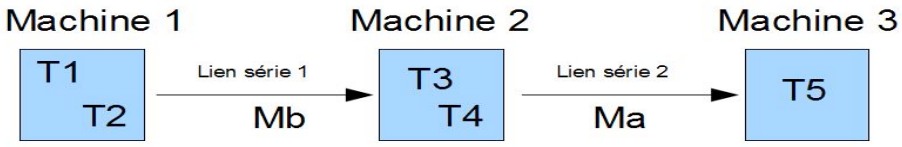
Année 2007-2008

INF342 - COU
Contrôle de Connaissances
29 juin 2010
3h00 - Sans document – Barème indicatif

Veuillez répondre à cette partie sur une feuille séparée

I. Ordonnancement (5 points)

Il s'agit de vérifier le bon fonctionnement d'une chaîne de production industrielle.



On vous demande d'assister un ingénieur qui doit analyser une chaîne de production de casseroles. Cette chaine est constituée de trois tâches, hébergées sur 3 machines (cf. figure ci-dessus):

1. Dans la première machine (machine 1), la tâche T1 produit par emboutissage les fonds de casseroles.
2. Puis, la tâche T3 de la seconde machine (machine 2) soude les manches aux fonds de casseroles.
3. Enfin, la dernière machine (machine 3) emballe le produit terminé. Ce dernier traitement est effectué par la tâche T5.

Sur les machines hébergeant T1 et T3, d'autres tâches existent, mais elles ne participent pas à la fabrication des casseroles. Enfin les trois machines sont connectées par deux liens séries. Les liens séries ne sont pas partagés, ainsi:

1. Le lien série entre la machine 1 et la machine 2 est uniquement destiné à transmettre le message Mb. Ce message est expédié lorsque T1 termine son exécution afin d'avertir la tâche T3 qu'elle peut commencer son exécution.
2. Le lien série entre la machine 2 et la machine 3 est uniquement destiné à transmettre le message Ma. Ce message est expédié lorsque T3 termine son exécution afin d'avertir la tâche T5 qu'elle peut commencer son exécution.

L'ingénieur souhaite connaître le pire temps de fabrication d'une casserole: c'est à dire le délai entre le réveil de la tâche T1 et la terminaison de la tâche T5. On vous demande de l'aider.

Question I-1

Il est possible d'utiliser l'approche holistique de Tindell pour déterminer le pire temps de fabrication des casseroles. Expliquez à l'ingénieur comment fonctionne la méthode holistique.

Question I-2

Vous avez effectué quelques mesures sur la chaîne de production et vous en avez déduit que:

1. Les messages Ma ont un temps de communication inférieur à 1 seconde. Il en est de même pour Mb. Ces temps de communication incluent les temps de transmission sur le lien série, les temps de propagation ainsi que les temps de traversée des couches logicielles et matérielles.
2. Les tâches sont caractérisées par le tableau ci-dessous. Notez que la priorité 1 est le niveau de priorité le plus fort et 2 le niveau de priorité le plus faible.

Tâches	Priorité	Capacités (en seconde)	Périodes (en seconde)
T1	1	12	500
T2	2	4	500
T3	2	6	500
T4	1	20	500
T5	1	100	500

Appliquez l'approche holistique au jeu de tâches ci-dessus afin de calculer le pire temps de production d'une casserole, c'est à dire le pire temps de réponse de T5.

Question I-3

Après votre analyse, l'ingénieur effectue quelques mesures sur la chaîne de production. Il s'aperçoit vite que le pire temps de production d'une casserole calculé par l'approche holistique est très pessimiste. En pratique, le temps de production d'une casserole semble bien plus petit.

C'est un défaut connu de la méthode holistique (et de la théorie de l'ordonnancement temps réel en général). Expliquez d'où cela provient.

Question I-4

Une autre solution classique pour relier les trois machines consiste à les connecter par un bus de terrain. Le bus est alors partagé par les messages Ma et Mb. Quelle est l'implication de ce changement d'architecture sur l'analyse holistique effectuée dans la question 2.

Veuillez répondre à cette partie sur une feuille séparée

II Ordonnancement de tâches Ada périodiques et apériodiques (3 points)

$$3*(2^{1/3} - 1) = 0,779$$

$$4*(2^{1/4} - 1) = 0,756$$

```
task type A_Task (P : Priority; C : Natural; D : Natural) is
end A_Task;
procedure Consume (D : Duration) is - -  Compute during D seconds
T0 : constant Time := Clock;

task body A_Task is
   T : Time := T0 + Duration (D);
begin
   loop
      Consume (Duration (C));
      delay T – Clock ;
      T := T + Duration (D);
   end loop;
end A_Task;

T1 : A_Task (P => Default_Priority + 3, C => 1, D => 10);
T2 : A_Task (P => Default_Priority + 2, C => 2, D => 5);
T3 : A_Task (P => Default_Priority + 1, C => 3, D => 15);
```

Question II-1
En quoi consiste le code ci-dessus ? (5 lignes au plus) Compléter la spécification des tâches de type A_Task pour que la tâche s'exécute à la priorité de son attribut P.

Question II-2
Expliquer et corriger l'erreur de programmation introduite dans l'implantation de A_Task.

Question II-3
Pourquoi la configuration des priorités des tâches est-elle incorrecte dans le contexte de RMS ? Corriger la configuration puis déterminer si celle-ci est ordonnançable.

Question II-4

On ajoute à cette configuration deux tâches apériodiques.

	Calcul	Activation
Tâche 4	3	7
Tâche 5	4	11

On va traiter ces tâches apériodiques à l'aide d'un serveur sporadique On considérera une capacité du serveur de 1 et une période de 4. Déterminer l'ordonnançabilité de cette nouvelle configuration selon RMS.

Question II-5

Décrire graphiquement l'exécution des tâches dans ces quatre cas et commenter le déroulement. Il sera particulièrement tenu compte des indispensables explications décrivant le fonctionnement du serveur sporadique.

Veuillez répondre à cette partie sur une feuille séparée

III. Bus et Réseaux Temps Réel (1 point)

Question III-1

Soit 3 sites, S1, S2 et S3 interconnectés par un bus de type CAN, leurs priorités sont telles que Priorité (S1) > Priorité (S2) > Priorité (S3). La priorité de S1 est 8.
Comment numéroter les priorités de S2 et S3 de façon à ce que ces sites prennent le bus en fonction de leur priorité ? Exemple de fonctionnement pour l'accès au bus.

Question III-2

Pourquoi le protocole CSMA/CD n'est-il pas adapté au temps réel ?

IV. Java Temps Réel (1 point)

Question IV-1

Comment le ramasse-miettes (*garbage collector*) et un thread temps réel de type `NoHeapRealTime` peuvent-ils se trouver impliqués dans un schéma d'inversion de priorité sur une JVM RTSJ ? Donner un exemple commenté.

Question IV-2

Que propose l'interface `Scheduler` pour le contrôle de l'ordonnancement ?

V. Noyaux Temps Réel (1 point)

Question V-1

L'édition de liens, la pagination peuvent chacun introduire un facteur d'indéterminisme. Dans chaque cas : expliquer pourquoi et indiquer comment éviter l'introduction de ce facteur.

Question V-2

Décrivez succinctement comment implémenter un lot de tâches ordonnancées selon une politique RMS sous le système d'exploitation RTEMS

Veuillez répondre à cette partie sur une feuille séparée

VI. Tolérance aux Fautes (2 points)

Question VI-1

Définir les concepts de Fiabilité et Disponibilité. Illustrer ces deux concepts sur l'exemple d'un serveur web de transactions boursières qui propose une opération d'achat `buy(n).` Donner un exemple de défaut de fiabilité et un exemple de défaut de disponibilité.

Question VI-2

Définir composant autotestable, recouvrement arrière et recouvrement avant.

VII. Tolérance aux Fautes et RT-POSIX (3 points)

On souhaite implémenter un système constitué de deux tâches temps réel de priorités respectives p1 et p2, avec p1 > p2 (la tâche 1 est plus prioritaire que la tâche 2).
Les deux tâches sont implémentées par des threads POSIX et communiquent en utilisant un tableau de caractères de taille fixe servant de tampon de communication. On va utiliser les primitives `pthread_mutex_lock et pthread_mutex_unlock` pour assurer l'exclusion mutuelle sur les accès en lecture ou écriture à cette variable (le tampon).

Question VII-1

Comment s'assurer que la politique d'ordonnancement sera toujours cohérente avec les priorités définies ci-dessus ?

Note :
La tâche la plus prioritaire peut travailler sur l'ancienne valeur de la variable partagée à condition que le décalage n'excède pas 2 périodes de la tâche la plus prioritaire. L'accès à la variable se fait une seule fois par période à travers l'appel à une fonction `get_data()` qui retourne une copie de la valeur du tampon. Le comportement de la tâche t1 peut être modélisé par trois appels de fonction successifs :

```
{    traitement1() ; t=get_data() ;  traitement2(t) ;  }
```

La fonction `pthread_mutex_trylock()` permet de prendre un verrou, si ce dernier est libre, sans pour autant suspendre l'application si ce verrou est déjà pris.

Question VII-2

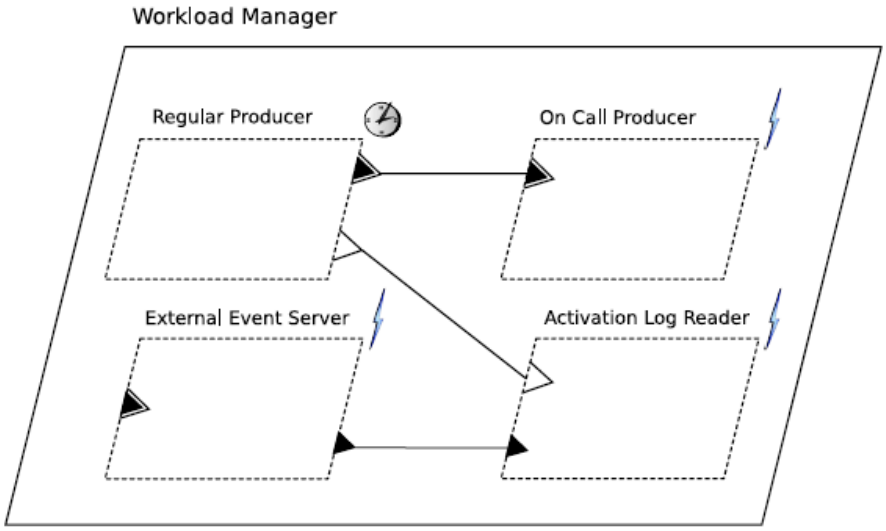
Consulter la documentation de la fonction (voir l'annexe en fin de document) et expliquer comment cette fonction pourrait être utilisée pour implémenter un recouvrement avant dans la tâche t1 en utilisant une variable auxiliaire. ce recouvrement avant permet de traiter le cas où le verrou est détenu par la tâche 2 : on veut on veut pas que t1 attende.

Question VII-3

Expliquez l'intérêt de cette démarche du point de vue du pire temps de réponse pour la tâche la plus prioritaire ? Est-ce que cela résout le problème de l'inversion de priorité ?

Veuillez répondre à cette partie sur une feuille séparée

VIII. Langage AADL (4 points) :



VIII.1 Historique

On considère la représentation AADL ci-dessus : cette représentation illustre un exemple issu du document présentant les recommandations pour le profil Ravenscar[1] pour la construction d'application temps réels.

VIII.2 Présentation

L'exemple que nous avons choisi décrit un système de gestion de charge de travail. Le processus *Workload Manager* gère des charges de travail variables. Le système reçoit interruptions venant de l'extérieur. Ces interruptions sont reçues par un processus léger de très haute priorité et sont enregistrées dans un tampon spécifique. Le traitement de ces interruptions est délégué à un processus léger sporadique de très faible priorité qui est réveillé de temps en temps par le processus périodique principal.

L'exemple décrit les entités suivantes :

1. Quatre processus légers que nous donnons ci après par ordre décroisant de priorité :
   a. Un processus léger sporadique (*External Event Server*) effectue la réception des interruptions extérieures et leur enregistrement dans un tampons spécifique.
   b. Un processus léger périodique (*Regular Producer*) effectue la charge de travail régulière. Il délègue, sous des conditions spécifiques, la charge de travail supplémentaire et le traitement des interruptions extérieures à d'autres processus légers,
   c. Un processus léger sporadique (*On Call Producer*) effectue la charge de travail supplémentaire.
   d. Un processus de travail sporadique (*Activation Log Reader*) effectue une quantité de travail correspondant au traitement de la dernière interruption reçue.

2. Trois données partagées et protégées :
   a. Un tampon (*Request Buffer*) reçoit les ordres de travaux supplémentaires. Il est rempli par *Regular Producer* et consulté par *On Call Producer*,
   b. Une file d'attente (*Event Queue*) des interruptions extérieures. Elle est remplie par le périphérique émettant les interruptions et consultée par *External Event Server*,
   c. Un journal (*Activation Log*) des interruptions à traiter. Il est rempli par *External Event*

[1] Burns et al. 2004.

Server et consulté par *Activation Log Reader*.

3. Une entité passive (*ProductionWorkload*) abrite l'opération qui traite les travaux demandés par un des processus légers.

Nom	Protocole	Période	Echéance	WCET	Priorité
Regular_Producer	Périodique	1000 ms	500 ms	498 ms	7
On_Call_Producer	Sporadique	1000 ms	800 ms	250 ms	5
Activation_Log_Reader	Sporadique	1000 ms	1000 ms	125 ms	3
External_Event_Server	Sporadique	5000 ms	100 ms	2 ms	11

Tableau 1 - Caractéristiques des processus légers

Question VIII-1

Vous devez compléter la description AADL ci-dessous. Les processus légers peuvent être décrits à l'aide des composants threads AADL. On rappelle que les propriétés AADL permettent de spécifier les caractéristiques d'un composant thread (type, période, deadline, wcet,…).

Compléter la spécification des composants thread AADL représentant les processus légers.

Question VIII-2

Donner la spécification AADL du processus *Workload Manager*. Vous n'oublierez pas de compléter la section *connections*.

Question VIII-3

A quoi sert l'outil *Cheddar* ?

On rappelle que pour qu'un modèle AADL soit analysable par l'outil *Cheddar*, il est obligatoire d'utiliser les propriétés AADL définies pour l'outil *Cheddar*. A l'aide de la propriété *Fixed_Priority* définie dans le property set *Cheddar_Properties* compléter la description des composants thread AADL.

Question VIII-4

Expliquer brièvement pourquoi il n'a pas été utile de spécifier explicitement les trois données partagées et protégées en AADL.

```
data Workload
properties
  Data_Model::Data_Representation => Integer;
end Workload;

data Interrupt_Counter
properties
  Data_Model::Data_Representation => Integer;
end Interrupt_Counter;

thread Regular_Producer
features
  Additional_Workload : out event data port Workload;              --  réveille On_Call_Producer
  Handle_External_Interrupt  : out event port;                     --  réveille Activation_Log_reader

properties
  Compute_Entrypoint                 => "Work.Regular_Producer_Operation";
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;

end Regular_Producer;

thread On_Call_Producer
features
  Additional_Workload_Depository : in event data port Workload     --
    {Queue_Size          => 5;
     Compute_Entrypoint => "Work.On_Call_Producer_Operation";};
  --  Pour recevoir la charge de travail supplémentaire de la part de
  --  Reagular_Producer. Cette taille est donnée par le guide
  --  Ravenscar.

properties
  Dispatch_Protocol                  => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
end On_Call_Producer;

thread External_Event_Server
features
  External_Interrupt_Depository : in event data port Interrupt_Counter
   {Queue_Size          => 1;
    Compute_Entrypoint => "Events.Delegate_External_Event";};
  --  Délivrée par le processus lourd

  External_Interrupt : out data port Interrupt_Counter;
  --  Vers Activation_Log_Reader

properties
  Compute_Execution_Time             => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
end External_Event_Server;

thread Activation_Log_Reader
features
  External_Interrupt_Depository : in data port Interrupt_Counter;
  --  Enregistre la dernière interruption reçue

  Signal : in event port
  {Queue_Size          => 1;
   Compute_Entrypoint => "Logs.On_Signal";};
  --  Reçoit les ordre de réveil de la part de Regular_Producer pour
  --  traiter la dernière interruption.

properties
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
end Activation_Log_Reader;
```

```
thread External_Event_Source
features
  External_Interrupt : out event data port Interrupt_Counter;
properties
  Initialize_Entrypoint              => "Event_Source.Init";
  Compute_Entrypoint                 => "Event_Source.New_External_Event";
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
  ……                                 => ……;
end External_Event_Source;

process Workload_Manager
features
  External_Interrupt_Depository : in event data port Interrupt_Counter;
end Workload_Manager;

process implementation Workload_Manager.Impl
subcomponents
  ……: ……;
  ……: ……;
  ……: ……;
  ……: ……;

connections
  event data port External_Interrupt_Depository
     -> ……;

  event data port ……
     -> On_Call_Producer.Additional_Workload_Depository;

  event port Regular_Producer.Handle_External_Interrupt
     -> ……;

  data port External_Event_Server.External_Interrupt
     -> Activation_Log_Reader.External_Interrupt_Depository;

end Workload_Manager.Impl;
```

ANNEXE

Nom

`pthread_mutex_init, pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock, pthread_mutex_destroy`

Synopsis

pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t recmutex = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
pthread_mutex_t errchkmutex = PTHREAD_ERREURCHECK_MUTEX_INITIALIZER_NP;
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex));
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);

Description
Un mutex est un objet d'exclusion mutuelle (MUTual EXclusion device), et est très pratique pour protéger des données partagées de modifications concurrentes et pour implémenter des sections critiques.
Un mutex peut être dans deux états : déverrouillé ou verrouillé (possédé par un thread). Un mutex ne peut être la pris que par un seul thread à la fois. Un thread qui tente de verrouiller un mutex déjà verrouillé est suspendu jusqu'à ce que le mutex soit déverrouillé.
pthread_mutex_init initialise le mutex pointé par mutex selon les attributs de mutex spécifié par mutexattr. Si mutexattr vaut NULL, les paramètres par défaut sont utilisés.

L'implémentation LinuxThreads ne supporte qu'un seul attribut, le type de mutex, qui peut être soit ``rapide'', ``récursif'' ou à ``vérification d'erreur''. Le type de mutex détermine s'il peut être verrouillé plusieurs fois par le même thread. Le type par défaut est ``rapide''. Voir pthread_mutexattr_init(3) pour plus d'informations sur les attributs de mutex.

Les variables de type pthread_mutex_t peuvent aussi être initialisées de manière statique, en utilisant les constantes PTHREAD_MUTEX_INITIALIZER (pour les mutex rapides), PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP (pour les mutex récursifs), et PTHREAD_ERREURCHECK_MUTEX_INITIALIZER_NP (pour les mutex à vérification d'erreuur).

pthread_mutex_lock verrouille le mutex. Si le mutex est déverrouillé, il devient verrouillé et est possédé par le thread appelant; et pthread_mutex_lock rend la main immédiatement. Si le mutex est déjà verrouillé par un autre thread, pthread_mutex_lock suspend le thread appelant jusqu'à ce que le mutex soit déverrouillé.
Si le mutex est déjà verrouillé par le thread appelant, le comportement de pthread_mutex_lock dépend du type de mutex. Si ce dernier est de type ``rapide'', le thread appelant est suspendu jusqu'à ce que le mutex soit déverrouillé, donc plaçant le thread appelant en situation de deadlock. Si le mutex est de type ``à vérification d'erreur'', pthread_mutex_lock rend la main immédiatement avec le code d'erreur EDEADLK. Si le mutex est de type ``récursif'', pthread_mutex_lock rend la main immédiatement avec un code de retour indiquant le succès, enregistrant le nombre de fois où le thread appelant a verrouillé le mutex. Un nombre égal d'appel à pthread_mutex_unlock doit être réalisé avant que le mutex retourne à l'état déverrouillé.

pthread_mutex_trylock se comporte de la même manière que pthread_mutex_lock, excepté qu'elle ne bloque pas le thread appelant si le mutex est déjà verrouillé par un autre thread (ou par le thread appelant dans le cas d'un mutex ``rapide''). Au contraire, pthread_mutex_trylock rend la main immédiatement avec le code d'erreur EBUSY. Les autres codes d'erreur sont EINVAL lorsque la valeur de la référence du mutex est invalide.

pthread_mutex_unlock déverrouille le mutex. Celui-ci est supposé verrouillé, et ce par le thread courant en entrant dans pthread_mutex_unlock. Si le mutex est de type ``rapide'', pthread_mutex_unlock le réinitialise toujours à l'état déverrouillé. S'il est de type ``récursif'', son

compteur de verrouillage est décrémenté (nombre d'opérations pthread_mutex_lock réalisées sur le mutex par le thread appelant), et déverrouillé seulement quand ce compteur atteint 0.
Sur les mutex ``vérification d'erreur'', pthread_mutex_unlock vérifie lors de l'exécution que le mutex est vérrouillé en entrant, et qu'il est verrouillé par le même thread que celui appelant pthread_mutex_unlock. Si ces conditions ne sont pas réunies, un code d'erreur est renvoyé et le mutex n'est pas modifié. Les mutes ``rapide'' et ``récursif'' ne réalisent pas de tels tests, permettant à un mutex verrouillé d'être déverrouillé par un thread autre que celui l'ayant verrouillé. Ce comportement n'est pas portable et l'on ne doit pas compter dessus.

pthread_mutex_destroy détruit un mutex, libérant les ressources qu'il détient. Le mutex doit être déverrouillé. Dans l'implémentation LinuxThreads des threads POSIX, aucune ressource ne peut êtire associé à un mutex, aussi pthread_mutex_destroy ne fait en fait rien si ce n'est vérifier que le mutex n'est pas verrouillé.

Annulation
Aucune des primitives relatives aux mutex n'est un point d'annulation, ni même pthread_mutex_lock, malgré le fait qu'il peut suspendre l'exécution du thread pour une longue durée. De cette manière, le statut des mutex aux points d'annulation est prévisible, permettant aux gestionnaires d'annulation de déverrouiller précisément ces mutex qui nécessitent d'être déverrouillés avant que l'exécution du thread ne s'arrête définitivement. Aussi, les threads travaillant en mode d'annulation retardée ne doivent-ils jamais verrouiller un mutex pour de longues périodes de temps.

Fiabilite Par Rapport Aux Signaux Asynchrones
Les fonctions relatives aux mutex ne sont pas fiables par rapport aux signaux asynchrones et ne doivent donc pas être utilisées dans des gestionnaires de signaux [Ndt: sous peine de perdre leur propriété d'atomicité]. En particulier, appeler pthread_mutex_lock ou pthread_mutex_unlock dans un gestionnaire de signal peut placer le thread appelant dans une situation de deadlock [Ndt: exclusion mutuelle avec lui-même !!].

Valeur Renvoyée
pthread_mutex_init retourne toujours 0. Les autres fonctions renvoient 0 en cas succès et un code d'erreur non nul en cas de problème.

Erreurs
La fonction **pthread_mutex_lock** renvoie l'un des codes d'erreur suivants en cas de problème:
EINVAL : le mutex n'a pas été initialisé.
EDEADLK : le mutex est déjà verrouillé par un thread autre que l'appelant (mutex à vérification d'erreur seulement).

La fonction **pthread_mutex_trylock** renvoie l'un des codes d'erreur suivants en cas de problème:
EBUSY : le mutex ne peut être verrouillé car il l'est déjà.
EINVAL : le mutex n'a pas été initialisé.

La fonction **pthread_mutex_unlock** renvoie le code d'erreur suivant en cas de problème:
EINVAL : le mutex n'a pas été initialisé.
EPERM : le thread appelant ne possède pas le mutex (mutex à vérification d'erreur seulement).

La fonction **pthread_mutex_destroy** renvoie le code d'erreur suivant en cas de problème:
EBUSY : le mutex