4. `Réalisation du noyau` : PedagOS :

Objectif : description du fonctionnement d'un système complet appelé PedagOS

4.1.**Les composantes de La machine POP : Petit Ordinateur Personnel**

- **UC** : processeure Monoprocesseur
- **mémoire**
- **clavier**
- **écran**
- **disque**
- **registres généraux** : `reg`
- **mep:** c'est un registre qui stocke l'adresse prochiane de l'instruction qui serà exécute par le CPU .
- **Horloge:**
  - Un emplacement de mémoire noté `timer`
  - **les opérations :**
    - **Mise en route de l'horloge** = chargement dans timer d'une valeur positive
    - **Arrêt de l'horloge** : `timer <- 0` # masque des interruptions
    - **Décrémenté** : chaque milliseconde
    - **Déclenche une interruption** : si `timer = 0`

4.2 **Les Modes de notre SE :**

- **Mode Noyau (Mode Maître) :** Le mode noyau est le mode d'exécution qui donne au système d'exploitation un contrôle complet sur les ressources matérielles de l'ordinateur.
- **Mode Utilisateur (Mode Esclave) :** Le mode utilisateur est le mode d'exécution dans lequel les applications utilisateur s'exécutent. Dans ce mode, les applications n'ont qu'un accès limité aux ressources matérielles et ne peuvent pas effectuer d'opérations privilégiées directes. Elles doivent faire des appels système pour demander au noyau d'effectuer des opérations en leur nom.

4.3 **Les interruptions :**

- Vecteur d'interruption :
  - Contient les mots d'état des traitants
- Sauve le `mep` dans un emplacement fixe noté `ancien_mep`

- Charge dans `mep` l'entrée du vecteur d'interruption qui correspond à la cause de l'interruption
- On posséde une Instruction de retour d'interuption : `rti`
- `Masquage-démasquage` :
  - **Masquage** : Le masquage des interruptions est un mécanisme qui désactive temporairement les interruptions matérielles sur un processeur ou un cœur de processeur. Lorsque les interruptions sont masquées, le processeur ignore les demandes d'interruption et continue d'exécuter son programme actuel.
  - **Démasquage des interruptions** : Le démasquage des interruptions est le processus inverse du masquage. Lorsque les interruptions sont démasquées, le processeur redevient sensible aux demandes d'interruption. Cela signifie que les interruptions en attente d'être traitées seront maintenant traitées.
- `Appel système` :
  - pour le passage du `Mode escalve` au `mode maitre` : le SE lance une interruption
  - pour le retourt du `mode maitre` au `mode esclave` : le SE appéle l'instruction `rti`.

4.4. **Noyau de processus :**

- les composantes de la noyau de notre mini-os :

a. **États des processus :**

- **états :**
  - **Élu** : ce processus s'exécute
  - **Éligible** : attente de l'UC
  - **Bloqué** : attente dans une file de sémaphore
- **Transitions entre états :**
  - **Élu —> Bloqué :** si P(sémaphore.cpt) < 0
  - **Bloqué —> Éligible :** si un autre processus applique la fonction V sur le sémaphore .

b. **structure d'un processus :**

- Pour chaque processus, on conserve un contexte :
  - Registres généraux `reg`
  - Mot d 'état : `psw` l'instruction prochaine à éxecuter
  - Indicateurs `etat` : l'état d'un processus

- Priorité `prio` : son ordre de priorité pour l'ordonnacement
- Liens de chaînage
- Chaque processus est identifié par un `pid` :
  - ona : `pid->reg`, `pid->prio` …etc

```
struct descripteur_de_processus
{
    reg ; //registres généraux
    psw ; // mot d'état de
    programme
    etat ; // élu, éligible ou bloqué
    prio ; // priorité
    liens ;
}
```

c. **strucuture d'un sémaphores:**

- Pour chaque sémaphore :
  - Valeur du sémaphore `cpt`
  - Pointeur vers la file d'attente file `file`.
- Chaque sémaphore est identifié par un `sid` :
  - on les accées suivant : `sid->cpt`, `sid->file`

```
struct descripteur_de_semaphore
{
    cpt ; // valeur du semaphore
    file ; // file du semaphore
}
```

d. **files d'attente :**

- Une file d'attente par sémaphore
- Une file d'attente de l'UC
- Opérations standards sur une file `f` :
  - `entrer (pid,f)` // ajouter le processus pid a f .
  - `premier (f)` -> pid // return le tete de la file .
  - `sortir (f)` -> pid // return le queue de la file .
  - `vide (f)` -> booléen // return true si la file est vide .

e. **Allocation de l'unit centrale :**

- **ordonnaceur** : qui sera charger d'allouer l'UC et de decider à chaque instant le processus qui a le droit d'étre exécuter .

- le code de l'ordonnanceur se fonctionnne qq soit le mode Avec ou sans réquisition(on intermope le processus mais si il n'est pas d'accord).
- **Méthode du tourniquet :** (**quantum d'UC**, **f_eligibles**)
- pour la gestion de l'UC `ordonnacment` on choisit la méthode de tourniquet
- **Méthode du tourniquet :**
  - C'est une technique de gestion de processus qui permet de partager équitablement le temps CPU entre plusieurs processus en cours d'exécution
- **fonctionnement :**
  - **Sélection des processus :**
    - Les processus prêts à s'exécuter sont mis dans une file d'attente circulaire `f_eligibles`.
    - **Le tourniquet** choisit le premier processus dans la file pour l'exécution.
  - **Exécution :**
    - Le processus sélectionné est autorisé à s'exécuter pendant une certaine période de temps appelée **quantum d'UC**.
    - Ce quantum est généralement court, typiquement de quelques millisecondes.
  - **Commutation :** Une fois que le quantum d'UC est écoulé ou que le processus se bloque (par exemple, en attendant une entrée utilisateur), le tourniquet interrompt ce processus et le met en fin de file d'attente.
  - **Sélection du processus suivant :** Le tourniquet sélectionne ensuite le processus suivant dans la file d'attente et lui alloue le CPU pour un quantum d'UC. Ce processus est également interrompu après la fin du quantum ou s'il se bloque.
  - **Répétition :** Ce processus de sélection, d'exécution et de commutation se répète en continu, ce qui garantit que chaque processus obtient sa part équitable du temps CPU.
- **les avantages :**
  - L'avantage principal de cette méthode est qu'elle assure une utilisation équitable du CPU entre les processus actifs.
  - Elle est simple à mettre en œuvre et convient bien aux environnements multitâches.
- **les inconvientées :**

– elle n'est pas efficace en termes de performance, car les commutations fréquentes entre les processus introduisent un certain surcoût en termes de temps de commutation.