TP5 : Threads et sémaphores

1 Utilisation des threads Linux

• Commencez par lire cette petite documentation sur les threads[1] et faites tourner le programme fourni en exemple. Vous pouvez utiliser la commande `ps -eLf` pour observer les *threads* créés par cet exemple.

• Essayez d'utiliser la fonction `sched_yield()` pour améliorer la fluidité de vos deux threads en organisation le passage de la CPU de l'un à l'autre.

2 Utilisation des sémaphores

2.1 Préparation

• Commencez par lire cette petite documentation sur les sémaphores[2] et faites tourner le programme fourni en exemple.

2.2 Algorithme de Peterson

• Dans cet exemple, remplacez l'exclusion mutuelle avec sémaphore par l'exclusion mutuelle avec l'algorithme de Peterson. Vérifiez que la consommation de CPU a augmenté. Pour mémoire, avec l'algorithme de Peterson, le processus Pi (avec i valant 0 ou 1) utilise les codes ci-dessous

```
init                demande[0] := faux
                    demande[1] := faux

prologue pour Pi    demande[i] := vrai;
                    tour := (1 - i);
                    Répeter
                    Jusqu'à (demande[1 - i] = faux) ou (tour = i);

                    < section critique >

épilogue pour Pi    demande[i] := faux
```

2.3 Producteur consommateur

• Reprenez le premier exemple des threads et remplacez le drapeau par deux sémaphores pour implanter un schéma producteur consommateur (le tampon est constitué d'un seul caractère). Rappel du schéma producteur/-consommateur :

```
NPlein : sémaphore := (0, {});
NVide : sémaphore := (n, {});  // Taille du tampon
```

Algorithme du producteur :

[1] ref:dil-doc-thread
[2] ref:dil-doc-thread-sem

```
Pour toujours faire
    P(NVide);
    < produire et déposer dans le tampon >
    V(NPlein);
Fin-pour
```

Algorithme du consommateur :

```
Pour toujours faire
    P(NPlein);
    < retirer du tampon et consommer >
    V(NVide);
Fin-pour
```

• Modifiez l'exemple précédent pour avoir maintenant un tampon d'une dizaine de caractères.

```
char tampon[ TAILLE_TAMPON ];
int ptr_entree = 0;
int ptr_sortie = 0;
```

La variable `ptr_entree` donne l'indice de la prochaine case vide du tampon tandis que `ptr_sortie` donne l'indice de la prochaine case pleine. Ces variables sont gérées comme des pointeurs circulaires.

Prenez soin de ralentir le producteur (avec `sleep`) puis dans un deuxième temps le consommateur pour tester tous les cas de figure (tampon vide ou tampon plein).