1. **Exercices**
2. Décrivez les différences entre la planification à court, moyen et long terme.

**Réponse :**

* 1. **Court terme (ordonnanceur CPU)** - sélectionne parmi les tâches en mémoire celles qui sont prêtes à être exécutées et leur alloue le CPU.
	2. **Moyen terme** utilisé en particulier avec les systèmes de partage du temps comme niveau d'ordonnancement intermédiaire. Un système de permutation est mis en œuvre pour retirer de la mémoire les programmes partiellement exécutés et les réintégrer ultérieurement pour qu'ils continuent là où ils se sont arrêtés.
	3. **Long terme (job scheduler)** détermine quels travaux sont amenés en mémoire pour être traités. La principale différence réside dans la fréquence de leur exécution. Le court terme doit sélectionner un nouveau processus assez souvent. Le long terme est utilisé beaucoup moins généralement, car il s'occupe de placer les travaux dans le système et peut attendre un certain temps qu'un travail se termine avant d'en admettre un autre.
1. Considérez le mécanisme RPC (Remote Procedure Call).
* Décrivez les circonstances indésirables qui pourraient résulter de la non-application de la sémantique "au plus une fois" ou "exactement une fois".
* Décrivez les utilisations possibles d'un mécanisme qui n'offrirait aucune de ces garanties.

**Réponse :**

* Si un mécanisme RPC ne peut pas supporter la sémantique "au plus une fois" ou "au moins une fois", alors le serveur RPC ne peut pas garantir qu'une procédure distante ne sera pas invoquée plusieurs fois.
* Imaginons qu'une procédure distante retire de l'argent d'un compte bancaire sur un système ne supportant pas cette sémantique.
* Il est possible qu'une seule invocation de la procédure distante entraîne de multiples retraits sur le serveur.
* Pour qu'un système supporte l'une ou l'autre de ces sémantiques, il faut généralement que le serveur maintienne une certaine forme d'état du client, comme l'horodatage décrit dans le texte.
* Si un système n'était pas en mesure de prendre en charge l'une ou l'autre de ces sémantiques, il ne pourrait fournir en toute sécurité que des procédures à distance qui ne modifient pas les données ou ne fournissent pas de résultats sensibles au temps.

Si l'on prend l'exemple de notre compte bancaire, nous avons certainement besoin de la sémantique "au plus une fois" ou "au moins une fois" pour effectuer un retrait (ou un dépôt !). Cependant, une demande de renseignements sur le solde du compte ou sur d'autres informations relatives au compte, comme le nom, l'adresse, etc. ne nécessite pas cette sémantique.

1. Donnez deux exemples de programmation dans lesquels le multithreading n'offre pas de meilleures performances qu'une solution monofilaire

**Réponse :**

* 1. Tout type de programme séquentiel n'est pas un bon candidat pour le multithreading. Un exemple de ceci est un programme qui calcule une déclaration d'impôts individuelle.
	2. Un autre exemple est un programme "shell" tel que le shell C ou le shell Korn. Un tel programme doit surveiller de près son propre espace de travail, comme les fichiers ouverts, les variables d'environnement et le répertoire de travail actuel.
1. Quelle est la signification de l'expression "attente active" ? Quels autres types d'attente existe-t-il dans un système d'exploitation ? Peut-on éviter complètement l'attente active ? Expliquez votre réponse.

**Réponse :**

L'attente active signifie qu'un processus attend qu'une condition soit satisfaite dans une boucle serrée sans abandonner le processeur. Alternativement, un processus peut attendre en abandonnant le processeur, et se bloquer sur une condition et attendre d'être réveillé à un moment approprié dans le futur. L'attente peut être évitée, mais elle entraîne les frais généraux associés à la mise en veille d'un processus et à la nécessité de le réveiller lorsque l'état approprié du programme est atteint.

1. Expliquez pourquoi les spinlocks ne sont pas appropriés pour les systèmes monoprocesseurs, mais sont souvent utilisés dans les systèmes multiprocesseurs.

**Réponse :**

Les spinlocks ne sont pas appropriés pour les systèmes monoprocesseurs, car la condition qui permettrait à un processus de sortir du spinlock ne peut être obtenue qu'en exécutant un processus différent. Si le processus ne cède pas le processeur, les autres processus n'ont pas la possibilité de définir la condition de programme requise pour que le premier processus puisse progresser. Dans un système multiprocesseur, d'autres processus s'exécutent sur d'autres processeurs et modifient ainsi l'état du programme afin de libérer le premier processus du spinlock.

1. Le programme suivant permet-il de résoudre le problème d’accès à la section critique ?

*Tour=0*

*while (1)*

*{ // attente active*

*while (tour !=0) ;*

*Section\_critique() ;*

*tour = 1 ;*

*section\_noncritique() ;*

*...*

*}*

*// Processus P2*

*while (1)*

*{ // attente active*

*while (tour !=1) ;*

*Section\_critique() ;*

*tour = 0 ;*

*Section\_noncritique() ;*

*...*

*}*

**Réponse :** On peut vérifier assez facilement que deux processus ne peuvent entrer en section critique en même temps, toutefois le problème n'est pas vraiment résolu car il est possible qu'un des deux processus ait plus souvent besoin d'entrer en section critique que l'autre; l'algorithme lui fera attendre son tour bien que la section critique ne soit pas utilisée. Un processus peut être bloqué par un processus qui n'est pas en section critique.

P1 lit la valeur de tour qui vaut 0 et entre dans sa section critique. Il est suspendu et P2 est exécuté. P2 teste la valeur de tour qui est toujours égale à 0. Il entre donc dans une boucle en attendant que tour prenne la valeur 1. Il est suspendu et P1 est élu de nouveau. P1 quitte sa section critique, met tour à 1 et entame sa section non critique.

Il est suspendu et P2 est exécuté. P2 exécute rapidement sa section critique, tour = 0 et sa section non critique. Il teste tour qui vaut 0. Il attend que tour prenne la valeur 1.

1. Le programme suivant permet-il de résoudre le problème d’accès à la section critique ?
* 1 :

*Occupé est un booleen := false*

*Entrer\_sc()*

*{*

*While (occupé)*

*Occupé=TRUE*

*}*

*Sortir\_sc()*

*{*

*Occupé= FALSE*

*}*

Non il y’a problème, si tous les deux arrivent en même temps, ils trouvent que c’est faux ils peuvent entrer en même temps à la section critique, et donc il y’a un problème comme vu au cours.

* 2 :

*Int tour=0 ;*

*Bool je\_veux\_entrer [2]*

*Entrer\_sc () {*

*Je\_veux\_entrer[i] = TRUE*

*Tour = i*

*Attendre (je\_veux\_entrer[1 – i] == FALSE et tour == 1 – i )*

*}*

*Sortir\_sc () {*

*Je\_veux\_entrer[i]=FALSE*

 *}*

**Réponse :**

Cette méthode marche pour deux processus, mais elle est mauvaise, elle occupe le processus à 100% pendant le quantum de temps du processus.

1. Décrivez comment l'instruction Swap() peut être utilisée pour fournir une exclusion mutuelle qui satisfait à l'exigence d'attente limitée.

**Réponse :**

****

1. Les serveurs peuvent être conçus pour limiter le nombre de connexions ouvertes. Par exemple, un serveur peut souhaiter n'avoir que *N* connexions de socket à un moment donné. Dès que *N* connexions sont établies, le serveur n'accepte pas d'autres connexions entrantes jusqu'à ce qu'une connexion existante soit libérée. Expliquez comment les sémaphores peuvent être utilisés par un serveur pour limiter le nombre de connexions simultanées.

**Réponse :**

Un sémaphore est initialisé au nombre de connexions socket ouvertes autorisées. Lorsqu'une connexion est acceptée, la méthode **P** (acquire()) est appelée, lorsqu'une connexion est libérée, la méthode **V**(release()) est appelée. Si le système atteint le nombre de connexions socket autorisées, les appels ultérieurs à acquire() se bloqueront jusqu'à ce qu'une connexion existante soit terminée et que la méthode release soit invoquée.

1. Montrez comment implémenter les opérations de sémaphore wait() et signal() dans des environnements multiprocesseurs en utilisant l'instruction TestAndSet(). La solution doit présenter un minimum d'attente occupée.

**Réponse :**

Voici le pseudo-code pour implémenter les opérations :



1. **Le problème du coiffeur endormi**. Un salon de coiffure se compose d'une salle d'attente avec n chaises et d'une salle de coiffure avec une chaise de coiffeur. S'il n'y a aucun client à servir, le coiffeur se couche. Si un client entre dans le salon de coiffure et que toutes les chaises sont occupées, alors le client quitte le salon. Si le coiffeur est occupé, mais que des chaises sont disponibles, alors le client s'assied sur l'une des chaises libres. Si le coiffeur est endormi, le client le réveille.

Écrivez un programme pour coordonner le coiffeur et les clients.

1. Démontrez que les moniteurs et les sémaphores sont équivalents dans la mesure où ils peuvent être utilisés pour implémenter les mêmes types de problèmes de synchronisation.

**Réponse :**

Un sémaphore peut être implémenté en utilisant le code moniteur suivant :



1. Écrivez un moniteur qui implémente un réveil permettant à un programme appelant de se retarder pendant un nombre spécifié d'unités de temps (ticks). Vous pouvez supposer l'existence d'une horloge matérielle réelle qui invoque une procédure tick dans votre moniteur à intervalles réguliers.

**Réponse :**



1. Soient trois processus concurrents P1, P2 et P3 qui partagent les variables *n* et out. Pour contrôler les accès aux variables partagées, un programmeur propose les codes suivants :

*Semaphore mutex1 = 1 ;*

*Semaphore mutex2 = 1 ;*

***Code du processus p1 :***

*P(mutex1) ;*

*P(mutex2) ;*

*out=out+1 ;*

*n=n-1 ;*

*V(mutex2) ;*

*V(mutex1) ;*

***Code du processus p2 :***

*P(mutex2) ;*

*out=out-1 ;*

*V(mutex2) ;*

***Code du processus p3 :***

*P(mutex1) ;*

*n=n+1 ;*

*V(mutex1) ;*

Cette proposition est-elle correcte ? Sinon, indiquer parmi les 4 conditions requises pour réaliser une exclusion mutuelle correcte, celles qui ne sont pas satisfaites ?

Proposer une solution correcte.

**Réponse :**

Non, car si P2 est en section critique et P1 a exécuté P(mutex1) alors P1 est bloqué et empêche P3 d’entrer en section critique.

Conditions non vérifiées : Un processus en dehors de sa section critique bloque un autre processus.

*Processus P1 P(mutex1) ;*

*n=n-1 ;*

*V(mutex1) ;*

*P(mutex2) ;*

*out = out +1 ;*

*V(mutex2) ;*

1. On veut effectuer en parallèle le produit de deux matrices A et B d’ordre n (nxn). Pour ce faire, on crée m (m<n) processus légers (threads). Chaque processus léger se charge de calculer quelques lignes de la matrice résultat R :

Pour j = 0 à n-1 R[i,j] = ¦ k=0,n-1 A[i,k]\*B[k,j] ;

Donner, sous forme de commentaires (en utilisant les sémaphores et les opérations P et V), le code des processus légers : CalculLignes ( ). Préciser les sémaphores utilisés et les variables partagées.

**Réponse :**

On va utiliser un vecteur T de n booléens. T[i] est 0 si le calcul de la ligne i n’est pas encore entamée. T[i] est égal à 1 sinon. Ici, on n’a pas besoin de sémaphores pour les accès aux matrices (en lecture uniquement). Par contre, on a besoin d’un sémaphore binaire mutex pour contrôler les accès au vecteur T. Initialement, tous les éléments de T sont nuls.

*fonction CalculLignes ( )*

*{*

*pour i = 0 à n-1 pas 1 faire*

*P(mutex) si ( T[i]==0)*

*{*

*T[i] = 1 ;*

*V(mutex) ;*

*Pour j = 1 à n pas 1*

*faire*

*Pour k=1 à n pas 1*

*faire R[i,j] += A[i,k] \* B[k,j]*

*fait*

*fait*

*} else V(mutex) ;*

*fait*

*}*