SYS2

Système d'exploitation

M.Bastreghi (mba)

Haute École Bruxelles Brabant — École Supérieure d'Informatique

Année académique 2020 / 2021

définitions

Multiprogrammation - Timeslicing

multiprogrammation : plusieurs programmes sont chargés en mémoire et s'exécutent de manière entrelacée

timeslicing : ajoute une contrainte de temps à la Multiprogrammation.

Multiprogrammation - Timeslicing

multiprogrammation : plusieurs programmes sont chargés en mémoire et s'exécutent de manière entrelacée

timeslicing : ajoute une contrainte de temps à la Multiprogrammation.

- La **multiprogrammation** permet de réattribuer le CPU au moment d'une Entrée/Sortie.
- Le **timeslicing** (tranche de temps) définit un temps maximum pendant lequel un processus peut utiliser le CPU sans interruption. Ce délai passé, le CPU est retiré au processus même si il ne fait pas de demande d'Entrée/Sortie.

Processus

Processus définition

processus = "programme en mémoire"

un processus s'exécute en mode normal et en **mode privilégié** lorsqu'on exécute du code système pour son compte

Multiprogrammation

La Multiprogrammation doit son invention dans les années 60 à une volonté de rentabilisation du CPU. .

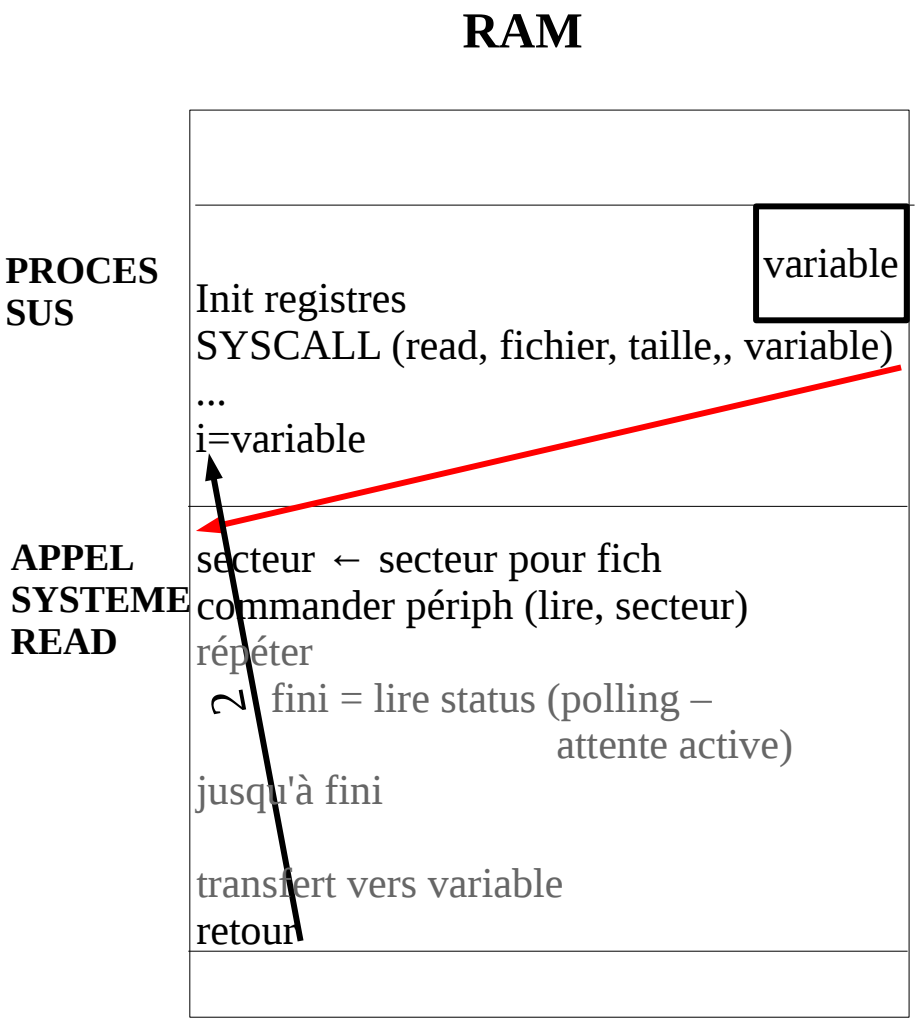
En effet, en monoprogrammation le CPU ne peut être employé à quelque chose d'utile pendant les entrées-sorties gérées par les périphériques lents.

La Multiprogrammation doit son invention dans les années 60 à une volonté de rentabilisation du CPU. .

En effet, en monoprogrammation le CPU ne peut être employé à quelque chose d'utile pendant les entrées-sorties gérées par les périphériques lents.

- Les ordinateurs sont des machines très chères dans les années 60. Leur rentabilité est primordiale.
- Les périphériques sont très lents comparativement aux CPU's.

Monoprogrammation - cas Lecture 1



Monoprogrammation - cas Lecture 1

Code du **programme** :

- *registres* <- numéro d'appel système read, fichier, taille, variable
- SYSCALL : basculement dans le noyau
- *un peu plus tard . . .*
- exploiter les données lues en RAM (i = variable)

Monoprogrammation - cas Lecture 1

Code de l'Appel Système **read** appelé par l'instruction SYSCALL :

- n° secteur <- numéro de secteur correspondant
- commander le périphérique (lire, n° secteur)
- scruter périodiquement le status périphérique : **polling** (**attente active** du CPU) ! !
- **rappatrier** les données lues dans la variable

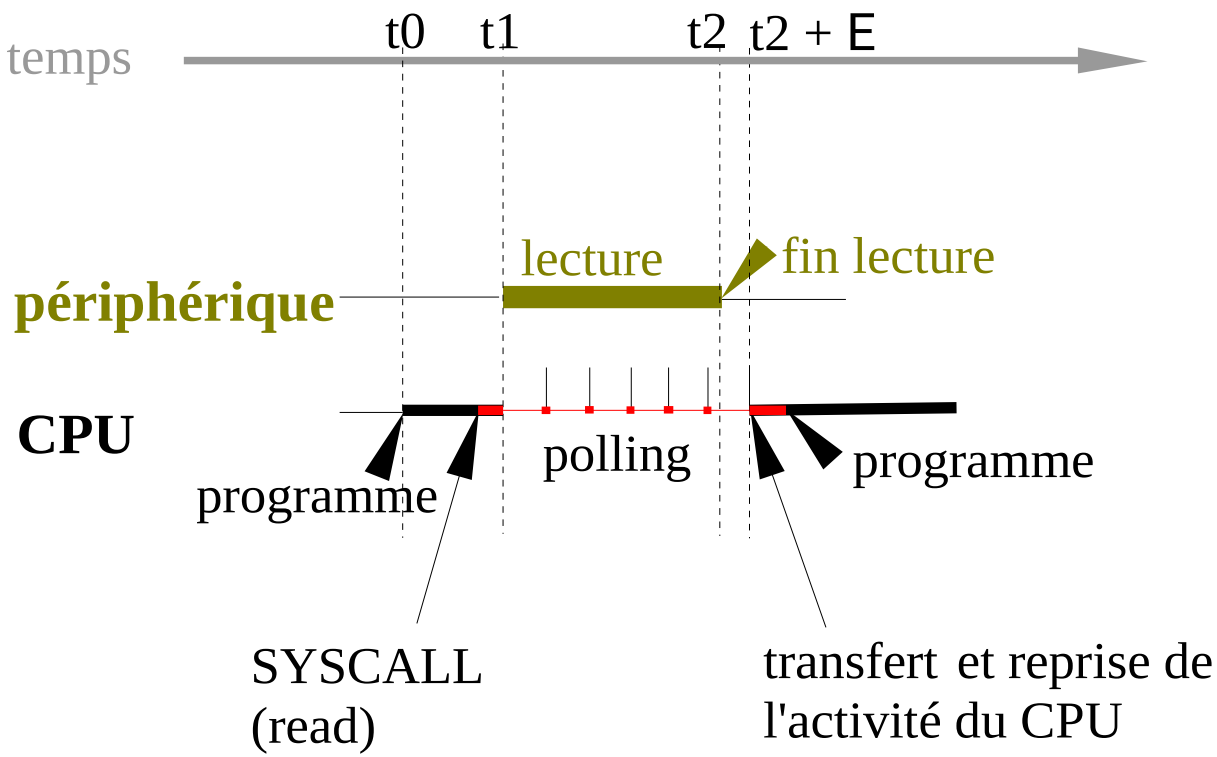
le code utilisateur reprend . . .

Monoprogrammation - cas Lecture 1

Ici, le CPU **attend** que le périphérique ait mis à disposition les données pour les transférer en RAM et reprendre son activité au profit du programme.

Monoprogrammation - cas Lecture 1

Activité CPU-périphérique dans le temps



Monoprogrammation - cas Lecture 1

Constat :
Le CPU passe beaucoup de temps à attendre (attente active) que le périphérique ait fini sa lecture sur disque.

-> utilisation inefficace du CPU

Processeur canal ou DMA

Processeur Canal

idée :
associer au CPU un processeur externe relié par le bus (CANAL) et lui confier la gestion du périphérique et le transfert vers et depuis la RAM

Processeur canal ou DMA

le **processeur canal** (DMA dans le monde PC)

- commande le périphérique
- **accède à la RAM** pour transférer la donnée lue
- génère une interruption pour prévenir de la fin de la lecture

-> le CPU est libéré au profit d'un deuxième processus chargé en RAM

Processeur canal ou DMA

le **processeur canal** (DMA dans le monde PC)

- commande le périphérique
- **accède à la RAM** pour transférer la donnée lue
- génère une interruption pour prévenir de la fin de la lecture

-> le CPU est libéré au profit d'un deuxième processus chargé en RAM

Le processeur Canal est relié au bus et accède à la RAM.

le Canal interfère avec le fonctionnement du CPU

-> une synchronisation CPU/Canal est nécessaire (régler les accès à la RAM via les bus)

Multiprogrammation

le CPU ne gère plus polling et transfert de données

Cela va permettre de mieux le rentabiliser.
C'est le principe de la **Multiprogrammation** :

- plusieurs programmes sont chargés en mémoire en vue d'une exécution entrelacée
- un programme qui demande une lecture, est **bloqué** au profit d'un autre qui récupère le CPU

éviter les conflits ?

plusieurs processus en RAM => précautions :-(

- mémorisation de l'état des registres du processeur pour chacun (RIP, RAX, ...) (**contexte** et **table des process**)
- à qui le tour ? -> **ordonnanceur**
- **protection** d'accès à la mémoire (**segmentation** sur intel)
- attribution de ressources non partageables -> problématique des **interblocages**

contexte et table des process

plusieurs processus en mémoire et pour chacun :

- une valeur de **RIP**
- une valeur de **RSP** pour sa **pile**
- un état de **registres** du CPU (RAX, RBX, ...)
- un **mode** de fonctionnement du CPU (privilégié ou non)
- un **état** (élu, prêt, bloqué)

C'est le **contexte** à mémoriser dans **la table des processus** lorsque un processus cède le CPU.

état des processus

L'état d'un processus est une indication pour l'ordonnanceur

- **bloqué** : dans l'attente d'une fin de lecture
- **prêt** : pourrait s'exécuter si il avait le processeur
- **élu** : s'exécute (un seul)

NB. Plusieurs demandes de lecture et donc plusieurs processus BLOQUES peuvent coéxister.

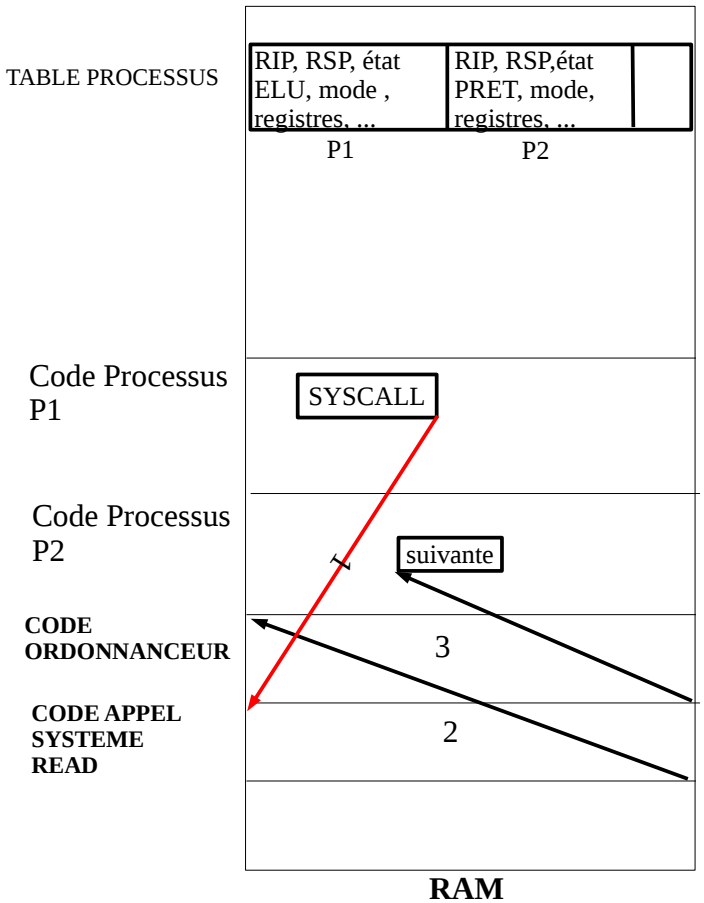
ordonnancement

ordonnanceur

Choisit un des processus **prêts** et lui attribue le CPU

multiprogrammation - cas Lecture 2

demande de lecture en multiprogrammation :



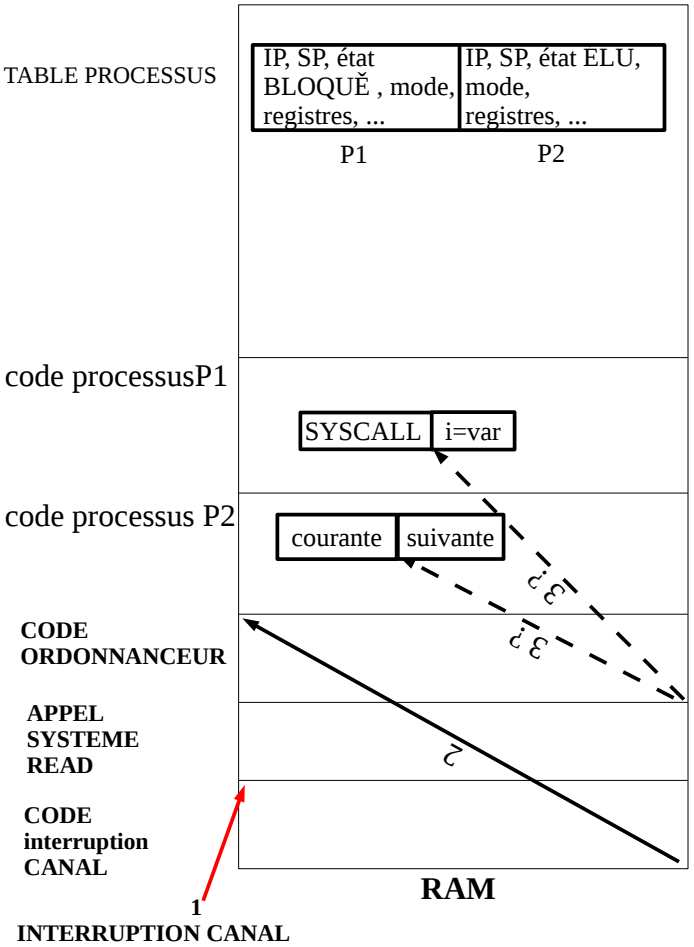
multiprogrammation - cas Lecture 2

Scénario "Demande de lecture"

1. P1 demande une lecture (SYSCALL)
   - SYSCALL
2. Appel Système
   - état P1 <– **bloqué**
   - sauvegarde le **contexte** de P1 dans la table
   - commande périphérique et canal
   - RIP < adresse de l'ordonnanceur
3. Ordonnanceur
   - état P2 dans la table <– **élu** (seul prêt)
   - charge les registres et le mode de P2 (depuis la table)
   - RIP <- RIP de P2 depuis la table
4. P2 a la main, il continue à s'exécuter

multiprogrammation - cas Lecture 2

Qu'en est-il de la **fin de lecture** ?

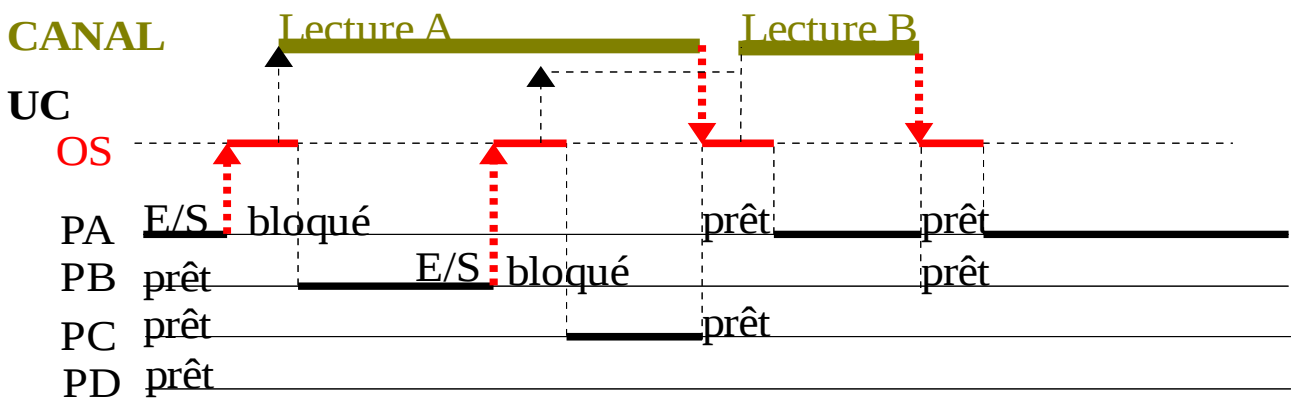


multiprogrammation - cas Lecture 2

Scénario "fin de lecture"

1. **interruption CANAL**
   - pile<-RIP et mode, mode<-privilégié, RIP<-adresse gestion d'interruption
2. **Traitement de l'interruption CANAL**
   - état P2 <– **prêt**
   - sauvegarde le contexte de P2 dans la table
   - état P1 dans la table <– **prêt** car fin de lecture
   - RIP <- adresse de l'ordonnanceur
3. **Ordonnanceur**
   - état P1 ou P2 dans la table <– élu
   - charge les registres et le mode de l'élu
   - RIP <- RIP de l'élu depuis la table
4. **l'élu a la main**

CPU et Canal dans le temps



CPU et Canal dans le temps

Et si le processus D avait eu la main en étant un processus de pur calcul ?

CPU et Canal dans le temps

Un processus rend la main lorsque il fait une lecture.

constat :

- les processus d'E/S rendent vite la main (gentils)
- les processus de calcul ne rendent jamais la main (consommateurs de CPU)

point fort - CPU bien rentabilisé

point faible -

canal non rentabilisé avec processus de calcul

time slicing

Time Slicing : motivation

Le Time Slicing a comme but de **rentabiliser le processeur Canal**

On va se servir de l'**interruption horloge** pour retirer le CPU au process qui l'utilise.

C'est un mécanisme de **préhemption**

time slicing

Time Slicing : le principe

time slice = tranche de temps

- un processus garde le CPU pendant une tranche de temps de **durée maximum** t
- après on le lui **retire** au profit d'un autre

-> permet un contrôle plus fin de l'enchaînement

-> permet de rentabiliser le Processeur Canal

time slicing

l'**ordonnanceur** est appelé par l'interruption horloge

inconvénients : gérer les **changements de contexte** prend du temps CPU, on ne pourra pas le faire trop souvent

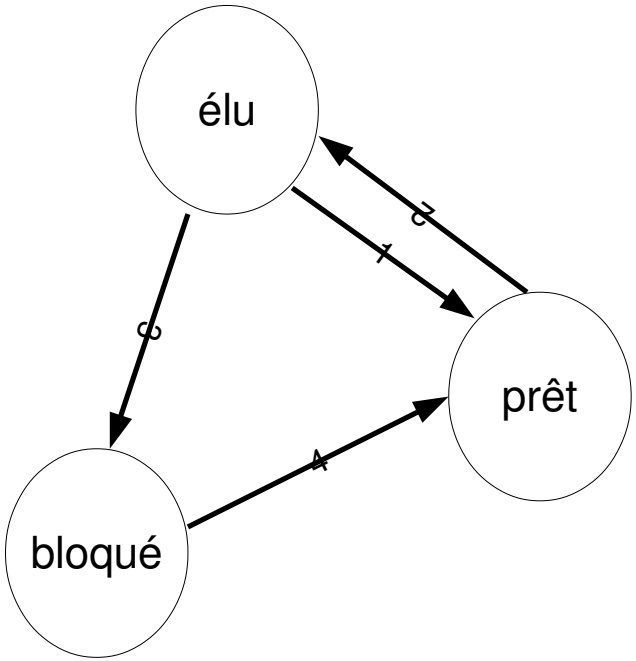
l'**ordonnanceur** est appelé par l'interruption horloge

inconvénients : gérer les **changements de contexte** prend du temps CPU, on ne pourra pas le faire trop souvent

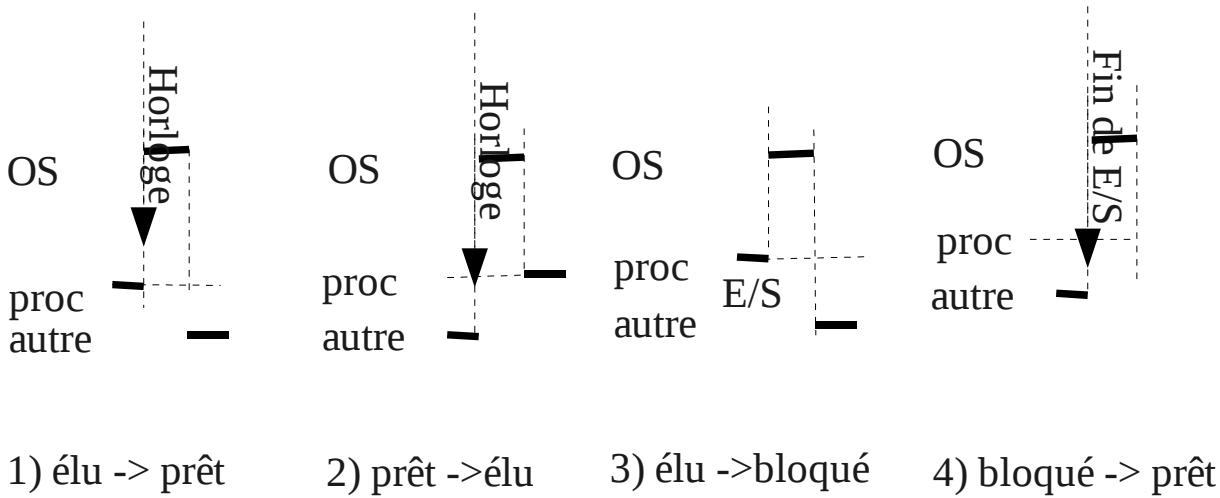
Cette gestion du CPU permettra également l'exploitation intéractive (**time sharing ou partage de temps**) car elle minimise les temps de réponse

time slicing

transitions d'état d'un processus



time slicing



préemption

système **non préemptif** : un processus se bloque (E/S, ...) ou rend la main spontanément

système **préemptif** : un processus se bloque (E/S, ...) ou rend la main spontanément, ou son temps est écoulé (Windows NT, 2K, XP, linux depuis noyau 2.6,...)

système **coopératif** : un processus rend la main spontanément (premiers windows)

time slicing

Un ordonnanceur préemptif ne peut exister sans interruptions horloge

Un ordonnanceur peut être non préemptif même en présence d'interruptions horloge

time slicing

quand a lieu l'ordonnancement ?

1. demande d'E/S
2. fin d'E/S
3. interruption horloge (préemption)
4. nouveau processus
5. fin d'un processus
6. ...

–> à la fin de chaque interruption / appel système (en simplifiant)

questions

1. si P est l'élu, alors RIP pointe vers une des instructions de P [V-F]
2. combien de processus en mémoire ?
3. combien de **prêts** en même temps ?
4. combien de **bloqués** en même temps ?

questions

1. un processus peut passer de l'état prêt à bloqué[V-F]
2. un processus peut passer de l'état bloqué à élu[V-F]
3. quels avantage(s)/inconvénient(s) a le Time Slicing par rapport à la Multiprogrammation ?
4. donnez des cas où l'état d'un processus bascule
   - de élu -> prêt
   - de bloqué -> prêt

Questions ?

remerciements

remerciements à P.Bettens et M.Codutti
pour la mise en page :-) Mba

Crédits

Ces slides sont le support pour la présentation orale de l'activité d'apprentissage **SYS2** à la HE2B-ÉSI

Crédits Crédits

La distribution opensuse
du système d'exploitation GNU Linux.
LaTeX/Beamer comme système d'édition.
GNU make, rubber, pdfnup, ... pour les petites tâches.

Images et icônes

deviantart, flickr, The Noun Project