S.E.

Sections critiques (Communication)

On dispose d'1 zone et on ne veut pas que 2 ou plus processus "l'utilisent" en meme temps.

Section critique = partie du code exécuté exclusivement par les processus désignés à partager la ressource. Il y a une **"exclusion mutuelle"** lorsqu'un seul processus est habilité à accéder à la ressource.

QQ exemples

- pas 2 en meme temps (en cas d'exclusion mutuelle) ou i (si i est la borne)
- indépendant des vitesses des processus et du nombre de processus
- seul 1 processus en section critique bloque les autres
- On ne doit pas attendre indéfiniment l'accès à la section critique

Mécanismes

Masquage des interruptions, variables de verrouillage, sémaphores.

Rappel gestion des interruptions

- Il existe plusieurs types d'interruption : 0 → horloge, 1 → disque, 4 → tmp (appels systèmes), ...
- Une table d'interruptions qui appelle la routine associée à chaque interruption

Quelques conditions

Ces différentes étapes de la gestion des interruptions :

- sauvegarde le compteur ordinal, détermine le type d'interruptions et on passe en mode noyau pour permettre l'exécution de la routine associée.
  - Dans cette routine on peut stocker le contexte du processus, (en ASM)
  - On appelle le vrai code la routine (en général mix C, assembleur)
  - Au retour de la procédure on retourne le contexte du processus appelant si peut continuer, sinon on passe la main à l'ordonnanceur.

Masquage des interruptions (bloque les interruptions) et permet l'exécution d'un code en mode atomique.

- 1° instruction spéciale pour bloquer les interruptions

- on l'invoque avant d'entrer en section critique
- réactive les interruptions en fin de de section critique

Pas acceptable comme solution puisque qu'un processus peut s'accaparer toutes les ressources. Mais, reste 1° solution acceptable pour l'os (de temps en temps) dans **1 système mono-processeur**.

Variables de verrouillage

On définit une variable par **section critique** : c'est la variable qui conditionne à la section critique.

```
verrou = 1;
while (verrou == 1);
verrou = 1;
section_critique();
verrou = 0;
```

Cette solution n'est pas correcte car on n'a aucune garantie que la modif de verrou est atomique → il faut trouver un moyen de contourner la non-atomicité.

Solution 1 Masquage des interruptions**

Par ex le langage d'ispose d'une instruction qui permet de faire des affecations atomiques

Solution : Protéger nous-mêmes l'accès à la variable avec une seconde variable

1ère possibilité : une variable tour

```
while (tour != p)
    section_critique();
    tour = 1-p;
```

le processus le plus lent est attendu par les autres.

Vous comprendrez que cette solution ne marche que pour 2 processus.

Solution 2 (variable de Petersen)

```
int drapeau = {F, F};
int tour = 0;
enter_sec(int i) {
drapeau[i] = T;
tour = i;
while (drapeau[i+1]%2 == F || tour = (i+1) % 2)
    esc();
    sortie_esc(int i);
    drapeau[i] = F;
```

```
}
```

Exo étendre à n processus?

- Pour 1 exstension à n processus, m processeurs (avec 1 mécanisme de variable d'alternance)

On dispose d'une instruction matérielle test, set , lock → changer le contenu d'une mémoire On met une valeur non nulle) cette zone mémoire, l'atomicité garantit par un verrouillage du bus de données.

```
TSL(4 verrou) {old = verrou; verrou = 1; return old;}
enter_sl() {
    while (TSL(4 verrou))
        sortie_sc() {
            verrou = 0;
        }
}
```