**Synchronisation par sémaphores**

**Exercice1** : La piscine (Latteux, 1980)

|  |
| --- |
| Processus Nageuri  début  **Entrée\_piscine;**  <Se baigner>  **Sortie\_piscine;**  Fin; |

|  |
| --- |
| **Procédure entrée\_piscine;**  Début  **Demander\_panier;**  **Demander\_cabine;**  < Se changer et ranger ses vêtements dans le panier>  **Libérer\_cabine;**  <Déposer le panier à la consigne>  Fin; |

|  |
| --- |
| **Procédure sortie\_piscine;**  Début  <Retirer le panier de la consigne>  **Demander\_cabine;**  <Se changer>  **Libérer\_panier;**  **Libérer\_cabine;**  Fin; |

1. Les points et les contraintes de synchronisation :

**Demander\_panier**: Condition de franchissement: nombre de paniers occupés < np; ou npo < np.

**Demander\_cabine** : Condition de franchissement : nombre de cabines libres > 0 ; ou ncl > 0.

**Libérer\_panier** et **Libérer\_cabine**: ces points ne sont pas bloquants, la condition de franchissement est toujours vraie.

2- Pour réaliser le programme d’un nageur on utilise :

* Un sémaphore représentant le nombre de cabines libres.
* Une variable représentant le nombre de paniers demandés « ndp ».
* Une variable représentant le nombre de paniers occupés « nop ».
* Un sémaphore d’exclusion mutuelle protégeant ces deux variables « ndp » et « nop » et un sémaphore de synchronisation pour mettre en attente les processus.

**1ière solution** :

|  |  |
| --- | --- |
| Sémphore scab init(nc)  Entier ndp,npo init(0)  Sémaphore mutex init(1)  Sémaphore span init(0) | |
| **Procédure Demander\_panier** | **Procédure Libérer\_panier** |
| Début  P(mutex) ;  ndp++ ;  Si npo = np alors  V(mutex) ;  P(span);  Sinon  ndp--;  npo++;  V(mutex);  Finsi ;  Fin; | Début  P(mutex) ;  npo-- ;  Si ndp >0 alors  ndp-- ;  npo++ ;  V(span);  finsi  V(mutex);  Fin; |
| **Procédure Demander\_cabine** | **Procédure Libérer\_cabine** |
| Début  P(scab)  Fin | Début  V(scab)  Fin |

**2ière solution** :

Dans la **1ière solution**, c’est le processus qui libère le panier qui après avoir réveillé un processus en attente fait les modifications nécessaires sur les variables partagées ndp et npo (décrémentation de ndp et incrémentation de npo). Dans la **2ième solution** c’est le processus réveillé qui fait lui-même les modifications nécessaires sur les variables ndp et npo dans son programme.

|  |  |
| --- | --- |
| Sémphore scab init (nc)  Entier ndp,npo init (0)  Sémaphore mutex init (1)  Sémaphore span init (0) | |
| **Procédure Demander\_panier** | **Procédure Libérer\_panier** |
| Début  P(mutex) ;  ndp++ ;  Si npo = np alors  V(mutex) ;  P(span);  Fsi  ndp--;  npo++;  V(mutex);  Fin; | Début  P(mutex) ;  npo-- ;  Si ndp >0 alors  V(span);  Sinon  V(mutex);  Fsi  Fin; |
| **Procédure Demander\_cabine** | **Procédure Libérer\_cabine** |
| Début  P(scab)  Fin | Début  V(scab)  Fin |

3) Il y a risque d’interblocage si l’on demande les cabines avant les paniers, comme suit :

**Procédure entrée\_piscine;**

Début

**Demander\_cabine;**

**Demander\_panier;**

< Se changer et ranger ses vêtements dans le panier>

**Libérer\_cabine;**

<Déposer le panier à la consigne>

Fin;

Explication par un cas :

On suppose que nc = 4 cabines et np = 10 paniers

- Supposons qu’à l’instant t on a la situation suivante : 10 nageurs sont dans la piscine en train de nager, donc npo = 10 (nombre de paniers occupés) et ncl = 4 (les 4 cabines sont libres)

- à l’instant t+1 : 4 processus arrivent et demandent des cabines, leur demande est satisfaite vu qu’il y a des cabines libres, après l’affectation on aura ncl=0 (il ne reste plus de cabines libres), ils passent ensuite pour demander les paniers, ils vont rester **bloqués** enattendant que des paniers soient disponiblescar tous les paniers sont pris npo = np = 10.

- à l’instant t+2 : des processus veulent quitter la piscine, récupèrent leurs paniers à la consigne et demandent des cabines pour se changer, ils vont rester eux aussi **bloqués** car il n’y a pas de cabines libres (ncl=0) → **Situation d’interblocage** (blocage indéfini) entre les processus qui veulent sortir de la piscine et ceux qui veulent y entrer.

**Exercice 7 : Allocateur de ressources**

1. **Nombre de processus connu : n**

- La procédure **Allouer** a comme paramètres l’identificateur du processus et le nombre de ressources demandées (nbd).

- La procédure **Libérer** a comme paramètre le nombre de ressources à restituer (nbr).

- Ces deux procédures doivent être exécutées en exclusion mutuelle, car la variable nbres (qui représente le nombre de ressources disponibles) ne peut pas être utilisée en même temps par plusieurs procédures Allouer ou/et Liberer;

Soit mutex le sémaphore protégeant la variable nbres.

- On utilise :

- Une liste pour sauvegarder le nombre de ressources demandées et l’identificateur du processus.

- Un tableau de n sémaphores : un sémaphore de synchronisation par processus; ces sémaphores permettent de mettre les processus en attente; ces sémaphores sont initialisés à zéro.

**Structures et procédures utilisées :**

Entier nbres init nmax ;

sem : tableau[0..n-1] de sémaphore init 0 ;

Sémaphore mutex init 1;

Liste = Structure

idfp : [0..n-1] ;

nb : [0..nmax];

Suivant : \*liste ;

fin;

pliste pointeur sur liste init null;

- Procédure **insérer** (pliste, idfp, nbd, ordre) : permet d’insérer un nouvel élément contenant (idfpn,nb) dans la liste selon l’ordre indiqué en paramètre par rapport au nombre de ressources demandées par le processus (la liste doit être triée par ordre croissant des demandes).

- Procédure **retirer** (pliste, idfp, nbd): Affecte le contenu de l'élément pointé par pliste aux variables idfp et nbd et ensuite supprime cet élément de la liste.

- Fonction tête (pliste): renvoie le contenu du champ nb contenu dans le premier élément de la liste.

|  |  |
| --- | --- |
| **Allouer(idfp, nbd)** | **Liberer(nbr)** |
| **Début**  P(mutex);  **Si** nbd > nbres **alors**  /\* Attente \*/  insérer(pliste, idfp, nbd, croissant);  V(mutex);  P(sem[idfp]); /\* Attendre  **Sinon**  nbres := nbres - nbd;  V(mutex);  **Finsi**  **Fin** | Entier nbd, idf;  **Début**  P(mutex);  nbres := nbres + nbr ;  /\* Safistaire les demandes de processus en attente \*/  **Tantque** (pliste ≠ nul) et (tête(pliste)≤ nbres ) **faire**  retirer(pliste, idf , nbd) ;  nbres := nbres – nbd ;  V(sem[idf]);  **fintanque**  V(mutex);  **Fin** |

**Programme d’un processus i**

Entier nbd ;

Début

……

Allouer(i,nbd) ; /\* i :N° du processus

<utilisation des ressources>

Liberer(nbd) ;

……

Fin ;

**2) Nombre de processus n’est pas connu**

- La procédure *Allouer* aura comme paramètres un **sémaphore défini** par le processus et le nombre de ressources demandées (*nbd*).

- La procédure *Liberer* a comme paramètre le nombre de ressources à restituer (*nbr*).

- On utilise une liste pour sauvegarder le nombre de ressources demandées et un pointeur sur un sémaphore (*sem*). Ce sémaphore est déclaré et initialisé à zéro par le processus qui demande les ressources. Ce sémaphore permet de mettre le processus dont la demande ne peut pas être satisfaite en attente. On utilise un sémaphore de synchronisation par processus.

**Structures et procédures utilisées :**

Entier nbres init nmax ;

Sémaphore mutex init (1) ;

Liste = Structure

Sem : \*semaphore ; /\* pointeur sur un sémaphore

nb : entier ;

Suivant : \*liste ;

fin;

pliste pointeur sur liste init null;

- Procédure **insérer**(pliste, sem, nb,croissant)

- Procédure **retirer**(pliste, sem, nb) : Affecte le contenu de l'élément pointé par pliste dans sem, nb et supprime cet élément de la liste.

- Fonction **tête**(pliste) : renvoie le contenu du champ ‘nb’ du premier élément de la liste.

|  |  |
| --- | --- |
| **Allouer(sem, nbd)** | **Liberer(nbr)** |
| **Début**  P(mutex);  **Si** nbd > nbres **alors**  /\* Attente \*/  insérer(pliste, sem, nbd,croissant);  V(mutex);  P(sem);  **Sinon**  nbres := nbres - nbd;  V(mutex);  **Finsi**  **Fin** | Entier nbd;  Sem \*sémaphore ;  **Début**  P(mutex);  nbres := nbres + nbr ;  /\* Safistaire les demandes de processus en attente \*/  **Tantque** (pliste ≠ nul) et (tête(pliste)≤ nbres ) **faire**  retirer(pliste, sem , nbd) ;  nbres := nbres – nbd ;  V(sem);  **fintanque**  V(mutex);  **Fin** |

**Programme d’un processus i**

Entier nbd ;

Sémaphore S init 0 ;

Début

…

Allouer(S,nbd) ;

<utilisation des ressources>

Liberer(nbd) ;

…

Fin ;