$\frac{\quad}{15} + \frac{\quad}{25} + \frac{\quad}{20} + \frac{\quad}{20} + \frac{\quad}{20} = \frac{\quad}{100}$

Nom: _____ Code permanent: _____

FACULTÉ DES ARTS ET DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE

TITRE DU COURS: **Systèmes d'exploitation**
SIGLE DU COURS: **IFT2240**

NOM DU PROFESSEUR: Marc Feeley

DATE DE L'EXAMEN INTRA H06: 6 mars 2006 HEURE: 15h30-17h20 SALLE: PAA 1140

DIRECTIVES PÉDAGOGIQUES:
- Aucune documentation permise.
- Répondre directement sur le questionnaire dans l'espace réservé.
- Soyez clair et concis.

Numéro 1 – (15 points). Quelle est la différence principale entre les processus lourds ("process") et les processus légers ("threads")? Quel est l'avantage respectif de chaque type de processus?

1. *Différence principale:*

2. *Avantage des processus lourds:*

3. *Avantage des processus légers:*

Numéro 2 – (25 points). Supposez un système d'exploitation qui n'offre que les sémaphores binaires comme mécanisme de synchronisation, c'est-à-dire un type "`mutex`" et les prototypes C++ suivants pour initialiser, acquérir et céder ces sémaphores binaires:

```
void init (mutex& m);   // initialise la sémaphore binaire m
void lock (mutex& m);   // acquérir la sémaphore binaire m
void unlock (mutex& m); // céder la sémaphore binaire m
```

1. Comment peut-on implanter les sémaphores générales uniquement avec les sémaphores binaires? Pour le montrer, complétez la définition du type "`semaphore`" ci-dessous et les fonctions pour initialiser, acquérir et céder ces sémaphores générales.

```
typedef struct {



  } semaphore;

void sema_init (semaphore& s, int n) {



}

void sema_acquerir (semaphore& s) {









}

void sema_ceder (semaphore& s) {









}
```

2. Avec seulement les sémaphores binaires, et sans attente active, lesquels des cinq mécanismes de synchronisation suivants peuvent être implantés? (encerclez le nom de ceux qu'on peut implanter)

   (1) **moniteur** (2) **variable de condition** (3) **rendez-vous**

   (4) **barrière à N processus** (5) **sémaphores binaires récursives**

Numéro 3 – (20 points). Sur UNIX, l'exécutable du shell "`sh`" se trouve dans "`/bin/sh`". Lorsque "`/bin/sh`" est exécuté sans arguments de ligne de commande il est en mode interactif et lit des commandes du terminal. Cependant lorsqu'il est exécuté avec l'option "`-c`" suivi d'une commande, le shell exécute cette commande et termine son exécution. Par exemple

```
% ls *.pdf
tp1.pdf  tp2.pdf  tp3.pdf
% /bin/sh -c "ls *.pdf"
tp1.pdf  tp2.pdf  tp3.pdf
%
```

Cette information sera utile pour résoudre le problème suivant. On aimerait avoir une fonction en C/C++ dont le prototype est

```
void commande (char* cmd);
```

Cette fonction doit exécuter la commande indiquée par `cmd` comme si on l'avait tappée au shell interactif, et la fonction retourne seulement après la fin de l'exécution de la commande. Par exemple, il faut que l'appel de fonction

```
commande ("ls *.pdf");
```

affiche la liste des fichiers comme ci-dessus. Écrivez la fonction "`commande`". Vous pouvez ignorer l'inclusion des fichiers d'entête.

Numéro 4 – (20 points). Les questions suivantes ont rapport au premier travail pratique. Vous devez expliquer le fonctionnement du clavier et de la souris d'un PC. Quelles sont les informations qui sont envoyées par le clavier lorsqu'on enfonce et relâche une touche? Quelles sont les informations qui sont envoyées par la souris PS/2 lorsqu'on déplace la souris? Comment le CPU est-il informé d'un évennement de clavier et de souris (donnez les étapes dans la propagation des informations)? Soyez précis (nombre d'octets, contenu des octets, etc) sans dépasser l'espace réservé pour la réponse.

1. *Quelles sont les informations envoyées par le clavier?*

2. *Quelles sont les informations envoyées par la souris?*

3. *Comment le CPU est-il informé d'un évennement de clavier et de souris?*

Numéro 5 – (20 points). Cochez la bonne case pour indiquer si les énoncés suivants sont vrais ou faux (note: une bonne réponse compte pour 2 points et une mauvaise réponse pour -1 point).

		vrai	*faux*
1)	Dans un processus UNIX la section "text" contient les arguments de ligne de commande du processus.	☐	☐
2)	Pour qu'il y ait une étreinte fatale, le graphe d'allocation de ressources doit contenir un cycle.	☐	☐
3)	"`login`" est le premier processus créé par le kernel UNIX.	☐	☐
4)	Lorsque le parent d'un processus P meurt avant P, le processus P est un zombie.	☐	☐
5)	Si deux processus exécutent respectivement n et m opérations élémentaires, le nombre total d'exécutions possibles pour la combinaison de ces deux processus est égal à $(n! + m!)/(n! * m!)$.	☐	☐
6)	L'algorithme de Peterson d'exclusion mutuelle entre deux processus exige seulement que la lecture et l'écriture d'un bit de mémoire soient des opérations atomiques.	☐	☐
7)	L'algorithme d'ordonnancement "plus petite tâche en premier" est optimal du point de vue du temps d'attente moyen.	☐	☐
8)	Lorsqu'un système d'exploitation possède seulement 2 niveaux de priorité, le problème d'inversion de priorité est impossible.	☐	☐
9)	Le signal "SIGWINCH" est un signal asynchrone.	☐	☐
10)	Si tous les processus qui partagent des ressources font l'acquisition de ces ressources dans un ordre prédéterminé on élimine la possibilité d'étreinte fatale reliée au partage des ressources.	☐	☐