HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Yann Thoma

Programmation Temps Réel (PTR)
Semestre automne 2020-2021
Contrôle continu 2
06.01.2020

Prénom: Denis

Nom: Bourgin

— Aucune documentation n'est permise, y compris la feuille de vos voisins
— La calculatrice n'est pas autorisée
— Aucune réclamation ne sera acceptée en cas d'utilisation du crayon
— Ne pas utiliser de couleur rouge

Question	Points	Score
1	10	8
2	9	9
3	8	—
4	8	8
5	8	2
6	7	7
Total:	~~50~~ 42	34

Note: 5

APIs

Fonction	Description
`int rt_event_create(RT_EVENT *event, const char *name, unsigned long ivalue, int mode)`	Création d'un masque d'événements
`int rt_event_signal(RT_EVENT *event, unsigned long mask)`	Mise à un d'un ou plusieurs événements
`int rt_event_wait(RT_EVENT *event, unsigned long mask, unsigned long *mask_r, int mode, RTIME timeout)`	Attente sur un ou plusieurs des événements
`int rt_event_clear(RT_EVENT *event, unsigned long mask, unsigned long *mask_r)`	Remise à zéro d'un ou plusieurs événements
`int rt_event_delete(RT_EVENT *event)`	Destruction d'un masque d'événements
`int rt_mutex_create(RT_MUTEX *mutex,const char *name)`	Création d'un verrou (mutex)
`int rt_mutex_acquire(RT_MUTEX *mutex, RTIME timeout)`	Acquisition d'un verrou
`int rt_mutex_release(RT_MUTEX *mutex)`	Restitution d'un verrou
`int rt_mutex_delete(RT_MUTEX *mutex)`	Destruction d'un verrou
`int rt_sem_create(RT_SEM *sem, const char *name, unsigned long icount, int mode)`	Création d'un sémaphore
`int rt_sem_p(RT_SEM *sem, RTIME timeout)`	Réservation d'une entrée de sémaphore
`int rt_sem_v(RT_SEM *sem)`	Libération d'une entrée de sémaphore
`int rt_sem_delete(RT_SEM *sem)`	Destruction d'un sémaphore
`int rt_cond_create(RT_COND *cond, const char *name)`	Création d'une variable condition
`int rt_cond_wait(RT_COND *cond, RT_MUTEX *mutex, RTIME timeout)`	Suspend la tâche jusqu'à ce que la variable condition soit signalée
`int rt_cond_signal(RT_COND *cond)`	Signalement d'une variable condition
`int rt_cond_broadcast(RT_COND *cond)`	Signalement d'une variable condition pour toutes les tâches en attente
`int rt_cond_delete(RT_COND *cond)`	Destruction d'une variable condition
`int rt_queue_create(RT_QUEUE *q, const char *name, size_t poolsize, size_t qlimit, int mode)`	Création d'une file de messages
`void *rt_queue_alloc(RT_QUEUE *q, size_t size)`	Allocation d'un message (géré par un pool de messages)
`int rt_queue_send(RT_QUEUE *q, void *buf, size_t size, int mode)`	Envoi d'un message
`size_t rt_queue_receive(RT_QUEUE * q, void ** bufp, RTIME timeout)`	Réception d'un message
`int rt_queue_free(RT_QUEUE *q, void *buf)`	Restitution de la place mémoire utilisée pour le message (côté récepteur)
`int rt_queue_delete(RT_QUEUE *q)`	Destruction d'une file de messages
`int rt_queue_write(RT_QUEUE *q,const void *buf, size_t size,int mode)`	Ecriture d'un message
`size_t rt_queue_read(RT_QUEUE *q,void *buf, size_t size,RTIME timeout)`	Lecture d'un message
`int rt_heap_create(RT_HEAP *heap, const char *name, size_t heapsize, int mode)`	Création d'un pool de bloc dans le tas (mode=H_PRIO ou H_FIFO)
`int rt_heap_alloc(RT_HEAP *heap, size_t size, RTIME timeout, void **blockp)`	Allocation d'un bloc mémoire à partir du pool précédemment créé
`int rt_heap_free(RT_HEAP *heap, void *block)`	Restitution d'un bloc mémoire
`int rt_heap_delete(RT_HEAP *heap)`	Destruction d'un pool mémoire

Denis

Question 1: 10 points

Répondez aux questions suivantes :

1. Pour protéger une section critique il est possible d'utiliser un mutex ou un sémaphore initialisé à 1. Dans un contexte temps-réel, qu'est-ce qui peut vous faire choisir une option plutôt que l'autre ?

   Inversion de Priorité: les mutex implement un protocol pour l'inversion des priorité afin d'avoir un système plus deterministe. ✓

2. Quel est l'avantage du protocole PCP sur le protocole PIP ?

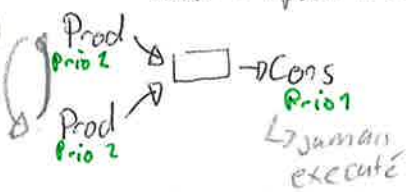
   + PCP empeche plus le deadlock ✓
   + ~~Système 2 plus déterministe~~
     le temps de blocage est bornée

3. Malgré ces avantages, pourquoi le protocole PCP n'est-il pas utilisé plus souvent ?

   il est nécessaire de savoir quelle tâche utilise quelle mutex. → Difficile à mettre en place. ✓

4. Dans le cas d'une implémentation du protocole producteur-consommateur avec des sémaphores, est-ce que vous utiliseriez des sémaphores avec une file d'attente FIFO ou par priorité ? (Justifiez)

   ~~Par priorité~~ Par fifo. Sinon si les Producteurs sont plus prioritaire le buffer va jamais ce vider. (Admis quils tournent en boucle ou à une periode très courte) ✓



5. Un pool mémoire peut être utilisé si de l'allocation dynamique est nécessaire. Pourquoi utiliser un pool mémoire plutôt que les fonctions d'allocation traditionnelles ?

   les fonction d'allocation standart ne sont pas des fonctions du contexte tant réel. Elle peuvent être bloquante. pas bloquantes, mais non bornées -2

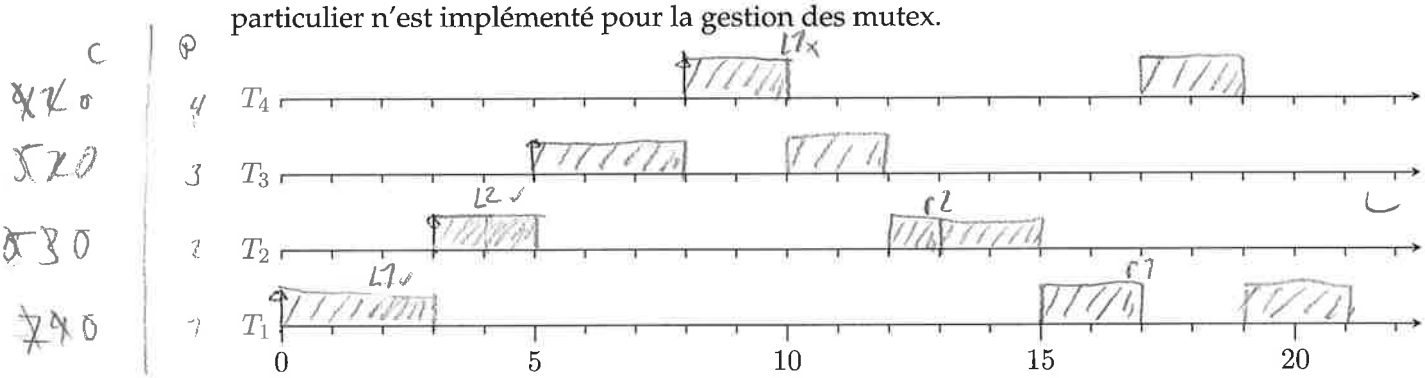
Question 2: 9 points

Soit les quatre tâches apériodiques suivantes :

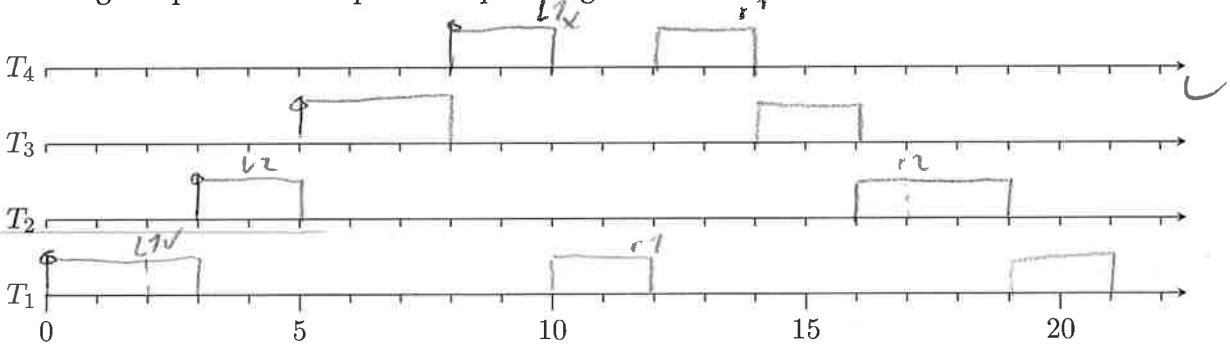
Tâche	Priorité	Coût	Arrivée	req. m1	rel. m1	req. m2	rel. m2
T_1	1	7	0	2	3	-	-
T_2	2	5	3	-	-	1	2
T_3	3	5	5	-	-	-	-
T_4	4	4	8	2	2	-	-

La valeur req indique à la fin de quel quantum de temps la tâche tente d'acquérir les mutex m1 et m2[1]. La valeur rel indique combien de quantums de temps la tâche a besoin de garder le mutex. Il s'agit du temps effectif de traitement.

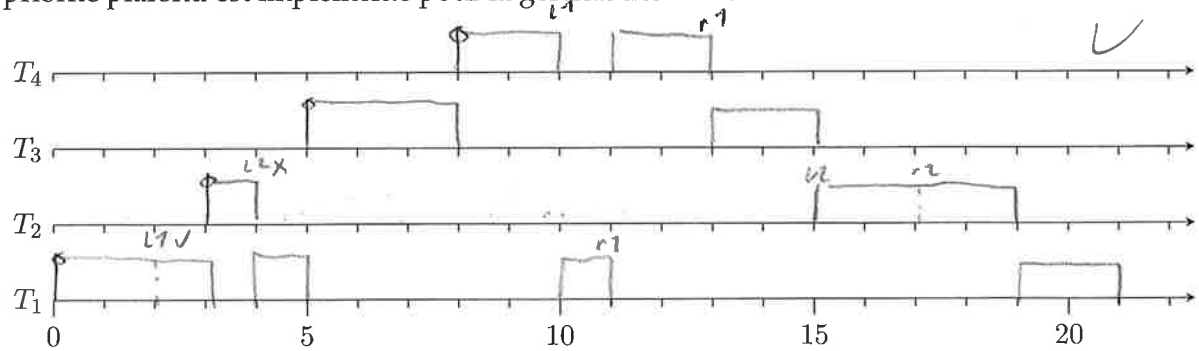
Construisez le chronogramme d'exécution des tâches, en considérant qu'aucun protocole particulier n'est implémenté pour la gestion des mutex.



Construisez le chronogramme d'exécution des tâches, en considérant que le protocole d'héritage de priorité est implémenté pour la gestion des mutex.



Construisez le chronogramme d'exécution des tâches, en considérant que le protocole de priorité plafond est implémenté pour la gestion des mutex.



1. Toute ressemblance avec des métros lausannois est totalement fortuite.

Denis

Question 3: 8 points

Soient les deux tâches ci-dessous. Est-ce que la gestion de la concurrence vous semble correcte ? Si oui, justifiez, et si non, justifiez et corrigez le code.

```
1  #define NBLOG 1000
2
3  log_t log1[NBLOG];
4  int nbLogs1;
5
6  log_t log2[NBLOG];
7  int nbLogs2;
8
9  RT_MUTEX mutex1;
10 RT_MUTEX mutex2;
11
12 void tacheA(void *cookie) {
13   log_t uneValeur;
14   for(int i = 0; i < NBLOG / 2; i++) {
15     getValeurA(&uneValeur);
16     rt_mutex_acquire(&mutex1,TM_INFINITE);
17     log1[nbLogs1] = uneValeur;
18     rt_mutex_release(&mutex1);
19     rt_mutex_acquire(&mutex2,TM_INFINITE);
20     log2[nbLogs2] = uneValeur;
21     rt_mutex_acquire(&mutex1,TM_INFINITE);
22     nbLogs1++;
23     rt_mutex_release(&mutex1);
24     nbLogs2++;
25     rt_mutex_release(&mutex2);
26     rt_task_wait_period();
27   }
28 }
29
30 void tacheB(void *cookie) {
31   log_t uneValeur;
32   for(int i = 0; i < NBLOG / 2; i++) {
33     getValeurB(&uneValeur);
34     rt_mutex_acquire(&mutex2,TM_INFINITE);
35     rt_mutex_acquire(&mutex1,TM_INFINITE);
36     log2[nbLogs2] = uneValeur;
37     log1[nbLogs1] = uneValeur;
38     nbLogs2++;
39     rt_mutex_release(&mutex2);
40     nbLogs1++;
41     rt_mutex_release(&mutex1);
42     rt_task_wait_period();
43   }
44 }
45
46 void initialize() {
47   rt_mutex_create(&mutex1, "MUTEX1");
48   rt_mutex_create(&mutex2, "MUTEX2");
49 }
```

Il ne faut pas relâcher le mutex1 dans tacheA, sinon on peut avoir écrasement des données car l'incrémentation n'est pas dans la même section critiques que la modification du tableau. Il faut ensuite évidemment faire attention à l'ordre de réquisition des mutex.

Question 4: 8 points

Soient les trois tâches ci-dessous. Les tâches A et B remplissent un tableau de log, et la tâche C va afficher le contenu de ces logs. Les tâches A et B sont périodiques et de priorité supérieure à C, cette troisième tâche étant apériodique. Analysez ce code. Est-il totalement correct? Justifiez votre réponse, et corrigez le code si nécessaire.

```
1  #define NBLOG 1000
2
3  log_t log[NBLOG];
4  int nbLogs;
5
6  RT_EVENT event;
7  RT_MUTEX mutex;
8
9  void tacheA(void *cookie) {
10   log_t uneValeur;
11   for(int i=0;i<NBLOG/2;i++) {
12     getValeurA(&uneValeur);
13     rt_mutex_acquire(&mutex,TM_INFINITE);
14     log[nbLogs]=uneValeur;
15     rt_event_signal(&event, 1);
16     nbLogs++;
17     rt_mutex_release(&mutex);
18     rt_task_wait_period();
19   }
20 }
21
22
23 void tacheB(void *cookie) {
24   log_t uneValeur;
25   for(int i=0;i<NBLOG/2;i++) {
26     getValeurA(&uneValeur);
27     rt_mutex_acquire(&mutex,TM_INFINITE);
28     log[nbLogs]=uneValeur;
29     rt_event_signal(&event, 1);
30     nbLogs++;
31     rt_mutex_release(&mutex);
32     rt_task_wait_period();
33   }
34 }
35
36 void tacheC(void *cookie) {
37     log_t uneValeur;
38     int index=0;
39     while(1) {
40         unsigned long unused;
41         rt_event_wait(&event, 1, &unused, EV_ALL, TM_INFINITE);
42         rt_event_clear(&event, 1, &unused);
43         printValeur(log[index++]);
44     }
45 }
```

RT_COND cond;

rt_cond_signal(&cond);

rt_cond_signal(&cond

rt_mutex_acquire(&mutex,TM_INFINITE);

rt_mutex_acquire(&mutex,TM_INFINITE);
if(index == nblogs){
rt_cond_wait(&cond, &mutex, TM_INFINITE);

rt_mutex_release(&mutex);

le problème c'est que si A et B ont déjà produit 10 logs avant la première execution de C, alors C va que lire un seul log. le prochain sera fait seulement au prochain signal. Il faut que s'endormir si on a lu tous les logs

En plus:
Ca serait mieux d'utiliser une var. cond. Comme ca on pourrais sécuriser l'accès au log par un mutex (En vert)

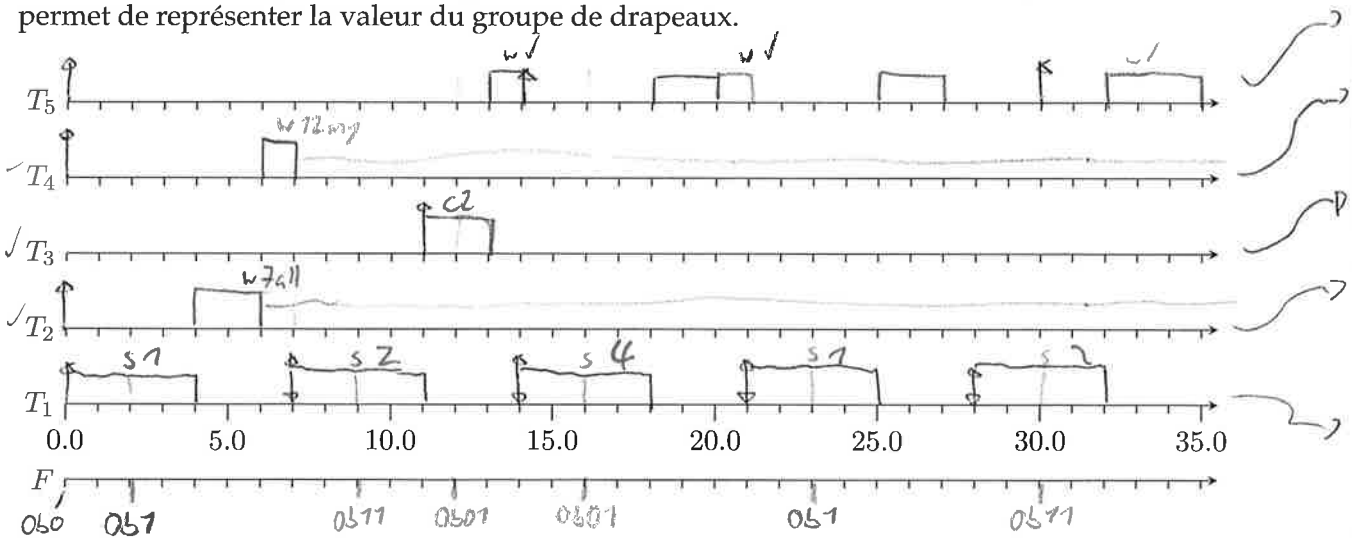
Denis

Question 5: 8 points

Soient les tâches suivantes :

- T_1
  - Périodique de période 7
  - Démarrage au temps 0
  - Coût 4
  - Priorité 5
  - A la fin de son deuxième slot elle effectue `signal((1 << (occurence % 3))`, où `occurence` est le numéro d'occurence, en commençant par 0
- T_2
  - Tâche apériodique
  - Démarrage au temps 0
  - Coût 4
  - Priorité 4
  - A la fin de son deuxième slot elle effectue `wait(7, EV_ALL)`
- T_3
  - Tâche apériodique
  - Démarrage au temps 11
  - Coût 2
  - Priorité 3
  - A la fin de son premier slot effectue `clear(2)`
- T_4
  - Tâche apériodique
  - Démarrage au temps 0
  - Coût 2
  - Priorité 2
  - A la fin de son premier slot effectue `wait(12, EV_ANY)`
- T_5
  - Tâche périodique de période 15
  - Démarrage au temps 0
  - Coût 3
  - Priorité 1
  - A la fin de son permier slot effectue `wait(7, EV_ANY)`

Proposez le chronogramme correspondant à l'exécution de ces tâches. La ligne F vous permet de représenter la valeur du groupe de drapeaux.



1<< (2%3) = 4 ...

Question 6: 7 points

Le code suivant souffre d'un problème : les logs générés par la tâche A ne sont pas forcément consommés par les deux tâches B et C. Expliquez pourquoi, et proposez une solution. La solution peut être une modification du code ou une description textuelle. Dans ce deuxième cas il faut qu'elle soit assez précise pour être validée (les numéros de ligne peuvent être utilisés au besoin).

```
1  RT_QUEUE mailboxB, mailboxC;
2  RT_TASK Ta, Tb, Tc, Td;
3
4  #define PERIOD_TACHEA 5*MS
5  #define PERIOD_TACHED 8*MS
6
7  void tacheA(void *cookie) {
8    log_t uneValeur;
9    char *mess;
10   for(int i=0;i<NBLOG;i++) {
11     getValeurA(&uneValeur);
12     mess= rt_queue_alloc(&mailboxB,sizeof(log_t));
13     memcpy(mess,&uneValeur,sizeof(log_t));
14     rt_queue_send(&mailbox, mess, sizeof(log_t), Q_BROADCAST);
15     rt_task_wait_period();
16   }
17 }
18
19 void tacheD(void *cookie) {
20   for(int i=0;i<NBLOG;i++) {
21     unTraitement();
22     rt_task_wait_period();
23   }
24 }
25
26 void tacheB(void *cookie) {
27   log_t uneValeur;
28   char *mess;
29   while(1) {
30     rt_queue_receive(&mailboxB,(void **) &mess, TM_INFINITE);
31     memcpy(&uneValeur,mess,sizeof(log_t));
32     rt_queue_free(&mailboxB,mess);
33     printValeur(uneValeur);
34   }
35 }
36
37 void tacheC(void *cookie) {
38   log_t uneValeur;
39   char *mess;
40   while(1) {
41     rt_queue_receive(&mailboxC,(void **) &mess, TM_INFINITE);
42     memcpy(&uneValeur,mess,sizeof(log_t));
43     rt_queue_free(&mailboxC,mess);
44     sendValeur(uneValeur);
45   }
46 }
47
48 void init_module() {
49   rt_queue_create(&mailboxB,"MyMailboxB", 3*sizeof(log_t), 3, Q_PRIORITY);
50   rt_task_create(&Ta,"TacheA", STACK_SIZE,4,0);
51   rt_task_create(&Td,"TacheD", STACK_SIZE,3,0);
52   rt_task_create(&Tb,"TacheB", STACK_SIZE,2,0);
53   rt_task_create(&Tc,"TacheC", STACK_SIZE,1,0);
54   rt_task_set_periodic(&Ta,TM_NOW,PERIOD_TACHEA);
55   rt_task_set_periodic(&Td,TM_NOW,PERIOD_TACHED);
56   rt_task_start(&Ta,tacheA,NULL);
57   rt_task_start(&Td,tacheD,NULL);
58   rt_task_start(&Tb,tacheB,NULL);
59   rt_task_start(&Tc,tacheC,NULL);
60 }
```

mailboxB, mailboxC;

rt_queue_alloc(&mailboxB, sizeof(log_t))

Q_BROADCAST → Q_NORMAL

2.0 mess = rt_queue_alloc(&mailboxC, sizeof(log_t));
memcpy(mess, &uneValeur, sizeof(log_t));
rt_queue_send(&mailbox, mess, sizeof(log_t), Q_NORMAL);

rt_queue_create(&mailboxC, "MyMailboxC", 3*sizeof(log_t), 3, Q_PRIORITY);

Denis

mettre la feuille de Q5 ici

cont 3 prio 1	fin slot 1	wait(0b111, ANY)
cont 2 prio 2	fin slot 1	wait(0b1000, ANY)
cont 2 prio 3	fin slot 1	clear(0b10)
cont 4 prio 4	fin slot 2	wait(0b111, ALL)
cont: 4 prio 5	fin slot 2	sig((1<<acc)%3) 0: sig 0b1 1: sig 0b10 2: sig 0b00 3: sig 0b1 4: sig 0b10

Q6)

Le problème est dans le fonctionnement du send Broadcast:

- Si aucune tâche attend sur un receive: le message reste jusqu'à ce que une tâche va le lire
- Si une seul tâche attend sur un receive: elle va lire le message et le supprimer
- Si les deux tâches sont en attende sur un receive elle vont les deux recevoir le message.

Dans notre cas on peut pas savoir (ou être sûr) que les deux tâches B et C sont en attende sur le receive. Si ils le sont pas, alors un des deux vas pas recevoir le message

Une possibilité serait de jouer avec le périodes et priorités afin de savoir qu'elle vont toujours être sur le receive au moment du send.
B et C

Mais c'est du bricolage faut pas faire ça 6

La bonne pratique c'est d'utiliser deux boîte aux lettres. Une par tâche B et C.