Sémaphore : définition

- Un sémaphore s est l'ensemble d'un compteur entier noté s.c et d'une file d'attente s.f
- Deux opérations qui sont exécutées en exclusion mutuelle
- P(s) s.c--; si s.c<0 alors le processus qui exécute P est bloqué dans s.f
- V(s) s.c++ ; si s.c<=0 alors activer un processus de s.f

Propriétés des sémaphores

- Soit c0 la valeur initiale d 'un sémaphore s, c0 >=0
- Si la valeur s.c >o, elle représente le nombre de processus qui peuvent exécuter P sans se bloquer (en l 'absence de tout nouveau V)
- Si la valeur s.c<0, elle représente le nombre de processus de s.f

Producteurs / Consommateurs

- Producteur :

```
…
produire(message_emis);
<???>
entrer(message_emis);
<???>
…
```

- Consommateur :

```
…
<???>
sortir(message_recu);
<???>
consommer(message_recu)
…
```

```
semaphore mutex vi : 1;  -- contrôle d'accès à la section critique
semaphore vide vi : N; -- nb d'emplacements libres
semaphore plein vi : 0; -- nb d'emplacements occupés
```


Producteurs / Consommateurs

- Producteur :

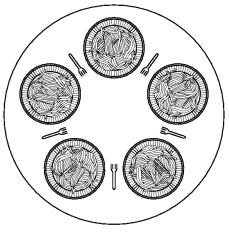
```
…
produire(message_emis);
P(vide);
P(mutex);
entrer(message_emis);
V(mutex);
V(plein);
…
```

- Consommateur :

```
…
P(plein);
P(mutex);
sortir(message_recu);
V(mutex);
V(vide);
consommer(message_recu)
…
```


Problème des philosophes

- 5 philosophes (#0 à 4) doivent penser et manger (phases de durées finies)
- Chaque philosophe a une place fixe et doit utiliser 2 fourchettes pour manger (obligatoirement celle à sa gauche et celle à sa droite)



Problème des philosophes

```
const   N       5; -- nb de philosophes
        pense   0;
        faim    1;
        mange   2;
        gauche (i-1)%N;
        droite  (i+1)%N;
var     etat : tableau[0..N-1] de entier;
        semaphore mutex vi : 1;
        spriv : tableau[0..N-1] de semaphore vi : 0; --  0 dans chaque case
```

Problème des philosophes

- philosophe (i) :

```
tantque (vrai) faire
   penser();
   prendre_fourchettes(i);
   manger();
   poser_fourchettes(i);
ftantque
```

- prendre_fourchettes(i) :

```
P(mutex);
etat[i] := faim;
test(i);
V(mutex);
P(spriv[i]);
```


Problème des philosophes

- poser_fourchettes(i) :

```
P(mutex);
etat[i] := pense;
test(gauche);
test(droite);
V(mutex);
```

- test(i) :

```
si (etat[i]=faim) et (etat[gauche]!=mange) et
   (etat[droite]!=mange) alors
   etat[i]:= mange;
   V(spriv[i]);
fsi;
```


Problème des philosophes (variante de base)

- Solution (FAUSSE !) avec un sémaphore par fourchette
- sem_t fourchettes[N];

```
prendre_fourchettes(i)
{
  P(fourchettes[i])
  P(fourchettes[(i+1)%N])
}
```

```
poser_fourchettes(i)
{
  V(fourchettes[i])
  V(fourchettes[(i+1)%N])
}
```

Problème des philosophes (variante 1)

- Solution avec un sémaphore par fourchette, tous prendre les fourchettes dans le même ordre (sauf un philosophe gaucher)
- sem_t fourchettes[N];

```
prendre_fourchettes(i)
{
 If (i == N-1) {
  P(fourchettes[(i+1)%N])
  P(fourchettes[i])
 }
 else {
  P(fourchettes[i])
  P(fourchettes[(i+1)%N])
 }
}
```

```
poser_fourchettes(i)
{
  V(fourchettes[i])
  V(fourchettes[(i+1)%N])
}
```

Problème des philosophes (variante 2)

- Solution avec un sémaphore par fourchette, les philosophes se servent l'un après l'autre (FIFO)
- sem_t fourchettes[N]; sem_t fifo = 1;

```
prendre_fourchettes(i)
{
  P(fifo)
  P(fourchettes[i])
  P(fourchettes[(i+1)%N])
  V(fifo)
}
```

```
poser_fourchettes(i)
{
  V(fourchettes[i])
  V(fourchettes[(i+1)%N])
}
```

Problème des philosophes (variante 3)

- Solution avec un sémaphore par fourchette, N-1 philosophes seulement peuvent essayer de prendre une fourchette
- sem_t fourchettes[N]; sem_t table = N-1;

```
prendre_fourchettes(i)
{
  P(table)
  P(fourchettes[i])
  P(fourchettes[(i+1)%N])
  V(table)
}
```

```
poser_fourchettes(i)
{
  V(fourchettes[i])
  V(fourchettes[(i+1)%N])
}
```

Lecteurs / Redacteurs

- Maintien de la cohérence

(nred = 0) et (nlect >= 0) -- fichier en lecture

ou (nred = 1) et (nlect = 0)-- fichier en écriture

- Priorité aux lecteurs
- semaphore mutex vi : 1; -- controle l'acces a nl
  semaphore fich vi : 1; -- controle l'acces aux donnees
  entier nl := 0; -- nb de lectures en cours ou en attente

Lecteurs / Redacteurs

- Lecteur :

```
…
P(mutex);
nl := nl + 1;
si (nl = 1) alors P(fich); fsi;
V(mutex);
lire_fichier();
P(mutex);
nl := nl-1;
si (nl = 0) alors V(fich);
V(mutex);
…
```

- Redacteur :

```
…
P(fich);
ecrire_fichier();
V(fich);
…
```


Lecteurs / Redacteurs FIFO

- Lecteur :

```
P(fifo)
P(mutex);
nl := nl + 1;
si (nl = 1) alors P(fich); fsi;
V(mutex);
V(fifo)
lire_fichier();
P(mutex);
nl := nl-1;
si (nl = 0) alors V(fich);
V(mutex);
…
```

- Redacteur :

```
…
P(fifo)
P(fich);
ecrire_fichier();
V(fich);
V(fifo)
…
```


Les moniteurs : un outil de synchronisation

- Définition : variables d ’état + procédures utilisant ces variables
- Procédures en exclusion mutuelle
- Condition : variable spéciale
  - C.attendre: blocage du processus appelant « en attente de C »
  - C.signaler : réveil d ’un processus en attente de C

Moniteur en sémaphore(1)

1 – Accès aux procédures du moniteur en exclusion mutuelle

- Mutex d’entrée dans la moniteur : sémaphore mut_mon initialisé à 1

- Entree_procedure_moniteur
  - P(mut_mon)
- Sortie_moniteur
  - V(mut_mon)

Moniteur en sémaphore (2)

1 – Chaque conditions c

- Réalisation à base de sémaphore initialisé à 0 : S_c

- 1ere approximation c.signaler
  - V(S_c)
  - **PB: ne faire que s’il y a des processus en attente : ajout d’un compteur initialisé à 0 (nb_c))**
  - 2ème approximation c.signaler :
    - Si (nb_c >0) alors nb_c--; V(s_c);
  - C.attendre : nb_c++; sortie_moniteur; P(s_c);

Pb : 2 processus sont actifs en même temps : nécessité de bloquer le processus ayant fait un signaler

Moniteur en sémaphore (3)

Pb : 2 processus sont actifs en même temps : nécessité de bloquer le processus ayant fait un signaler uniquement s'il y a des processus en attente:

⇒Ajout sémaphore s_sig et d'un compteur nb_sig : nombre de processus ayant fait un signal et en attente de dispo d'un moniteur.

c.signaler	sortie_moniteur
Si (nb_c >0) alors	Si (nb_sig>0) alors
nb_c--; nb_sig++;	nb_sig--; V(s_sig);
V(s_c);	sinon
P(s_sig);	V(mut_mon);
Fin si	Fin si

Moniteur en sémaphore (4)

```
sem s_c, s_d; sem mutex; nb_s_c=0; nb_s_d=0; n_sig_c=0; sem sig;
f() {
  P(mutex)
  If (test) { nb_s_c ++; P(s_c) }
  If (nb_s_d > 0) { nb_s_d--; n_sig ++;  V(s_d); P(sig) }
  If (n_sig > 0) V(sig)
  Else V(mutex);
}
```