Exclusion mutuelle et sémaphores

Partie 3 : exclusion mutuelle

*On nous demande d’ajouter au mini-noyau temps réel la gestion de la suspension et du réveil d’une tâche.

On a donc réalisé les deux primitives suivantes :

- `void dort(void)` : endort (suspend) la tâche courante
- `void reveille(register ushort tache)` : réveille la tâche tache. Le signal de réveil n’est pas mémorisé si la tâche n’est pas suspendue.*

Voici le code de ces fonctions :

```
void reveille(uint16_t t)
{
    CONTEXTE *p;
    p = &_contexte[t];

    // Erreur si tâche non créée
    if (p->status == NCREE)
    noyau_exit();

    // Protection des accès sur la file
    _lock_();
    if (p->status == SUSP) {
        p->status = EXEC;

        // Ajout de la tâche précédement endormie à la file des tâches à
exécuter
        ajoute(t);

        // Appel de la tâche suivante par le scheduler
        schedule();
    }
    _unlock_();
}

void dort(void)
{
    CONTEXTE *p;
    p = &_contexte[_tache_c];

    if (p->status == NCREE)
        noyau_exit();

    _lock_();
    if (p->status == EXEC) {
        p->status = SUSP;
        retire(_tache_c);
        schedule();
    }
    _unlock_();

}
```

Quand on suspend une tâche, on la retire donc de la file des tâches à exécuter et on la rajoute au réveil.

On réalise maintenant un producteur et un consommateur selon les besoins définis dans le sujet :
Vous écrirez un petit programme de test mettant en œuvre ces deux primitives dans le cas du modèle de communications producteur/consommateur. Le programme devra comporter au moins deux tâches ; la première, le producteur, produira des entiers courts dans une file circulaire, la seconde, le consommateur, retirera ces entiers de la file et les affichera.

Le producteur va s'endormir si le nombre de places vides est nul et il va réveiller le consommateur s'il y a un seul item dans la file. Le consommateur va s'endormir si le nombre places vides est égal à la taille de la file et va réveiller le producteur s'il n'y a plus qu'une place vide. Il peut donc y avoir les 2 tâches lancées en même temps mais elles ne peuvent pas être éteintes en même temps !

On va donc s'inspirer de notre file du TP précédent pour que les deux tâches se transmettent des entiers. En dehors de cette communication, on va les occuper pour simuler la préparation et le traitement des données.

ATTENTION !!
FIFO qu'on a faite --> pas très optimisée
Sémaphore permet d'avoir plusieurs producteurs et consommateurs sur la même ressource