

TP Temps Réel

Jérôme Pouiller <j.pouiller@sysmic.org>

Polytech' Paris - Mars 2012

Table des matières

1	Avant de commencer	1
1.1	Documentation	1
1.1.1	Références des pages de man	1
1.1.2	Standards	1
1.2	Erreurs	2
1.3	Compilation	2
2	Ordonnancement	2
2.1	Rate Monotonic	2
2.2	Earliest Deadline First	3
2.3	Ordonnancement conjoint	3
2.4	Serveurs de tâches asynchrones	4
2.5	Sporadic Server	4
2.6	Serveurs différé et sporadique	5
2.7	Partage de ressources - Inversions de priorités	5
2.8	Partage de ressources - Dead Locks	6
2.9	Inversion de priorité - 1	6
2.10	Inversion de priorité - 2	7
2.11	Inversion de priorité - 3	7
2.12	Contraintes de précédence	8
3	Watch file	8
3.1	Par scrutation	8
3.2	Par notification	9
4	Protection des structures de données	9
5	Tricky conditions	11
6	Structures de données et mutex	12
7	Protection de ressources	13
8	Threads et mutex	14
9	Tâches et sémaphores Xenomai	15

10 Implémentation d'ordonnanceurs	16
10.1 Algorithmes d'ordonnancement de tâches indépendantes	16
10.2 Algorithmes d'ordonnancement de tâches apériodiques	16
10.3 Ordonnancement et ressources partagées	17
11 Libtimer	17
11.1 Version synchrone	20
11.2 Version asynchrone	20
12 Calculs et temps réel	20
12.1 Introduction	20
12.2 Écriture du Watchdog	21
12.3 Ecriture de l'ISR	21
A Instructions d'installation de la machine virtuelle Xenomai	22
A.1 Mise en service de l'image	22
A.2 L'environnement	22
A.2.1 Se connecter	22
A.2.2 Sauver son travail	22
A.2.3 Accéder à internet	22
A.2.4 Travailler avec emacs, geany, etc...	23
B Écriture des programmes Xenomai	23
B.1 Documentation de Xenomai	23
B.2 Compilation	23
B.3 Verrouillage de la mémoire	23
B.4 A propos des noms des objets Xenomai	23
B.5 Gestion des Erreurs	24
B.6 Exécution	24
C Écriture des modules noyau	24
C.1 Compilation	24
C.2 Exécution	24
C.3 Log Buffer	24
D Modalités de rendu	25
E Modalités de rendu	25

1 Avant de commencer

Cette section est purement informative. Il n'y a pas de questions à l'intérieur.

1.1 Documentation

1.1.1 Références des pages de man

Comme le veut l'usage, les références des pages de man sont données avec le numéro de section entre parenthèses. Ainsi, *wait(2)* signifie que vous pouvez accéder à la documentation avec la commande `man 2 wait`. Si vous omettez le numéro de section, `man` recherchera la première page portant le nom `wait` (ce qui fonctionnera dans 95% des cas). Le numéro de section vous donne aussi une indication sur le type de documentation que vous aller trouver. D'après *man(1)*, voici les différentes sections :

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions (e.g. /etc/passwd)
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. *man(7)*, *groff(7)*
8. System administration commands (usually only for root)
9. Kernel routines [Non standard]

Vous pouvez trouver une version en ligne de la plupart des pages de man sur The Friendly Manual (<http://tfm.cz>).

Vous trouverez en particulier les pages relatives à l'API du noyau Linux qui ne sont pas présentes en local sur vos machine.

Prenez néanmoins garde aux éventuelles différences de versions d'API (très rare et quasiment exclusivement sur les pages relatives à l'ABI du noyau).

1.1.2 Standards

Vous subissez aujourd'hui tout le poids de l'histoire de la norme Posix (30 ans d'histoire de l'informatique). Ne soyez pas étonnés de trouver de multiples interfaces pour une même fonctionnalité. C'est particulièrement vrai sur les fonction relatives au temps.

Nous basons ces exercice sur la norme POSIX.1-2001 que vous pourrez trouver sur <http://www.unix-systems.org/version3/online.html>

Vous trouverez plus d'informations sur l'histoire des différentes normes existantes sur *standards(7)*.

1.2 Erreurs

Pour indiquer quelle erreur s'est produite, la plupart des fonctions Posix utilisent :

- soit leur valeur de retour
- soit une variable globale de type `int` nommée `errno`

Utilisez *strerror(3)* pour obtenir le message d'erreur équivalent au code de retour.

Exemple :

```
1 if (err = pthread_create(&tid, NULL, f, NULL))
    printf("Task create: %s", strerror(err));
if (-1 == pause())
    printf("Pause: %m");
```

1.3 Compilation

Pour la plupart des exercices, vous aurez besoin de linker¹ avec `-lpthread` et `-lrt`.

Par ailleurs, nous vous conseillons fortement de compiler avec l'option `-Wall` de `gcc` et d'utiliser un système de Makefile.

1. C'est à dire la phase de compilation où vous liez tous vos fichiers `.o` ensemble

2 Ordonnancement

2.1 Rate Monotonic

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	29	7	Fin de période
B	0	5	1	Fin de période
C	0	10	2	Fin de période

Question 2.1. Calculez le taux d'utilisation du CPU avec l'algorithme Rate Monotonic. Le jeu de tâche est-il ordonnan cable ?

Question 2.2. Calculez les temps de réponses des différentes tâches.

Question 2.3. Combien de temps devrait-on simuler le système pour avoir tous les ordonnancements possibles.

Question 2.4. Dessinez les 30 premières unités de temps de l'ordonnancement généré par Rate Monotonic en version préemptive puis en non-préemptive. Que constatez-vous ?

Modifions le jeu de tâches :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	30	6	Fin de période
B	0	5	3	Fin de période
C	0	10	2	Fin de période

Dans cet exemple, le jeu de tâche est dit *harmonique* car chaque période est multiple des autres périodes.

Question 2.5. Calculez le taux d'utilisation du CPU avec l'algorithme Rate Monotonic. Le jeu de tâche est-il ordonnan cable ?

Question 2.6. Dessinez les 30 premières unités de temps de l'ordonnancement généré par Rate Monotonic en version préemptive. Que constatez-vous ?

Question 2.7. Calculez les temps de réponse des différentes tâches. Que proposez-vous comme explication ?

2.2 Earliest Deadline First

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	12	5	Fin de période
B	0	6	2	Fin de période
C	0	24	5	Fin de période

Question 2.8. Calculez le taux d'utilisation du CPU avec l'algorithme EDF. Le jeu de tâche est-il ordonnan cable ?

Question 2.9. Déterminez le nombre d'unités de temps libres sur la période.

Question 2.10. Dessinez les 24 premières unités de temps de l'ordonnancement généré par EDF en version préemptive puis en non-préemptive.

Question 2.11. Dessinez les 24 premières unités de temps de l'ordonnancement généré par LST en version préemptive.

2.3 Ordonnancement conjoint

On considère le jeu de tâches suivant :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	12	5	Fin de période
B	0	6	2	Fin de période
C	0	24	5	Fin de période
D	7	-	1	9
E	12	-	3	21

Question 2.12. Calculez si le système est ordonnancable en utilisant l'algorithme Deadline Monotonic.

Question 2.13. Dessinez les 35 premières unités de temps de l'ordonnancement généré par Deadline Monotonic.

Question 2.14. Dessinez les 35 premières unités de temps de l'ordonnancement généré par EDF en version préemptive.

Question 2.15. Mesurez les temps de réponses des tâches aperiodiques.

Question 2.16. Dessinez les 35 premières unités de temps de l'ordonnancement généré par LST.

Question 2.17. Mesurez les temps de réponses des tâches aperiodiques.

2.4 Serveurs de tâches asynchrones

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	12	4	Fin de période
B	0	16	6	Fin de période
S	0	10	2	Fin de période

S est un serveur gérant les tâches asynchrones suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
1	2	-	1	Aucune
2	14	-	4	Aucune
3	28	-	2	Aucune

Question 2.18. Dessinez les 32 premières unités de temps de l'ordonnancement généré par RM (en version préemptive) avec un serveur par scrutation et gestion des tâche asynchrone en fond de tâche.

Question 2.19. Calculez les temps de réponse des différentes tâches.

Question 2.20. Dessinez les 32 premières unités de temps de l'ordonnancement généré par RM (en version préemptive) avec un serveur différé.

Question 2.21. Calculez les temps de réponse des différentes tâche.

Question 2.22. L'utilisation d'un serveur différé dans cette configuration est-elle sûre ?

Question 2.23. Dessinez les 32 premières unités de temps de l'ordonnancement généré par RM (en version préemptive) avec un serveur sporadique et utilisation des temps creux.

Question 2.24. Calculez les temps de réponse des différentes tâche.

2.5 Sporadic Server

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	24	4	Fin de période
B	0	28	8	Fin de période
S	0	20	4	Fin de période

S est un serveur sporadique gérant les tâches asynchrones suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
1	4	-	3	Aucune
2	12	-	4	Aucune

Question 2.25. Dessinez les 30 premières unités de temps de l'ordonnancement généré par RM (en version préemptive).

Question 2.26. Mesurez les temps de réponse des différentes tâche.

2.6 Serveurs différé et sporadique

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	7	2	Fin de période
S	0	6	3	Fin de période

S est un serveur de tâches asynchrones gérant les tâches asynchrones suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
1	21	-	6	Aucune

Question 2.27. Vérifiez que le système est ordonnan cable si on utilise l'algorithme RM

Question 2.28. Calculez les temps de réponse des tâches périodiques.

Question 2.29. Dessinez les 32 premières unités de temps de l'ordonnancement généré par RM avec un serveur différé. Les temps de réponse des tâches périodiques sont-ils respectés ?

Question 2.30. Pouviez-vous prédire ce comportement ?

Question 2.31. Dessinez les 32 premières unités de temps de l'ordonnancement généré par RM et un serveur sporadique. Les temps de réponse des tâches périodiques sont-ils respectés ?

Question 2.32. Mesurez les temps de réponse de la tâche 1.

Question 2.33. Quel devrait être la capacité du serveur pour garantir le temps réel avec un serveur différé ?

Question 2.34. Dessinez les 32 premières unités de temps de l'ordonnancement généré par RM et un serveur différé avec cette nouvelle configuration.

Question 2.35. Mesurez les temps de réponse de la tâche 1.

2.7 Partage de ressources - Inversions de priorités

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Priorité	Sections critiques
A	15	-	10	1	_____11111
B	0	25	10	2	_____
C	10	-	20	3	_1111111111111111_____

Question 2.36. Dessinez les 50 premières unités de temps en utilisant un algorithme d'ordonnancement à priorité fixe (préemptif) sans prendre en compte les sections critiques.

Question 2.37. Dessinez les 50 premières unités de temps en utilisant un algorithme d'ordonnancement à priorité fixe (préemptif).

Question 2.38. Proposez une modification du système de priorité afin de réduire la latence de A. Dessinez les 50 premières unités de temps en utilisant cette nouvelle stratégie d'ordonnancement.

2.8 Partage de ressources - Dead Locks

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Échéance	Sections Critiques
A	0	10	5	Fin de période	$_11\frac{1}{2}2$
B	0	25	8	Fin de période	$_222\frac{1}{2}$

Question 2.39. Dessinez les 18 premières unités de temps en utilisant un ordonnancement EDF et un priority inheritance.

Question 2.40. Dessinez les 18 premières unités de temps en utilisant un ordonnancement EDF et un highest lock (aka Immediate Ceiling Priority Protocol).

Question 2.41. Dessinez les 18 premières unités de temps en utilisant un ordonnancement EDF et un original ceiling priority protocol (OCP).

2.9 Inversion de priorité - 1

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Priorité	Sections critiques
A	4	-	4	1	$_11_$
B	4	-	1	2	$_$
C	2	-	4	3	$_22_$
D	2	-	1	4	$_$
E	0	-	6	5	$_2\frac{1}{2}2_$

Question 2.42. Dessinez les 16 premières unités de temps ordonnancement à priorité statique et sans algorithme de gestion d'inversion de priorité.

Question 2.43. Dessinez les 16 premières unités de temps en utilisant un ordonnancement à priorité statique et l'algorithme d'héritage de priorité.

Question 2.44. Calculez les temps de réponses des différentes tâches en utilisant un ordonnancement à priorité statique et l'algorithme highest lock (aka Immediate Ceiling Priority Protocol).

Question 2.45. Dessinez les 16 premières unités de temps du cas précédent.

Question 2.46. Calculez les temps de réponses des différentes tâches en utilisant un ordonnancement à priorité statique et l'algorithme original ceiling priority protocol (OCP).

Question 2.47. Dessinez les 16 premières unités de temps du cas précédent.

2.10 Inversion de priorité - 2

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Priorité	Sections critiques
A	4	-	4	1	_11_
B	4	-	1	2	_
C	2	-	6	3	_11 $\frac{1}{2}$ 1_
D	2	-	1	4	_
E	0	-	4	5	_22_

Question 2.48. Dessinez les 16 premières unités de temps ordonnancement à priorité statique et sans algorithme de gestion d'inversion de priorité.

Question 2.49. Dessinez les 16 premières unités de temps en utilisant un ordonnancement à priorité statique et l'algorithme d'héritage de priorité.

Question 2.50. Calculez les temps de réponses des différentes tâches en utilisant un ordonnancement à priorité statique et l'algorithme highest lock (aka Immediate Ceiling Priority Protocol).

Question 2.51. Dessinez les 16 premières unités de temps du cas précédent.

Question 2.52. Calculez les temps de réponses des différentes tâches en utilisant un ordonnancement à priorité statique et l'algorithme original ceiling priority protocol (OCP).

Question 2.53. Dessinez les 16 premières unités de temps du cas précédent.

2.11 Inversion de priorité - 3

Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Priorité	Sections critiques
A	5	-	4	1	_11_
B	5	-	1	2	_
C	2	-	6	3	_1 $\frac{1}{2}$ $\frac{1}{2}$ 1_
D	2	-	1	4	_
E	0	-	5	5	_222_

Question 2.54. Dessinez les 16 premières unités de temps ordonnancement à priorité statique et sans algorithme de gestion d'inversion de priorité.

Question 2.55. Dessinez les 16 premières unités de temps en utilisant un ordonnancement à priorité statique et l'algorithme d'héritage de priorité.

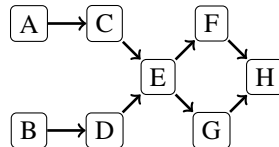
Question 2.56. Calculez les temps de réponses des différentes tâches en utilisant un ordonnancement à priorité statique et l'algorithme highest lock (aka Immediate Ceiling Priority Protocol).

Question 2.57. Dessinez les 16 premières unités de temps du cas précédent.

Question 2.58. Calculez les temps de réponses des différentes tâches en utilisant un ordonnancement à priorité statique et l'algorithme original ceiling priority protocol (OCPP).

Question 2.59. Dessinez les 16 premières unités de temps du cas précédant.

2.12 Contraintes de précedence



Soit les tâches suivantes :

Tâche	Arrivée	Période	Capacité	Échéance
A	0	-	3	12
B	0	-	1	10
C	0	-	3	20
D	0	-	1	15
E	0	-	5	25
F	0	-	4	37
G	0	-	1	20
H	0	-	1	40

Question 2.60. Dessinez les 20 premières unités de temps en utilisant un ordonnancement EDF (préemptif) sans prendre en compte les contraintes de précedence. Est-ce conforme aux besoins de l'application ?

Question 2.61. Utilisez la méthode de Blazewicz/Chetto pour supprimer les contraintes de précedence.

Question 2.62. Calculez de nouveau le taux d'utilisation et redessinez l'ordonnancement ainsi généré sur sa période d'étude. Conclure sur l'ordonnancabilité de l'application.

3 Watch file

3.1 Par scrutation

Question 3.1. Implémenter un programme `watch_files` prenant en paramètre un nom de fichier. Par scrutation, votre programme détectera chaque fois que le fichier est créé ou supprimé. Votre programme devra afficher un message à chaque fois qu'il détecte une modification. Vous devrez être le plus réactif possible lors de la modification de statut du fichier.

Documentation utile: `stat(2)`, `touch(1)`, `rm(1)`

Question 3.2. Sans effectuer de mesure précise, quel est l'ordre de grandeur du temps réponse de votre programme ?

Question 3.3. Que constatez-vous sur le taux d'utilisation du CPU ?

Documentation utile: `top(1)`

Question 3.4. *Afin de rendre notre programme moins gourmand, ajouter une temporisation de 1s entre chaque scrutation de l'état du fichier.*

Documentation utile: `sleep(3)`

Question 3.5. *Que constatez-vous sur le taux d'utilisation du CPU ?*

Documentation utile: `top(1)`

Question 3.6. *Quel est maintenant le temps de réponse de votre programme ?*

Question 3.7. *Comment évoluerait le taux d'utilisation du CPU et le temps de réponse si nous devons surveiller l'état de plusieurs millions de fichiers changeant rarement ?*

3.2 Par notification

Afin d'améliorer notre programme, nous allons utiliser un système piloté par évènements. Ce mode de fonctionnement est assez proche d'un système piloté par interruption.

Question 3.8. *Modifiez votre programme afin d'être notifié des modifications sur votre fichier.*

Documentation utile: `inotify(7)`, `inotify_init(2)`, `inotify_add_watch(2)`, `dirname(3)`, `basename(3)`, `strncpy(3)`, `strcmp(3)`, `read(2)`, `close(2)`

Question 3.9. *Quel est maintenant le temps de réponse de votre programme ?*

Question 3.10. *Que constatez-vous sur le taux d'utilisation du CPU ?*

Documentation utile: `top(1)`

Question 3.11. *Comment évoluerait le taux d'utilisation du CPU et le temps de réponse si nous devons surveiller l'état de plusieurs millions de fichiers changeant rarement ?*

4 Protection des structures de données

Nous avons implémenté une structure de liste simplement chaînée :

```

/*
 * Creation date: 2010-01-03 11:22:08+01:00
 * Licence: GPL
 * Main authors:
 * - édôJrme Pouiller <jerome@sysmic.org>
 *
 * Sample use of mutexes.
 */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

```

```

typedef struct list_item_s {
    struct list_item_s *next;
    int value;
} list_item_t;

/* Add an element in list */
20 void list_add(list_item_t **list, int n) {
    list_item_t *item = malloc(sizeof(list_item_t));
    item->value = n;
    item->next = *list;
    *list = item;
}

/* Remove an element */
30 void list_del(list_item_t **list, int n) {
    list_item_t *i, *tmp;

    if ((*list)->value == n) {
        tmp = *list;
        *list = (*list)->next;
        free(tmp);
        return ;
    }
    for (i = *list; i->next; i = i->next)
        if (i->next->value == n) {
40         tmp = i->next;
            i->next = tmp->next;
            free(tmp);
            return ;
        }
    printf("Item %d not found\n", n);
}

```

Nous avons écrit un scénario de test comme il se doit :

```

éô
#define N_ITERATIONS 110
void *task(void *arg) {
4     list_item_t **l = arg;
    int i;

    list_add(l, 0);
    for (i = 1; i < N_ITERATIONS; i++) {
        list_add(l, i);
        list_del(l, i - 1);
        if (!(i % 100))
            printf("%08lx: %d\n", pthread_self(), i);
    }
14     list_del(l, N_ITERATIONS - 1);

    return NULL;
}

#define NB_THREAD 2
int main(int argc, char **argv) {
    // Initialize an empty list
    list_item_t *l = NULL;
    pthread_t id[NB_THREAD];
24     int i;
}

```

34

```

for (i = 0; i < NB_THREAD; i++)
    pthread_create(&id[i], NULL, task, (void *) &l);

for (i = 0; i < NB_THREAD; i++)
    pthread_join(id[i], NULL);

if (l)
    printf("Error: list is not empty\n");
return 0;
}

```

Néanmoins, le résultat n'est pas celui que nous attendions.

Question 4.1. *Observez le comportement avec différentes valeurs de N_ITERATIONS et NB_THREAD. Quel est le problème ?*

Remarque: La question peut se faire en environ 2 lignes.

Documentation utile: `pthread_create(3)`

Question 4.2. *Résolvez le problème. Vous prendrez soin de limiter la portée de vos sections critiques.*

Remarque: La question peut se faire en moins de 10 lignes.

Documentation utile: `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`

5 Tricky conditions

Nous avons un problème avec le code ci-dessous :

```

/*
 * Creation date: 2010-01-03 11:22:08+01:00
 * Licence: GPL
 * Main authors:
5  * - éôJrme Pouiller <jerome@sysmic.org>
 *
 * Sample use of conditions.
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>

15 static pthread_cond_t c = PTHREAD_COND_INITIALIZER;
static pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
static int count = 0;

void *sender(void *arg) {
    int i;

    for (i = 0; i < 10; i++) {
        count++;
        printf("%08lX: Increment count: count=%d\n", pthread_self(), count);
25     if (count == 3) {
        pthread_cond_signal(&c);
        printf("%08lX: Sent signal. count=%d\n", pthread_self(), count);
    }
}
}

```

```

    return NULL;
}

void *receiver(void *arg) {
    printf("%08lX: Begin waiting\n", pthread_self());
35 pthread_cond_wait(&c, &m);
    printf("%08lX: Received signal. count=%d\n", pthread_self(), count);
    return NULL;
}

int main(int argc, char **argv) {
    pthread_t id[2];

    pthread_mutex_lock(&m);
    pthread_create(&id[0], NULL, sender, NULL);
45 pthread_create(&id[1], NULL, receiver, NULL);

    pthread_join(id[0], NULL);
    pthread_join(id[1], NULL);
    pthread_mutex_unlock(&m);

    return 0;
}

```

De temps en temps, le signal envoyé par la thread *sender* n'est pas reçu par la thread *receiver*.

Question 5.1. Exacerber le problème en plaçant une temporisation dans le code.

Remarque: La question peut se faire en 2 lignes.

Documentation utile: `sleep(3)`

Question 5.2. Utilisez le mutex `m` de manière appropriée pour résoudre le problème.

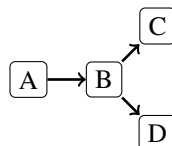
Remarque: La question peut se faire en 4 lignes.

Documentation utile: `pthread_cond_wait(3)`, `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`

6 Structures de données et mutex

Un réseau de chemin de fer relie les villes A, B, C et D. Deux trains circulent sur ce réseau :

- Le train circulant entre A, B et C
- Le train circulant entre A, B et D



On souhaite effectuer une simulation du fonctionnement de ce réseau de chemin de fer sachant que chaque segment de la ligne ne peut être utilisé que par un train à la fois. Néanmoins, un nombre illimité de trains peuvent se trouver dans la même gare.

Nous avons décidé de modéliser chaque train par une thread et le déplacement d'une ville à une autre par une temporisation d'1s pendant laquelle le segment de ligne est occupé.

Nous utilisons une structure de graphe pour représenter notre réseau ferré. Ce choix nous permettra de facilement étendre notre algorithme à de nouveaux cas.

Question 6.1. Implémenter cette simulation.

Remarque: Une manière commune de représenter un graphe de X nœuds en informatique est d'utiliser une matrice de $X \times X$ booléens. Chaque valeur représente l'existence (ou l'absence) d'une arête. Dans le cas d'un graphe non-orienté, la matrice sera symétrique :

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Remarque: La question peut se faire en environ 80 lignes.

Documentation utile: `pthread_create(3)`, `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`, `sleep(3)`

7 Protection de ressources

Nous avons besoin d'envoyer 4 messages vers un serveur distant. Nous utilisons le code ci-dessous :

```
#define SZ 20

void *task(void *arg) {
    char *buf = (char *) arg;

    send(buf, SZ);
    return NULL;
}

10 int main(int argc, char **argv) {
    pthread_t id[4];
    int i;
    char msgs[4][SZ] = { "HELO sysmic.org\n", "SEND jezz\n", "QUOTA\n", "BYE\n" };

    for (i = 0; i < 4; i++)
        pthread_create(&id[i], NULL, task, msgs[i]);

    for (i = 0; i < 4; i++)
        pthread_join(id[i], NULL);

20 printf("\n");
    return 0;
}
```

Dans le cadre de notre étude, nous n'avons pas d'accès réel au serveur, nous avons donc écrit une fonction de simulation de l'envoi des données :

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

5 void send_char(char data) {
    double time = (double) random() / RAND_MAX * 8;
    struct timespec t;
    t.tv_sec = time / 1000;
    t.tv_nsec = (int) (time * 1000000) % 1000000000;
    nanosleep(&t, NULL);
    write(1, &data, 1);
}

15 void send(char *data, size_t len) {
```

```

unsigned i;

for (i = 0; i < len && data[i]; i++)
    send_char(data[i]);
}

```

Malheureusement, il semblerait qu'il y ait un problème car les messages reçu par le serveurs ne sont pas cohérents :

```
Aa[1b{cB2(deC3)}DE45Ff]6
```

Or, pour que le serveur puisse interpréter les données, il ne faut pas que les messages des *tâches* concurrentes soit enchevêtrés. On propose d'emballer `send` dans une nouvelle fonction alternative permettant de régler le problème :

```
int safe_send(char *data, size_t len);
```

Question 7.1. Implémentez `safe_send`.

Remarque: La question peut se faire en moins de 10 lignes.

Documentation utile: `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`

La fonction `safe_send` résout le problème de cohérence, mais cause des problèmes de latences dans la fonction `task`. Dans certains cas, celle-ci s'exécute beaucoup plus lentement qu'avant.

Question 7.2. Écrivez un scénario de test permettant de mettre en évidence et de mesurer le phénomène de latence.

Remarque: La question peut se faire en moins de 20 lignes.

Documentation utile: `clock_gettime(3)`, `memset(3)`

Nous allons essayer de résoudre le problème de latence. Pour cela, nous allons créer une nouvelle *tâche* permettant d'envoyer les données de manière asynchrone.

Afin de transmettre à la *tâche* les données à envoyer, nous devons mettre en place une structure de données adéquate. Nous proposons d'utiliser une file (FIFO) sous la forme d'un *buffer circulaire*.

Enfin, notre *buffer circulaire* nécessitera un mécanisme de réveil de la *tâche* d'envoi lorsque des données sont disponible dans le *buffer*.

Question 7.3. Implémentez cette solution.

Remarque: La question peut se faire en moins de 40 lignes.

Documentation utile: `pthread_create(3)`, `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`, `pthread_cond_wait(3)`, `pthread_cond_signal(3)`

8 Threads et mutex

Question 8.1. Écrivez un programme avec N tâches, N étant une constante définie à la compilation. Chacune des tâches contiendra une boucle incrémentant et affichant un compteur. Utilisez des sémaphores pour synchroniser les boucles. Vous devriez obtenir le comportement suivant ($N = 4$) :

```

$ ./concurrence
Task: 0, count: 0
Task: 1, count: 0
Task: 2, count: 0
Task: 3, count: 0
Task: 0, count: 1
Task: 1, count: 1
Task: 2, count: 1
Task: 3, count: 1

```



```
Task: 0, count: 2
Task: 1, count: 2
Task: 2, count: 2
[...]
```

Documentation utile: `pthread_create(3)`, `pthread_cancel(3)`, `pthread_join(3)`, `pthread_mutex_init`, `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`, `pthread_mutex_destroy(3)`

Question 8.2. Modifiez votre code pour créer un dead lock sur toutes les taches.

Question 8.3. Fixez N à 4. Ajoutez une cinquième tache. La cinquième tache devra monopoliser processeur (ie : `for(;;)`). En modifiant les priorités des taches, produisez une inversion de priorité.

Remarque: Sur un système multicoeurs, il est possible que vous ayez besoin de créer autant de tâches que de coeurs afin de monopoliser le système.

Documentation utile: `pthread_setschedparam(3)`, `pthread_setschedpolicy(3)`

9 Tâches et sémaphores Xenomai

Afin de faire ce TP, nous vous avons fourni un Linux avec un noyau patché avec Xenomai. Ce patch nous permet d'avoir un système temps réel et profiter d'une API spécialisée pour le temps réel. Notez que nous utiliserons exclusivement l'API Native durant ce TP.

Reportez-vous à l'annexe B pour les informations concernant l'usage de la bibliothèque Xenomai.

Question 9.1. Écrivez un programme avec N taches, N étant une constante définie à la compilation. Chacune des taches contiendra une boucle incrémentant et affichant un compteur. Utilisez des sémaphores pour synchroniser les boucles. Vous devriez obtenir le comportement suivant ($N = 4$) :

```
$ ./concurrence
Task: 0, count: 0
Task: 1, count: 0
Task: 2, count: 0
Task: 3, count: 0
Task: 0, count: 1
Task: 1, count: 1
Task: 2, count: 1
Task: 3, count: 1
Task: 0, count: 2
Task: 1, count: 2
Task: 2, count: 2
[...]
```

Documentation utile: `native/task.h`, `rt_task_create`, `rt_task_start`, `rt_task_delete`, `native/sem.h`, `rt_sem_create`, `rt_sem_delete`, `rt_sem_p`, `rt_sem_v`, `mlockall(2)`

Question 9.2. Vérifiez le bon fonctionnement de votre programme avec $N = 100$.

10 Implémentation d'ordonnanceurs

Le fichier `sched.c` implémente un environnement capable de simuler des ordonnanceurs. Ce simulateur est basé sur les fonction Posix `makecontext` et `swapcontext` qui permettent de contrôler manuellement le changement de contexte entre plusieurs tâches.

Question 10.1. Étudiez `sched.c`.

Documentation utile: `makecontext(3)`, `swapcontext(3)`

10.1 Algorithmes d'ordonnement de tâches indépendantes

Question 10.2. A partir de `sched.c`, modifiez la fonction `elect` afin d'implémenter un ordonnanceur à priorité statique.

Question 10.3. Plutôt que d'utiliser une durée de simulation arbitraire, calculez celle-ci afin que la durée de la simulation soit suffisante pour faire simulation hors-ligne.

Question 10.4. Implémentez un ordonnanceur Rate Monotonic. Bien entendu le membre `priority` de la structure `task_t` ne sera pas utilisé pour cet algorithme.

Question 10.5. Implémentez un ordonnanceur Earliest Deadline First. Bien entendu le membre `priority` de la structure `task_t` ne sera pas utilisé pour cet algorithme.

10.2 Algorithmes d'ordonnement de tâches apériodiques

Nous souhaitons ajouter des tâche apériodiques. Nous les intégrons à notre système en utilisant des tâche avec une période nulle.

Question 10.6. Ajoutez la gestion de ces tâches. Utilisez le temps CPU non-utilisé par les tâches périodiques pour exécuter ces tâches. Vous n'avez pas besoin de gérer de priorités entre les tâches asynchrones.

Question 10.7. Effectuez la gestion des tâches apériodiques dans un serveur de scrutation (polling server).

Question 10.8. Effectuez la gestion des tâches apériodiques dans un serveur différé (defferable server)

Question 10.9. (Bonus) Effectuez la gestion des tâches apériodiques dans un serveur sporadique (sporadic server).

Question 10.10. Implémentez l'algorithme de Round-Robin.

10.3 Ordonnancement et ressources partagées

Nous souhaitons ajouter la gestion des mutex dans notre système. Le patch `sched_mutex.patch` ajoute les structures nécessaires pour la gestion des mutex.

Question 10.11. Appliquez `sched_mutex.patch` à votre implémentation d'ordonnanceur à priorités statiques et implémentez les fonctions `lock` et `unlock` ;

Question 10.12. Implémentez un scénario avec un deadlock.

Question 10.13. Implémentez un scénario avec une inversion de priorité.

Question 10.14. Implémentez l'algorithme ICPP (Immediate Ceiling Priority Protocol).

Documentation utile: `qsort(3)`

Question 10.15. (Bonus) Implémentez l'algorithme d'héritage de priorité.

Documentation utile: `qsort(3)`

Question 10.16. (Bonus difficile) Dans notre simulation, les date de préemption sont connues à l'avance. Par conséquent, nous avons pu nous affranchir des problèmes de réentrance. Modifier votre implémentation d'héritage de priorité afin de la rendre réentrante.

Question 10.17. (Sujet de thèse) Ajoutez à l'ordonnanceur la possibilité d'ordonner les tâches sur plusieurs processeurs simultanément. Maintenez votre implémentation d'héritage de priorité fonctionnelle (et réentrante). Garantissez des temps de réponse de vos algorithmes en $O(n) + O(s)$ (où n est le nombre de tâches et s le nombre de sections critiques). Vos propositions sont à envoyer à Ingo Molnár, auteur des mutex à héritage de priorité, de l'ordonnanceur Completely Fair et de la préemption dite Temps Réelle sur le noyau Linux.

Question 10.18. Implémentez l'algorithme OCPP (Original Ceiling Priority Protocol).

Documentation utile: `qsort(3)`

Question 10.19. Détectez les possibles inversions de priorités dans les fonctions `m_lock`, `m_unlock`.

Question 10.20. Détectez les dead locks dans les fonctions `m_lock`, `m_unlock`.

11 Libtimer

On se propose d'implémenter une bibliothèque `libtimer` permettant de lancer des tâches à une date fixe ou après une certaine période. Le fonctionnement de cette bibliothèque est décrit dans `timer.h` :

```
/*
 * Author:   Jerme Pouiller <jerome@sysmic.fr>
 * Created: Fri Dec 11 22:57:13 CET 2009
 * Licence: GPL
 *
 * Define interface for timer services.
 */
```

```

9  #ifndef MYTIMER_H
   #define MYTIMER_H
   #include <pthread.h>
   #include <time.h>

   /* Helper to define a task */
   typedef void (*task_t) ();

   /* INSERT YOUR mytimer_s DEFINITION HERE */

19  /* Contains all needed data */
   typedef struct mytimer_s mytimer_t;

   /*
    * All following functions return
    * - 0 on success
    * - another value on error
    */

   /*
29  * Initialize new timer instance.
    * Instance is returned using parameter out.
    * User haveto call this function first
    */
   int mytimer_run(mytimer_t *out);

   /* Stop timer instance */
   int mytimer_stop(mytimer_t *mt);

   /* Add a task to timer. Task will be executed at time tp */
39  int mytimer_add_absolute(mytimer_t *mt, task_t t, const struct timespec *tp);

   /* Add a task to timer. Task will be executed in ms milliseconds */
   int mytimer_add_msecond(mytimer_t *mt, task_t t, unsigned long ms);

   /* Add a task to timer. Task will be executed in s seconds */
   int mytimer_add_second(mytimer_t *mt, task_t t, unsigned long sec);

   /* Add a task to timer. Task will be executed in h hours */
   int mytimer_add_hour(mytimer_t *mt, task_t t, unsigned long h);

49  /* Remove first occurrence of task t from timer. */
   int mytimer_remove(mytimer_t *mt, task_t t);

   #endif /* MYTIMER_H */

```

- La structure `mytimer_s` (que vous définirez) contient tout le contexte d'exécution.
- Les *tâches* à lancer sont décrite par le type `task_t`.
- L'utilisateur de la bibliothèque commencera par un appel à `mytimer_run`.
- Il ajoutera les *tâches* avec les fonctions `mytimer_add`.
- `mytimer_add_absolute` exécute la *tâche* à une heure fixe donnée tandis que les autres fonctions `mytimer_add` exécutent la *tâche* à un temps relatif à l'instant présent. Par exemple, `mytimer_add_second(s, t, 3)` exécutera `t` 3 secondes après l'appel à cette fonction.
- Toutes les fonctions de `libtimer` retournent un entier. Il s'agit d'un code d'erreur. Vous êtes libre dans le choix des codes de retour.

On considère que les *tâches* sont exemptes de bugs et qu'elles ne tentent pas de mettre en péril la stabilité de votre bibliothèque et du système.

libtimer tentera d'être le plus précis possible dans l'heure de lancement des *tâches*. Néanmoins, on ignorera le biais introduit par le système d'exploitation et les autres processus.

Afin de vous aider, vous trouverez une implémentation de liste chaînée dans `list.h` et `list.c` :

```

/*
 * Author: éôJrme Pouiller <jerome@sysmic.fr>
 * Created: Fri Dec 11 22:57:54 CET 2009
 * Licence: GPL
5  *
 */
#ifdef LIST_H
#define LIST_H
#include <stdlib.h>
#include <stdbool.h>

typedef struct list_item_s {
    /* ADD YOUR MEMBERS HERE */
    struct list_item_s *next;
15 } list_item_t;

#define EMPTY_LIST NULL
#define NEXT(s) (s)->next
#define FOREACH(I, LIST) for (I = LIST; I; I = NEXT(I))

bool list_isEmpty(list_item_t *l);

int list_size(list_item_t *list);

25 /* Add an element. Don't forget to allocate it before */
void list_add(list_item_t **list, list_item_t *item);

/* Remove an element. Return removed element. Don't forget to free it AFTER removing
 */
list_item_t *list_remove(list_item_t **list, list_item_t *item);

#endif /* LIST_H */

```

```

/*
 * Author: éôJrme Pouiller <jerome@sysmic.fr>
 * Created: Fri Dec 11 22:57:54 CET 2009
 * Licence: GPL
 *
 */
#include "list.h"
9 bool list_isEmpty(list_item_t *l) {
    return (l == EMPTY_LIST);
}

int list_size(list_item_t *list) {
    int count = 0;
    list_item_t *i;

    FOREACH(i, list)
        count++;
19 return count;
}

```

```

29 /* Add an element. Don't forget to allocate it before */
void list_add(list_item_t **list, list_item_t *item) {
    item->next = *list;
    *list = item;
}

/* Remove an element. Return removed element. Don't forget to free it AFTER removing
*/
39 list_item_t *list_remove(list_item_t **list, list_item_t *item) {
    list_item_t *i;

    if (*list == item) {
        *list = (*list)->next;
        return item;
    }
    FOREACH(i, *list)
        if (i->next == item) {
            i->next = item->next;
            return item;
        }
    // Nothing found
    return NULL;
}

```

11.1 Version synchrone

On considère pour l'instant que le temps d'exécution des *tâches* est nul.

Question 11.1. Implémentez `libtimer` tel que décrit dans le fichier d'entête `timer.h`.

Documentation utile: `pthread_cond_timedwait(3)`, `pthread_cond_wait(3)`, `pthread_cond_signal(3)`, `clock_gettime(2)`, `sleep(3)`, `pthread_create(3)`, `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`, `pthread_mutex_init(3)`, `pthread_cond_init(3)`, `pthread_cancel(3)`, `ctime_r(3)`

11.2 Version asynchrone

Nous avons supposé dans que le temps d'exécution des *tâches* étaient nul. Nous souhaitons maintenant écrire la version finale de la bibliothèque exécutant les *tâches* de manière asynchrone.

Question 11.2. Modifiez votre bibliothèque de manière à exécuter les tâches de manière concurrente.

Documentation utile: `pthread_attr_setdetachstate(3)`, `pthread_create(3)`, `pthread_attr_init(3)`

12 Calculs et temps réel

12.1 Introduction

Ce TP est inspiré du fonctionnement d'un téléphone portable. Sur ce système, un microphone utilise un DMA pour placer les données acquises en mémoire. Une interruption est ensuite déclenchée lorsque que le DMA contient assez de données par être traitées. Le téléphone portable utilise ensuite un système de compression par paquet dont le temps de traitement est variable. Enfin, le téléphone doit envoyer ses données à l'antenne toutes les 4.6ms, soit à une fréquence de 217Hz environ.

Afin de faire ce TP, nous vous avons fourni un Linux avec un noyau patché avec Xenomai. Ce patch nous permet d'avoir un système temps réel et profiter d'une API spécialisée pour le temps réel.

12.2 Écriture du Watchdog

Nous allons utiliser le périphérique *Real Time Clock (RTC)* présent sur les architectures i386 pour générer des interruptions nécessaires à la suite de l'exercice.

A chaque période, la RTC produit une interruption. Le driver RTC traduit cette interruption par l'écriture d'un octet sur le fichier `/dev/rtc0`. Le fichier `rtc.c` vous donne un exemple de programmation du périphérique RTC.

Question 12.1. Étudiez le programme, compilez-le, puis programmez la RTC à 2 Hz et lancez trois périodes.

Nous utiliserons la bibliothèque Xenomai pour les questions suivantes. Reportez-vous à l'annexe B pour plus d'informations sur son utilisation.

Nous allons dans la suite du TP travailler avec l'interruption produite par la RTC. Une mauvaise gestion de l'interruption pourra produire un IRQ storm. Nous allons donc mettre en place watchdog afin de rendre notre développement plus confortable. Notre watchdog nous permettra de détecter si notre système ne répond plus et de reprendre la main dessus.

Question 12.2. Modifiez `rtc.c` de manière à ajouter un watchdog. Déclencher votre watchdog sur 1 seconde. Lorsque le watchdog se déclenche, désactivez la RTC et avertissez l'utilisateur.

Documentation utile: `native/alarm.h`, `rt_alarm_start`, `rt_alarm_create`, `rt_alarm_delete`, `rt_alarm_wait`, `native/task.h`, `rt_task_create`, `rt_task_delete`, `rt_task_start`, `rt_timer_ns2ticks`

12.3 Ecriture de l'ISR

Maintenant que nous avons un bon outil de test, nous pouvons commencer à travailler sur l'interruption en elle-même.

Les interruptions doivent forcément être exécutées dans le noyau de Linux. Pour cette raison les questions suivantes sont à écrire sous forme de module. Reportez-vous à l'annexe C pour les informations concernant le développement des modules noyau.

Question 12.3. Placez une ISR sur le l'interruption 8 (Real Time Clock). Dans votre ISR, gérez un compteur d'appel n . Affichez n dans le buffer du noyau lorsqu'une interruption survient.

Documentation utile: `native/intr.h`, `rt_intr_create`, `rt_intr_enable`, `rt_intr_delete`

Remarquez le fonctionnement du nano-kernel Adeos. Votre interruption apparaît dans `/proc/ipipe/Xenomai` alors que les autres interruption apparaissent dans `/proc/ipipe/Linux`. Adeos gère Xenomai et Linux comme deux OS différents.

Nous allons maintenant simuler le traitement de données. Nous choisissons de calculer la suite de Fibonacci pour simuler le temps de traitement des données. Nous utilisons cette fonction en raison de sa grande amplitude de temps de calcul. Ainsi, `fibonacci(34)` se calcule 4 milliards de fois plus lentement que `fibonacci(2)`.

Question 12.4. Faites une implémentation naïve (récursive) de Fibonacci. Affichez `fibonacci(n % M)` lorsqu'une interruption survient, M étant une constante définie à la compilation.

Augmentez M jusqu'à ce que le temps passé dans l'interruption soit mesurable ($> 100ms$) lorsque n approche un multiple de M .

Remarque: Vous pouvez utiliser le timestamp produit dans le log buffer du noyau comme mesure approximative.

Question 12.5. Augmentez la fréquence de la RTC à 128Hz. Vérifiez le fonctionnement de l'interruption.

Remarque: Vous pouvez utiliser le timestamp produit par le log buffer. Vu que l'on casse le temps réel du noyau, ce timestamp n'est pas fiable en absolu. Néanmoins, le temps en moyenne est fiable. On peut calculer le nombre d'interruptions entre le début et la fin du test et ainsi voir si on rate des interruptions.

Question 12.6. Déportez le calcul et l'affichage de `fibonacci` dans une seconde tâche à l'aide d'une queue. Affichez le nombre d'éléments dans la queue. N'oubliez pas de gérer les erreurs.

Documentation utile: `rt_queue_create`, `rt_queue_delete`, `rt_queue_write`, `rt_queue_read`, `rt_queue_inquire`, `rt_task_create`, `rt_task_start`, `rt_task_delete`

Question 12.7. Ajoutez la gestion d'une file de résultats. Dans l'ISR, dépiler un résultat et affichez-le. Gérez intelligemment le cas où la file est vide.

Documentation utile: `native/queue.h`, `rt_queue_create`, `rt_queue_delete`, `rt_queue_write`, `rt_queue_read`

Question 12.8. Chargez le CPU à l'aide des commande `dd(1)` et `ping(1)`. Vérifiez la qualité de vos taches temps réelles.

A Instructions d'installation de la machine virtuelle Xenomai

A.1 Mise en service de l'image

Nous avons besoin de configurer votre compte pour utiliser l'image virtuelle patché avec Xenomai.

Ouvrez un terminal. Commencez par importer la configuration de la machine virtuelle :

```
vboxmanage import "/opt/TP_vms/jezz-324/Ubuntu 9.04 Xenomai 32bits-nodisk.ovf"
```

Cette configuration ne contient pas de disque dur. Nous en attachons donc un :

```
vboxmanage storageattach "Ubuntu 9.04 Xenomai 32bits" --storagectl "SATA Controller"
--port 1 --type hdd --medium "/opt/TP_vms/jezz-324/Ubuntu 9.04 Xenomai 32bits-
disk1.vmdk"
```

Le disque dur que nous avons ajouté n'est pas accessible en lecture. Nous allons utiliser le système d'image incrémentale proposé par Virtual Box. Ainsi, vous n'avez besoin d'avoir un accès en écriture que sur le fichier contenant le delta. Ce delta pourra tout de même atteindre quelques centaine de mégaoctet. Nous choisissons donc un emplacement en local pour le contenir :

```
vboxmanage modifyvm "Ubuntu 9.04 Xenomai 32bits" --snapshotfolder /var/tmp/snapshots-
LOGIN/
```

Nous pouvons maintenant créer le premier incrément :

```
vboxmanage snapshot "Ubuntu 9.04 Xenomai 32bits" take "Initial"
```

Vous pouvez démarrer `VirtualBox` et démarrer la machine virtuelle "Ubuntu 9.04 Xenomai 32bits"

A.2 L'environnement

A.2.1 Se connecter

```
login : user
password : user
```

A.2.2 Sauver son travail

Dans un terminal utiliser la commande `scp(1)` pour vous connecter à votre machine hôte par l'IP 10.0.2.2 :

```
scp -r ex01 pouiller@10.0.2.2:
```

L'inverse fonctionne aussi évidemment :

```
scp -r pouiller@10.0.2.2:ex01 .
```

A.2.3 Accéder à internet

Vous devez configurer votre navigateur pour utiliser le proxy de l'école.

A.2.4 Travailler avec emacs, geany, etc...

Vous avez la possibilité d'exécuter des commandes en root avec la commande `sudo(1)` :

```
sudo apt-get install emacs
```

B Écriture des programmes Xenomai

B.1 Documentation de Xenomai

Vous pouvez trouver la documentation de l'API de Xenomai dans `/usr/share/doc/xenomai-doc/html/api` ou en ligne sur <http://www.xenomai.org/documentation/xenomai-2.4/html>.

Remarquez que est capable de présenter plusieurs interfaces de programmation. Dans notre cas, nous n'utiliserons que l'interface *native*. Par conséquent, seule la section "Modules/Native Xenomai API" de la documentation vous sera utile.

Vous pouvez trouver les headers à inclure en explorant l'onglet *files*

B.2 Compilation

Les commandes `xeno-config --xeno-cflags` et `xeno-config --xeno-ldflags` permettent d'obtenir respectivement les options de compilation et les options de link à utiliser pour que votre programme fonctionne avec Xenomai. Vous devrez de plus préciser que vous utilisez la skin Native avec l'option `-lnative`.

Par ailleurs, nous vous conseillons fortement de compiler avec l'option `-Wall` de `gcc` et d'utiliser un système de Makefile.

N'oubliez pas que dans un Makefile, il faut utiliser le mot-clef `shell` pour lancer une commande. Par exemple :

```
bin: file.o
    gcc -Wall $(shell xeno-config --xeno-ldflags) -lnative $< -o $@
file.o: file.c
    gcc -Wall $(shell xeno-config --xeno-cflags) -c $< -o $@
```

B.3 Verrouillage de la mémoire

En cas de nécessité, le gestionnaire de mémoire du noyau Linux peut-être amené à placer des pages de mémoire virtuelle sur le disque dur (processus de *swapping*). Lorsqu'une de ces pages de mémoire est nécessaire, elle doit être replacée en mémoire avant d'être utilisée. Cela crée une latence qui peut poser problème dans le cas d'application temps réelles.

Un appel à la fonction `mlockall(1)` permet de demander au noyau de verrouiller les pages de mémoire en mémoire physique :

```
mlockall(MCL_CURRENT | MCL_FUTURE);
```

La mémoire doit être verrouillée avant tout appel à l'API Xenomai.

B.4 A propos des noms des objets Xenomai

Il est possible dans Xenomai d'associer chaque objet à un nom unique sur le système. Cela permet de *binder* des objets entre des processus ou entre des domaines d'exécution.

L'utilisation des noms oblige à être rigoureux et à correctement détruire les objets avant de sortir des processus. Les instances des objets resteront en mémoire.

Par conséquent, nous vous conseillons de ne pas utiliser cette fonctionnalité des les exercices suivants et à passer un pointeur `NULL` comme nom.

B.5 Gestion des Erreurs

La plupart des fonctions Xenomai retournent une valeur négative pour indiquer qu'une erreur s'est produite. Vous pouvez utiliser `strerror(3)` pour obtenir le message d'erreur équivalent.

Exemple classique :

```
if (err = rt_task_start(t, f, NULL))
    printf("Task start: %s\n", strerror(-err));
```

B.6 Exécution

Vous aurez besoin d'exécuter les programmes que vous créerez avec les droits `root`. Nous vous déconseillons l'utilisation de `sudo(1)` pour lancer vos programmes car `Ctrl+C` sera moins réactif.

C Écriture des modules noyau

C.1 Compilation

La manière classique de compiler un module noyau est de créer un Makefile contenant :

```
EXTRA_CFLAGS=-I/lib/modules/$(shell uname -r)/build/include/xenomai
obj-m += module.o
```

Puis lancer la compilation avec :

```
make -C /lib/modules/$(uname -r)/build SUBDIRS=$(pwd) modules
```

Le système de compilation du noyau se chargera de compiler votre module avec les bonnes options.

Nous vous fournissons le fichier `Makefile` reprenant ces lignes.

C.2 Exécution

Les modules ne sont pas des applications. Ils n'ont pas de fonction `main`. Comme c'est souvent le cas lorsque l'on souhaite étendre une application en cours de fonctionnement, vous avez la possibilité de déclarer des fonctions qui seront appelées lors de certains événements. Les fonctions les plus utiles sont déclarées par `MODULE_INIT` (chargement du module) et `MODULE_EXIT` (déchargement du module).

Vous pourrez charger le module à l'aide de `insmod(1)` et le décharger à l'aide de `rmmmod(1)`.

Nous vous fournissons le fichier `module.c` qui vous donne un squelette de module pour le noyau Linux.

C.3 Log Buffer

Le noyau possède un buffer circulaire permettant de logger des messages. Il est possible d'écrire dans le buffer depuis un module à l'aide de `printk(9)` et des macros `pr_debug`, `pr_info`, `pr_notice`, `pr_warning`, `pr_err`, etc.... Il est possible de dumper le contenu du buffer avec la commande `dmesg(1)` (utilisant l'appel système `syslog(2)`). Il est aussi possible de suivre l'évolution du buffer avec la commande :

```
tail -f /var/log/kern.log
```

D Modalités de rendu

Votre rendu fera l'objet d'un traitement automatisé. Afin d'éviter tout problème, veuillez suivre les règles suivantes.

Vous devez rendre votre travail dans une archive gzippée. L'archive devra se nommer `TR-nom.tar.gz` et se décompresser dans le répertoire du même nom.

Votre archives ne pas contenir de fichiers binaires. Il ne doit contenir que vos sources (fichiers sources, fichiers entêtes, Makefile, etc...).

Aucun de vos noms de fichiers ne doit contenir de caractères d'espacements, accentués ou non-imprimables.

Exemple de création de rendu :

```
make clean
cd ..
tar cvzf TR-pouiller.tar.gz TR-pouiller
```

Vous devrez envoyer votre rendu par e-mail avant le 31 Décembre 1999 à 23h59 à rendus@sysmic.fr. Votre e-mail ne devra comporter que votre rendu comme unique pièce jointe.

Le sujet du mail devra suivre la nomenclature suivante : `[TR] nom` (exemple : `[TR] pouiller`).

Enfin, le corps du mail devra contenir une ligne indiquant votre prénom, votre nom et votre email sous la forme :

```
* Prénom Nom <email@entreprise.dom>
```

Exemple :

```
Veuillez trouver en pièce jointe le rendu de projet temps réel de :
* Jérôme Pouiller <j.pouiller@sysmic.fr>
```

Cordialement,

--

Jérôme Pouiller, Sysmic
Expert Linux embarqué

E Modalités de rendu

Votre rendu fera l'objet d'un traitement automatisé. Afin d'éviter tout problème, veuillez suivre les règles suivantes.

Vous devez rendre votre travail dans une archive gzippée. L'archive devra se nommer `TR-nom.tar.gz` où `nom` sera le nom d'un membre de votre équipe que nous désignerons par *contact administratif*. Votre archive devra et se décompresser dans le répertoire du même nom.

Votre archives ne pas contenir de fichiers binaires. Il ne doit contenir que vos sources (fichiers sources, fichiers entêtes, Makefile, etc...).

Aucun de vos noms de fichiers ne doit contenir de caractères d'espacements, accentués ou non-imprimables.

Votre archive devra contenir à sa racine un fichier nommé `AUTHORS`. Ce fichier devra au moins contenir les prénoms, les nom et les email des membre de votre équipe sous la forme :

```
* Prénom1 Nom1 <email1@entreprise1.dom>
* Prénom2 Nom2 <email2@entreprise2.dom>
[...]
```

Les lignes ne correspondant pas à cette nomenclature sont ignorées

Exemple de création de rendu :

```
make clean
echo '* Jérôme Pouiller <j.pouiller@sysmic.fr>' >> AUTHORS
echo '* Thibaud Hilaire <t.hilaire@sysmic.fr>' >> AUTHORS
cd ..
tar cvzf TR-pouiller.tar.gz TR-pouiller
```

Vous devrez envoyer votre rendu par e-mail avant le 31 Décembre 1999 à 23h59 à [<rendus@sysmic.fr>](mailto:rendus@sysmic.fr). Votre e-mail ne devra comporter que votre rendu comme unique pièce jointe.

Le sujet du mail devra suivre la nomenclature suivante : [TR] nom où nom est le nom de votre *contact administratif* (exemple : [TR] pouiller).

Enfin, le corps du mail devra se conformer au même formalisme que le fichier AUTHORS.

Exemple :

Veillez trouver en pièce jointe le rendu de projet temps réel de :

* Jérôme Pouiller <j.pouiller@sysmic.fr>

* Thibaud Hilaire <t.hilaire@sysmic.fr>

5 Cordialement,

--

Jérôme Pouiller, Sysmic

Expert Linux embarqué