Exemple : création et activation d'une tâche (one-shot)

Dans cet exemple il s'agit de créer et de lancer une tâche qui exécute une séquence simple d'instructions. La tâche est ensuite détruite naturellement quand le pme ppal qui l'a crée se termine. Par exemple, créer une tâche de forte priorité qui affiche « hello World ».
Créer la tâche dans le programme principal avec le mode T_JOINABLE et attendre que la tâche ait terminé son exécution avec la fonction rt_task_join(&task_desc).
Utiliser la fonction rt_task_spawn () pour créer et activer la tâche.

Penser à récupérer systématiquement le code retourné par la fonction de création de la tâche afin de s'assurer que cela s'est bien passé.

Rque : créer une tâche Xenomai qui affiche un message n'a aucun intérêt d'un point de vue temps réel mais permet simplement de présenter la structure d'une application à travers un exemple basique.

```
#include <stdio.h>
#include <sys/mman.h>
#include <native/task.h>
RT_TASK task_desc;
void task (void *cookie)      /* ... "cookie" should be NULL ... */
{
   printf("Hello world !\n");
}
int main (void)
{
int err;

//disable memory swap
  mlockall( MCL_CURRENT | MCL_FUTURE );

//create task 'mytask'
   if ((err=rt_task_spawn(
         &task_desc,    //descripteur de la tâche
         "my task",     //nom de la tâche
         0,             //taille de la pile, si 0, autogéré
         99,            //priorité (0 = faible, 99 = forte)
         T_JOINABLE,    //indique qu'on attend la fin de la tâche
         &task,         //la fonction a exécuter
         NULL           //pas de paramètres de la fonction
         ))!=0)
   {
      fprintf (stderr,"rt_task_spawn error %d\n",err);
      return 1;
   }
   //wait for the function to complete
   rt_task_join(&task_desc);
   return 0;
}
```

Le temps d'accès à la mémoire doit être constant. Or, ce n'est pas garanti lorsque le swap disque est activé. Il faut donc impérativement le désactiver avec la fonction mlockall.

le programme principal va attendre que la tâche "my task" se termine grâce à la fonction rt_task_join.
Pour que la tâche soit suspendue au moment de son démarrage, il faut mettre T_SUSP. Il faudra ensuite la redémarrer avec la fonction rt_task_resume.

Exemple (variante) : création et activation d'une tâche avec passage de paramètre à l'appel

Reprenons l'exemple précédent avec une variante : au lieu de prédéfinir la chaîne de caractère à afficher dans la tâche ('Hello World'), on la lui passe en argument.

On applique les bonnes pratiques de programmation et on note symboliquement les arguments des fonctions.

```
#include <stdio.h>
#include <sys/mman.h>
#include <native/task.h>

#define TASK_PRIO     99 /* Highest RT priority */
#define TASK_MODE    T_JOINABLE
#define TASK_STKSZ    0 /* Stack size (use default one) */

RT_TASK task_desc;

void task_body (void *cookie) {
                printf("%s\n", (char*) cookie);
}

int main (int argc, char**argv)
{
   int err;
   //disable memory swap
   mlockall( MCL_CURRENT | MCL_FUTURE );

   char my_string[50] = "Hello world !";

    err = rt_task_spawn(&task_desc, "MyTaskName", TASK_STKSZ, TASK_MODE, &task_body , my_string);

    if(!err) {  rt_task_join(&task_desc);  }

return 0;
}
```

Exemple : création, activation et destruction d'une tâche

Dans cet exemple il s'agit de créer et de lancer une tâche qui s'exécute mais ne se termine pas, puis de la détruire quand l'utilisateur le souhaite. Par exemple, cette tâche, de forte priorité, affiche « hello World » (voir premier exemple) puis se met en sommeil indéfiniment (rt_task_sleep_until(TM_INFINITE)).

NB : Utiliser pour changer les fonctions rt_task_create () et rt_task_start () pour respectivement créer et activer la tâche.
Penser à récupérer systématiquement le code retourné par les fonctions.
On applique toujours les bonnes pratiques de programmation et on note symboliquement les arguments des fonctions.

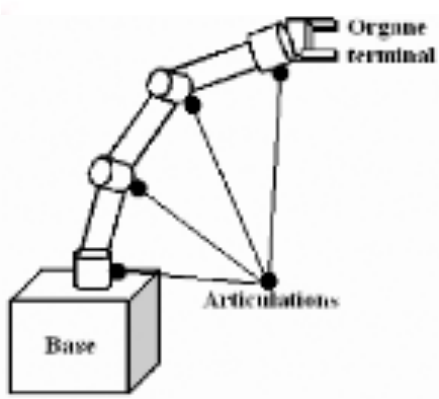
Définir spécifiquement une fonction de destruction de la tâche en sommeil qui sera appelée par l'utilisateur.

```
#include <sys/mman.h>
#include <native/task.h>
#include <unistd.h>
#include <stdio.h>

#define TASK_PRIO       99 /* Highest RT priority */
#define TASK_MODE       0 /* No flags*/
#define TASK_STKSZ      0 /* Stack size (use default one) */

RT_TASK task_desc;

void task_body (void *cookie) {
                rt_printf("hello world !");
                /* Wait infinitely */
                rt_task_sleep_until (TM_INFINITE);
}

int main (int argc, char**argv) {
                int err;
                /* disable memory swap */
                mlockall (MCL_CURRENT|MCL_FUTURE);
                err = rt_task_create (&task_desc,"MyTaskName", TASK_STKSZ, TASK_PRIO, TASK_MODE);
                if(!err) {   rt_task_start (&task_desc, & task_body, NULL);   }

                pause;   // équivalent à rt_task_sleep_until (TM_INFINITE);
                /* cancellation of the task */
                rt_task_delete (&task_desc);

}
```

Exemple : création, activation et destruction de plusieurs tâches – illustration de la réentrance

Créer trois tâches de même priorité, « poignet », « coude » et « épaule » qui vont piloter en temps réel et de façon autonome et indépendante le déplacement d'un bras de robot anthropomorphe. Le mouvement du bras résulte de la coordination fine des mouvements des trois articulations.

Coude, épaule et poignet fonctionnent sur le même principe.



NB :
Utiliser la fonction rt_task_spawn () pour créer et activer les tâches.
Utiliser la terminaison adoptée dans le TP.

```
RT_TASK t_Coude, t_Poignet, t_Epaule;

Double pos_coude, pos_poignet, pos_epaule;

Void Deplacer_Articulation (void * pos_articulation) {
        /* assure le déplacement de l'articulation vers la position à rejoindre */
}

int main (int argc, char**argv) {
        /* disable memory swap */
        mlockall (MCL_CURRENT|MCL_FUTURE);

        rt_task_spawn(&t_Coude, "Coude", TASK_STKSZ, TASK_MODE, & Deplacer_Articulation ,
&pos_coude);
        rt_task_spawn(&t_Poignet, "Poignet", TASK_STKSZ, TASK_MODE, & Deplacer_Articulation ,
&pos_poignet);
        rt_task_spawn(&t_Epaule, "Epaule", TASK_STKSZ, TASK_MODE, & Deplacer_Articulation ,
&pos_epaule);

        pause;

        /* cancellation of the task */
        rt_task_delete (& t_Coude);
        rt_task_delete (& t_Poignet);
        rt_task_delete (& t_Epaule);
}
```

Exemple : créer et activer une tâche périodique

Reprendre l’exemple du bras de robot et rajouter un capteur de position sur chaque articulation, qui donne à chaque seconde la position angulaire de l’articulation.

```
#define …

RT_TASK t_Coude, t_Poignet, t_Epaule,  t_Capt_Coude, t_Capt_Epaule, t_Capt_Poignet;
double pos_coude, pos_epaule, pos_poignet;

void Deplacer_Articulation (void * pos_articulation) {/* assure le déplacement de l'articulation vers la position à
rejoindre */ }

void Capturer_Pos (void * pos_articulation) {
        if ( rt_task_set_periodic  (NULL,            /* the current task  or rt_task_self() */
                                    TM_NOW,        /* delay before release, TM_NOW = none */
                                    ONE_SECOND   /* this value is in nanosec */
                                    ) != 0)
        { printf("rt_task_set_periodic error\n");  }
        while (1) {
                rt_task_wait_period(NULL); /* Attente de l'activation périodique sans détection d'overruns */
                /* lire la position courante pos_articulation et la mettre a jour dans l'application*/
                }
}
int main (int argc, char *argv[]) {
        mlockall (MCL_CURRENT|MCL_FUTURE);
        rt_task_spawn(&t_Coude, "Coude", TASK_STKSZ, TASK_MODE, & Deplacer_Articulation , pos_coude);
        ….
        /* puis creer et activer les trois capteurs à l'identique*/
        rt_task_spawn(&t_Capt_Coude, "Capt_Coude", TASK_STKSZ, TASK_MODE, & Capturer_Pos, pos_coude);
        …
        rt_task_sleep_until (20*ONE_SECOND);
        /* cancelation of all the tasks */
        ……
}
```

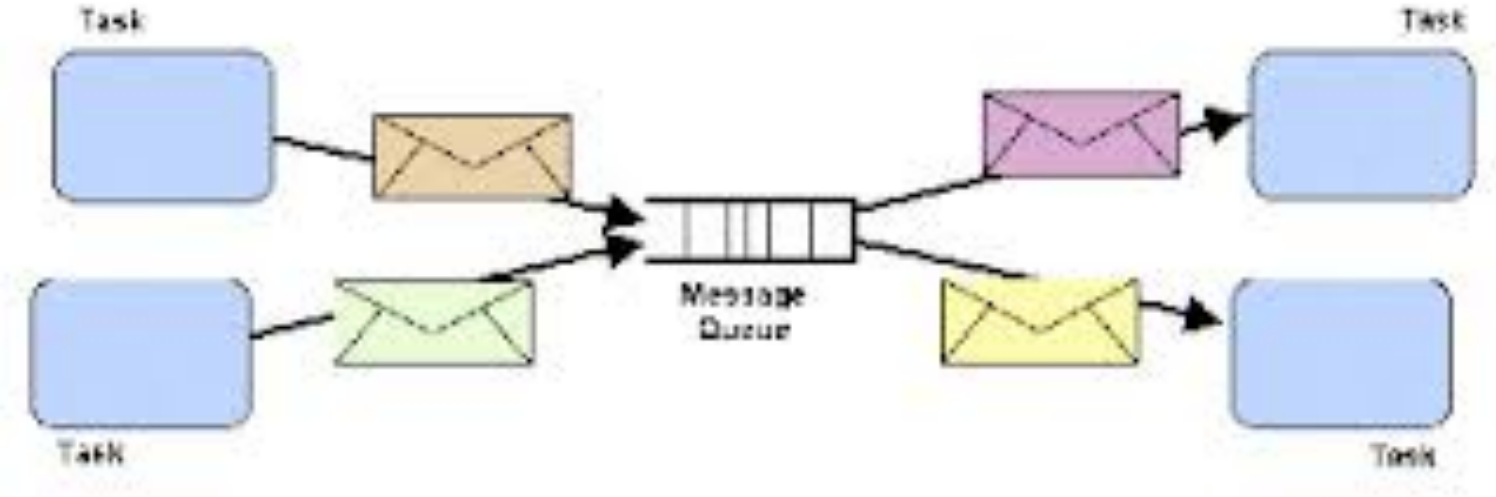
Exemple : utilisation d’un mutex

Reprendre l’exemple du bras de robot et rajouter la protection des variables de position des articulations

```
RT_TASK t_Coude, t_Poignet, t_Epaule,  t_Capt_Coude, t_Capt_Epaule, t_Capt_Poignet;
RT_MUTEX mutex_coude, mutex_epaule, mutex_poignet;
double pos_coude, pos_epaule, pos_poignet;

void Deplacer_Articulation (void * args) {/* assure le déplacement de l'articulation vers la position à rejoindre */
                        rt_mutex_acquire(&mutex_pos, TM_INFINITE);
                            /* lire la position courante pos_articulation */
                         rt_mutex_release(&mutex_pos);
                        /* et actionner l'articulation*/}

void Capturer_Pos (void * args) { /* passer une structure avec pos_articulation et  mutex_pos qui la protège */
            if ( rt_task_set_periodic  (NULL,            /* the current task  or rt_task_self() */
                                        TM_NOW,         /* delay before release, TM_NOW = none */
                                        ONE_SECOND   /* this value is in nanosec */
                                         ) != 0)
            { printf("rt_task_set_periodic error\n");  }
            while (1) {
                        rt_task_wait_period(NULL); /* Attente de l'activation périodique sans détection d'overruns */
                        rt_mutex_acquire(&mutex_pos, TM_INFINITE);
                            /* lire la position courante pos_articulation et la mettre a jour dans l'application*/
                         rt_mutex_release(&mutex_pos);
                      }
}
int main (int argc, char *argv[]) {
            mlockall (MCL_CURRENT|MCL_FUTURE);
            rt_task_spawn(&t_Coude, "Coude", TASK_STKSZ, TASK_MODE, & Deplacer_Articulation , /*structure*/);
            ....
            rt_task_spawn(&t_Capt_Coude, "Capt_Coude", TASK_STKSZ, TASK_MODE, & Capturer_Pos, /*structure*/);
            …
            rt_mutex_create (&mutex_coude,"Mutex_Coude");
            ……
}
```

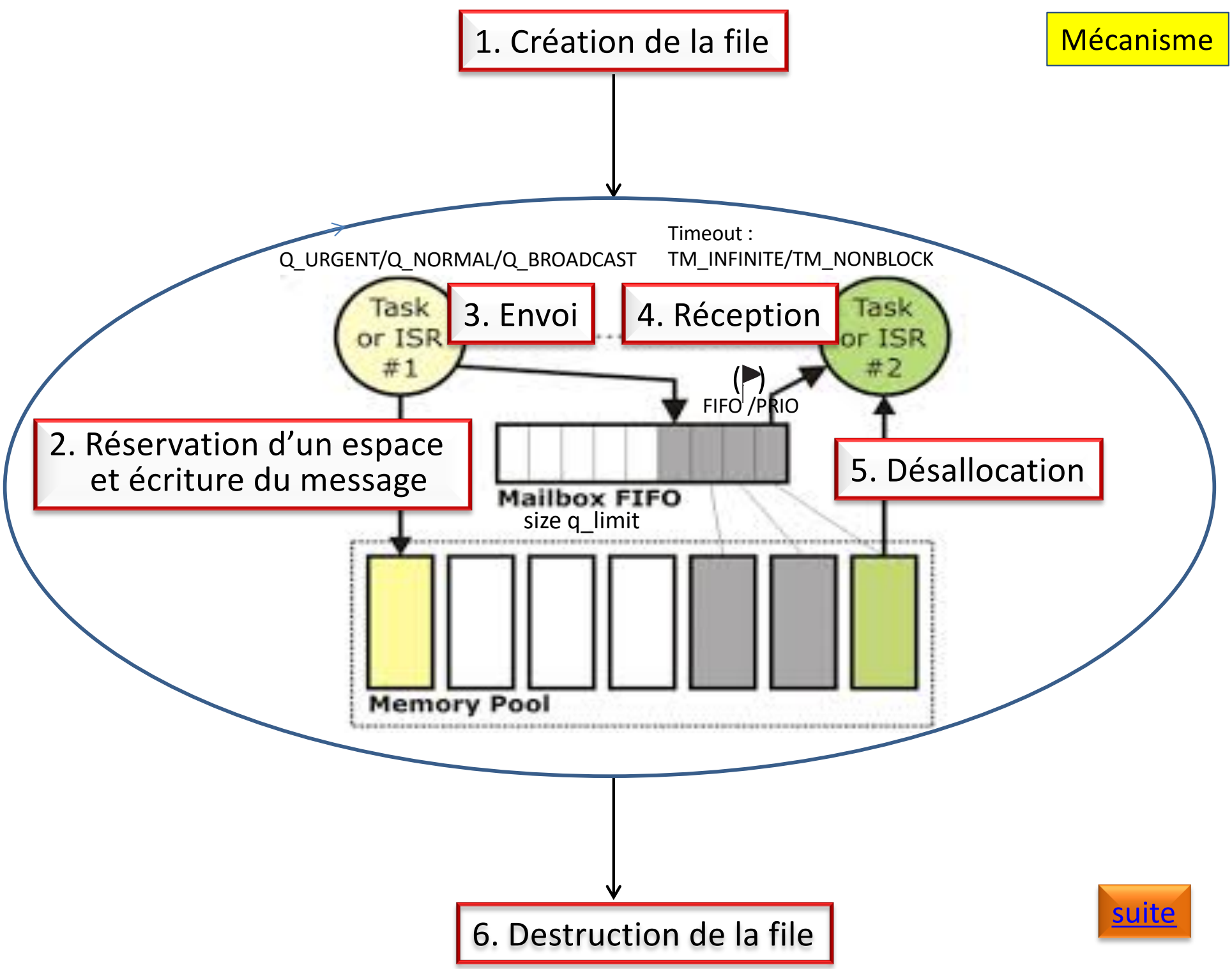
Communication par file de messages

Absorption différence rythme
Stockage des données sans écrasement
Synchronisation
Exclusion mutuelle
Priorisation des messages à l'envoi
Gestion des émetteurs/récepteurs en attente



Exemple : mailbox sous Gmail

Mécanisme



suite

Création de la file

```
int rt_queue_create ( RT_QUEUE* q,
                      const char * name,
                      size_t poolsize,
                      size_t qlimit,
                      int mode );
```

address of a queue descriptor

ASCII string - symbolic name

size (bytes) of the message buffer pool - fixe

maximum number of messages / Q_UNLIMITED

mode :

- Q_FIFO makes tasks **pend** in FIFO order on the queue for **consuming** messages.
- Q_PRIO makes tasks pend in priority order on the queue.

Create a message queue object that allows multiple tasks to exchange data through the use of variable-sized messages.
A message queue is created empty.

retour

Réservation d'un espace puis écriture du message

```
void  rt_queue_alloc ( RT_QUEUE* q,
                       size_t size );
```

address of a queue descriptor

requested size in bytes of the buffer

Allocate a message queue buffer from the queue's internal pool which can be subsequently filled by the caller (memcpy() ou strcopy()) then passed to rt_queue_send() for sending.

retour

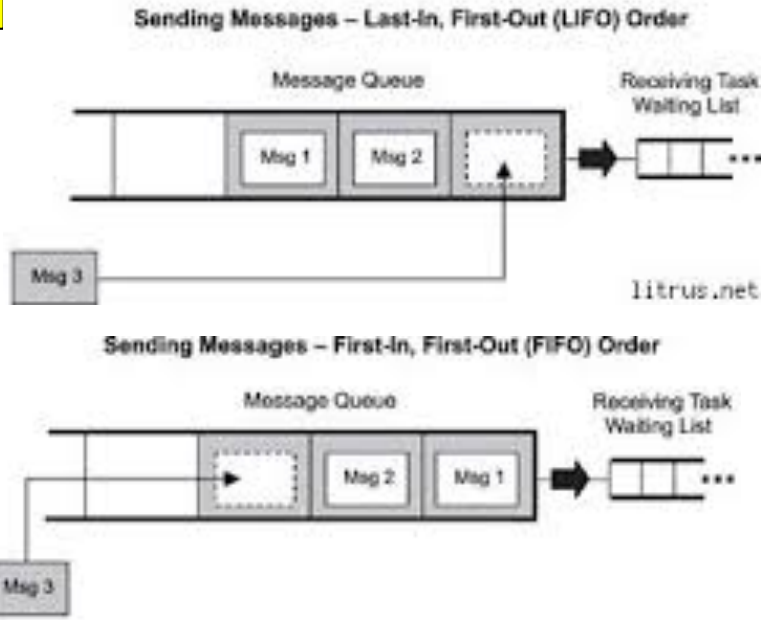
Envoi

```
int rt_queue_send (   RT_QUEUE* q,
                      void *mbuf,
                      size_t size,
                      int mode  );
```

address of a queue descriptor

address of the message buffer to be sent*

size in bytes of the message
(0: an empty message will be sent)



mode :

- Q_URGENT : message prepended to the message queue (LIFO ordering)
- Q_NORMAL : message appended to the message queue (FIFO ordering)
- Q_BROADCAST : message sent to all tasks currently waiting for messages.

Send a message to a queue.
The message must have been allocated by a previous call to rt_queue_alloc().
Returns the number of receivers which got awaken as a result of the operation

retour

Réception

```
ssize_t  rt_queue_receive (   RT_QUEUE* q,
                              void **bufp,
                              RTIME timeout  );
```

pointer to a memory location which will be written upon success with the address of the received message.

Timeout : number of clock ticks to wait for a message to arrive.

- TM_INFINITE : caller blocked indefinitely until some message is eventually available
- TM_NONBLOCK : no waiting if no message is available

Receive a message (next available) from a queue.
Unless otherwise specified, the caller is blocked for a given amount of time if no message is immediately available on entry.
The number of bytes available from the received message is returned upon success

retour

Désallocation

Once consumed, the message space should be freed using rt_queue_free().

```
void  rt_queue_free ( RT_QUEUE* q,
                      void *buf );
```

Releases a message buffer returned by rt_queue_receive() to the queue's internal pool.

retour

Destruction de la file

int rt_queue_delete (RT_QUEUE* q);

address of a queue descriptor

Delete a message queue.
Destroy a message queue and release all the tasks currently pending on it.
A queue exists in the system since rt_queue_create() has been called to create it, so this service must be called in order to destroy it afterwards.

retour

```
RT_TASK desc_envoi, desc_reception;
RT_QUEUE queue;

#define MSG_QUEUE_SIZE 10 /* nbre max de messages dans la file */

void t_envoi (void *arg)
{           DMessage dataAEnvoyer = …;                  /* à faire en boucle */
            void *msg;
            int err;
            msg = rt_queue_alloc (&queue, sizeof (DMessage));
  2         //on copie le message  a l'endroit ou pointe *msg
            memcpy (msg, &dataAEnvoyer, sizeof (DMessage));
            //Q_NORMAL  :a la file, Q_URGENT : en debut de file
  3         if ((err = rt_queue_send (&queue, msg, sizeof (DMessage), Q_NORMAL)) < 0)     /* posté FIFO*/

                        {
                        rt_printf("Error msg queue send: %s\n", strerror(-err));
                        }
}
void t_reception (void *arg)
{           DMessage *msg;                              /* à faire en boucle */
  4         int err;
            if ((err = rt_queue_receive (&queue ,&msg,  TM_INFINITE)) > 0)     /* timeout bloquant*/
                        {
  5                     rt_printf("message recu\n");
                        rt_queue_free (&queue, msg);
                        }
}
```

```
int main (int argc, char *argv[])
{
        int err;
        mlockall(MCL_CURRENT|MCL_FUTURE);
        err = rt_task_create (…….t_envoi ……..);
        err2 = rt_task_create (…….t_reception……..);

        /* Creation de la file de messages  : gestion des files d'attente fifo ou prio*/
1       if (err = rt_queue_create (&queue, « »ma_file", MSG_QUEUE_SIZE*sizeof(DMessage), /*pool*/
                                   MSG_QUEUE_SIZE,  /*nbre max messages*/
                                   Q_FIFO))              / * gestion de la file des tâches bloquées en lecture*/
                {
                rt_printf("Error msg queue create: %s\n", strerror(-err));
                }
        if (!err)
                rt_task_start( t_envoi……);
        if (!err2)
                rt_task_start (t_reception ……..);
        /* ... */
        pause;

        rt_task_delete (desc_envoi);
        rt_task_delete (desc_reception);
        */ détruire la file */
6       rt_queue_delete ( &queue);
}
```

Exemple de synchronisation

EXEMPLE 1 : T1 doit attendre que T2 et T3 aient terminé

```
RT_SEM sem;

void  T1 (void *arg) {
            rt_sem_p(&sem,TM_INFINITE);
            rt_sem_p(&sem,TM_INFINITE);
            // T2 et T3 ont terminé
}
void  T2 (void *arg) {
            // traitement
            rt_sem_v(&sem);
}
void  T3 (void *arg) {
            // traitement
            rt_sem_v(&sem);
}
void main() {
            rt_sem_create(&sem,"Semaphore", 0, S_PRIO);
            // création des taches
}
```

Exemple de synchronisation

EXEMPLE 2 : T1 permet à T2 et T3 de continuer

```
RT_SEM sem;

void T1 (void *arg) {
        // traitement
        rt_sem_v(&sem);
        rt_sem_v(&sem);
  }
void T2 (void *arg) {
        rt_sem_p(&sem,TM_INFINITE);
        // traitement
  }
void T3 (void *arg) {
        rt_sem_p(&sem,TM_INFINITE);
        // traitement
  }
void main() {
        rt_sem_create(&sem,"Semaphore",0, S_PRIO);
        // création des taches
}
```