Interblocage de processus

Conditions nécessaires (simultanées) d'interblocage :

- Exclusion mutuelle
- Occupation et attente : Processus occupant au moins une ressource et qui attends d'acquérir des ressources supplémentaires détenues par d'autres processus
- Pas de réquisition : Les ressources déjà détenues ne peuvent être retirées de force à un processus
- Attente circulaire

Si chaque type de ressource possède exactement un seul exemplaire alors : il y a situation d'interblocage SSI le graphe d'allocation possède un circuit
Cas des ressources possédant plusieurs exemplaires : Si le graphe d'allocation est sans circuit alors un processus n'est pas dans une situation d'interblocage (réciproque fausse)
=> Interblocage s'il existe un sous-ensemble D (ensemble des sommets formant un cicuit) non vide de processus tel que, pour tout Pi appartenant à D, l'inégalité : DEMANDE[i] <= N - somme(ALLOCATION) avec N[i] nombre max de la ressource i est fausse

Evitement des interblocages => Algorithme du banquier (évalue le risque d'interblocage pouvant être provoqué par une demande de ressource) :

1. Vérifier la cohérence de la requête : DEMANDE <= BESOIN
2. Vérifier la disponibilité des ressources : DEMANDE <= DISPONIBILITE
3. Accepter temporairement la demande et vérifier l'état du système :
   DISPONIBLE = BESOIN[i] - DEMANDE
   ALLOCATION[i] = :old.ALLOCATION[i]+DEMANDE
   BESOIN[i] = :old.BESOIN[i]+DEMANDE
4. Appliquer l'algorithme de détermination d'un état sain : TRAVAIL = DISPONIBLE
   Pour tous les Pi, vérifier que BESOIN[i] <= TRAVAIL, si oui TRAVAIL += ALLOCATION[i], si non => système malsain

Détection ds interblocages et reprise : Graphe d'attente si toutes les les ressources possèdent une seule instance (on élimine les noeuds de type ressource)
Interblocage SSI le graphe d'attente contient un circuit

Ordonnancement de processus

Critères d'ordonnancement :

- Rendement d'utilisation du CPU : pourcentage de temps pendant lequel le CP est actif => Plus élevé, mieux c'est
- Utilisation globale des ressources : assurer une occupation maximale des ressources de la machine et minimiser le temps d'attente pour l'allocation d'une ressource à un processeur
- Equité : capacité de l'ordonnanceur (module du système d'exploitation qui attribue le contrôle du CPU à tour de rôle aux différents processus en compétition) à allouer le CPU d'une façon équitable à tous les processus de même priorité (éviter la famine)
- Temps de rotation : durée moyenne nécessaire pour qu'un processus termine son execution
- Temps d'attente : durée moyenne qu'un processus passe à attendre le CPU
- Temps de réponse : temps moyen qui s'écoule entre le moment oùun utilisateur soumet une requête et celui où il commence à recevoir les réponses
  TMT = (somme(date de fin d'exécutionn - date d'arrivée du processus))/nombre de processus

Algorithmes d'ordonnancement :

- **FCFS** (First Come First Served) : Les tâches sont ordonnancées dans l'ordre où elles sont venues
- **SJF** (Shortest Job First) : Le CPU est attribué au processus dont le temps d'exéction estimé est minimal
- **SRT** (Shortest Time Remaining) : Le CPU est attribué au processus qui a le plus petit temps d'execution restant (réévaluation à chaque quantum)
- **RR** (Tourniquet) : Le controle du CPu est attribué a chaque processus pendant un quantum

Algorithmes avec priorité : CPU attribué au processus de plus haute priorité

Threads

Toutes les fonctions liées aux threads

```
#include <pthread.h>
pthread_t threads[nth];
pthread_mutex_t mutex;
pthread_cond_t event;
is = pthread_mutex_init(&mutex, NULL);
is = pthread_cond_init (&event, NULL);
is = pthread_mutex_lock(&mutex);
is = pthread_mutex_unlock(&mutex);
is = pthread_cond_signal(&event);
is = pthread_cond_broadcast(&event);
is = pthread_cond_wait(&event,&mutex);
is = pthread_create(&threads[i], NULL, th_fonc, (void *)i);
is = pthread_join( threads[j], &val);
```

Exemple de programme

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define nth 3 /* nbre de threads a lancer */
```

```c
#define ifer(is,mess) if (is==-1) perror(mess)

pthread_t threads[nth]; // tableau qui contient les threads
pthread_mutex_t mutex; // MUTEX COMMUN A TOUS LES THREADS
pthread_cond_t event; // CONDITION DE MUTEX

/* routine exécutée dans les threads */
void *th_fonc (void *arg) {
    int is, numero = (int)arg; // numéro = numéro du thread (i de pthread_create)
    printf("ici thread %d, i=%d\n",numero);
    is = pthread_mutex_lock(&mutex); // retourne 0 si succès. Alternative, non blocante : pthread_mutex_trylock()
    // zone critique
    is = pthread_mutex_unlock(&mutex);
    return ((void *)(numero)+101); // ou pthread_exit(&numero)
}

int main() {
  int is; // code de retour pour chaque thread
  int i=0; // numéro des threads
  void *val=0; // contient la valeur de retour des threads
  is = pthread_mutex_init(&mutex, NULL); // initialise le mutex
  /* créer les threads */
  for(i=0; i<nth; i++) {
      printf("ici main, création thread %d\n",i);
      is = pthread_create( &threads[i], NULL, th_fonc, (void *)i );
      ifer (is,"err. création thread");
  }
  /* attendre fin des threads */
  for(i=0; i<nth; i++) {
      is = pthread_join( threads[i], &val); // val contient la valeur retournée par chaque thread
      ifer (is,"err. join thread");
      printf("ici main, fin thread j=%d val=%d\n",i,(int)val); // val est de type (void*)
  }
  exit(EXIT_SUCCESS);
}
```

Gestion de la mémoire

Partitions multiplec contiguës :

- Contiguës fixes :
  - files d'attente separées
  - files d'attente communes
- Contiguës dynamique :
  - First fit (selon les adresses croissantes)
  - Best fit (selon les tailles croissantes)
  - Worst fit (selon les tailles décroissantes)
- Contiguë siamoise

Algorithme de remplacement de pages :

- Algorithme optimal : Lors d'un défaut de page, choisit comme victime une page qui ne fera l'objet d'aucune référence ultérieure ou qui fera l'objet de la référence la plus tardive
- FIFO : Choisit comme victime la page la plus anciennement chargée
- LRU : Choisit comme victime la page ayant fait l'objet de la référence la plus ancienne

Calcul du nombre de mots par page :

$$Nombres\ de\ mots\ par\ page = nbmots/nbpages \tag{1}$$