100`
- `0b1111 » 2 = 0b11`
- `0x2f =`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`
- `0b1111 « 2 = 0b111100`
- `0b1111 » 2 = 0b11`
- `0x2f = 0b0010....`

Notion pour le TP: operateurs bit à bit

- `0b01010101 & 0b11110000 = 0b01010000`
- `0b01010101 | 0b11110000 = 0b11110101`
- `0b1111 « 2 = 0b111100`
- `0b1111 » 2 = 0b11`
- `0x2f = 0b00101111`