Contrôle terminal Session 2 - 10/6/2021

Durée : 2h
Cours et/ou travaux dirigés autorisés.

Barème donné à titre indicatif : Ex.1 : 5 - Ex. 2 : 2 - Ex. 3 : 6 - Ex.4 : 7

Exercice 1. On considère les processus suivants définis par leur durée d'exécution (réelle ou estimée), et leur date d'arrivée :

Processus	durée	date
p_1	10	0
p_2	7	3
p_3	2	3
p_4	8	5
p_5	2	8

1. Dessiner un diagramme de Gantt correspondant au résultat d'un ordonnancement préemptif plus court d'abord, avec remise en fin de file. À chaque date d'arrivée, indiquer l'état de la file d'attente des processus, sachant qu'un processus qui arrive est mis dans la file avant celui qui est interrompu.
   Indiquer le temps d'attente moyen.
2. Dessiner un diagramme de Gantt correspondant au résultat d'un ordonnancement avec tourniquet et un quantum de temps fixé à 3. À chaque fin du quantum, indiquer l'état de la file d'attente des processus, sachant qu'un processus qui arrive est prioritaire par rapport à celui qui est interrompu.
   Indiquer le temps d'attente moyen.
3. Quel est le meilleur algorithme suivant le critère du temps d'attente moyen ?

Exercice 2. On considère un espace d'adresses logiques de 64 pages de 256 octets chacune, représenté dans une mémoire physique de 128 cadres de pages.

a) Combien de bits comporte l'adresse logique ?
b) Combien de bits comporte l'adresse physique ?

Exercice 3. On suppose un système de 4096 Ko de mémoire haute organisée avec des pages de 32 Ko et un seul niveau de pagination.

a) Décrire le système d'adressage logique. Quelle est la taille maximale de la table des pages ?
b) On suppose que dans ce système, on a trois processus qui s'exécutent : P_1 nécessitant 1250 Ko (code, données et pile), P_2 nécessitant 100 Ko et P_3 nécessitant 200 Ko.
   1. Quelle est la quantité de mémoire réellement utilisée par l'exécution de ces trois processus ?
   2. Quel est le taux de fragmentation ? Expliquer.
c) En considérant les huit premières entrées de la table des pages présentée par le schéma suivant :

numéro de cadre de page	numéro de page	Bit de présence/absence
7	0	0
6	0	0
5	0	1
4	2	1
3	0	0
2	1	1
1	0	0
0	3	1

et en supposant une taille de cadre de page de 32 Ko, donner les adresses logiques correspondantes aux adresses physiques 31792 et 90348, en expliquant clairement votre calcul.

Exercice 4. On veut simuler le fonctionnement d'une piscine municipale. Le comportement des baigneurs est modélisé au moyen de processus dont le nombre est aléatoire ainsi que le moment de leur lancement. Chaque processus exécute la fonction suivante puis disparaît :

```
1.  Fonction baigneur () :
2.     Si (payer_au_guichet() = -1) Alors
3.        retourner         // c'est complet!
       Fsi
4.     bracelet <- trouver_casier()
5.
6.     se_changer_puis_se_baigner()
7.
8.     liberer_casier (bracelet)
9.  Fin
```

Lorsqu'un baigneur arrive, il passe d'abord par un guichet pour payer le droit d'entrée (ligne 3). Une fois à l'intérieur de l'enceinte, le baigneur doit d'abord trouver un casier libre pour en récupérer la clé (ligne 5), puis il peut y déposer ses affaires et se baigner (ligne 7). En fin de baignade, il se change et libère son casier (ligne 9).

Dans un premier temps, on s'intéresse à la gestion des casiers. On suppose que le paiement au guichet ne garantit pas à un baigneur de trouver immédiatement un casier libre pour se changer [1]. Le code ci-dessous constitue une première tentative d'implantation de la fonction `trouver_casier`. On peut remarquer que si le nombre de baigneurs qui tentent d'obtenir un casier est supérieur à $MAX\_CASIERS$, certains sont obligés de patienter jusqu'à ce qu'un casier se libère...

```
boolean casiers [MAX_CASIERS] = { FAUX, ..., FAUX }; // FAUX = LIBRE
// retourne le numéro du casier obtenu
(sorte de "bracelet" que le baigneur portera au poignet)
3.  Fonction trouver_casier () retourne entier :
4.     TantQue(VRAI) Faire
5.          Pour i de 0 à MAX_CASIERS - 1 Faire
6.              Si (casiers [i] = FAUX) Alors
7.                   casiers [i] <- VRAI;
8.                   retourner i
                Fsi
            FinPour
        FinTantQue
9.  Fin

10.  Fonction liberer_casier (int num) :
11.       casiers [num] <- FAUX
      Fin
```

1. Montrez que cette implantation de la fonction `trouver_casier` risque d'aboutir à des erreurs.
2. Corriger le code en introduisant des moniteurs/conditions (et sans doute d'autres variables) partagées. Proposer une solution qui évite l'utilisation de boucles d'attente active.
   Préciser les valeurs initiales.
   NB : Une variable comptant le nombre de casiers libres facilitera les choses.

1. On peut considérer cela comme une forme de surbooking.

3. Voici une version préliminaire de la fonction `payer_au_guichet` :

```
entier reste <- CAPACITE_MAX        // nombre de places disponible à la vente
Fonction payer_au_guichet() retourne entier :
   Si (reste = 0) Alors
       retourner -1
   Fsi
   reste <- reste - 1

   delai_paiement()
   retourner 0     // tout va bien
Fin
```

   Donner une version de `payer_au_guichet` synchronisée avec des moniteurs de manière à ce que les baigneurs obtiennent leur ticket tour à tour. Noter qu'une file d'attente implicite va se créer en raison du temps de paiement de chaque baigneur. Néanmoins, dès que la capacité maximale de la piscine est atteinte, on veut que la fonction retourne immédiatement -1.