Files de messages (message queues)

- permet d'échanger des messages entre processus (plutôt qu'une séquence d'octets, comme avec les tubes)
- on peut associer un type (= entier) à un message; le type peut être utilisé lors de la commande de réception (ordre FIFO par défaut)
- désignation

  - externe: *clé* (désignation utilisée lors de la création/ouverture d'une file de messages; permet à deux processus, en utilisant la même clé, de désigner la même file de messages)

  - interne: *identificateur* (désignation utilisée par un processus après la création/ouverture d'une file de message par une processus)
- protection: basée sur UID et GID (comme pour les fichiers)
- mêmes mécanismes de désignation et protection que pour les *sémaphores* et les *segments de mémoire partagée* (mais 3 espaces de désignation distincts)

NB: les files de message n'existent pas dans POSIX.1

existent sous une forme différente dans POSIX.4

Création d'une file de messages

```
int msgget (key_t clé, int option)
```

- crée la file de message de nom `clé` si celle-ci n'existe pas encore; si la file existe déjà, permet d'obtenir l'identificateur de la file, que le processus utilisera par la suite pour manipuler la file de messages (ouverture de la file de messages). La clé peut être obtenue grâce à la fonction `ftok( )`. Le type `key_t` est le type `int`.
- `option`: séquence de bits; pour la création on doit avoir:

  `option` = bit `IPC_CREATE` à 1 + droits d'accès.

  Exemple: `id = msgget(clé, 0666 | IPC_CREATE)`
- l'appel retourne l'identificateur permettant par la suite au processus de désigner la file de messages;

NB: la commande `ipcs` permet de consulter les tables IPC du système et de visualiser l'ensemble des files de messages, de sémaphores et de segments de mémoire partagée.

Table des IPC (fonction `ipcs`)

commande ipcs sans paramètre

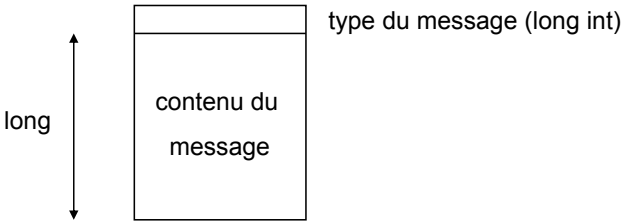
```
--> ipcs↩
IPC status from /dev/kmem as of Sun Jul  7 08:41:31 2002
T      ID     KEY        MODE        OWNER     GROUP
Message Queues:
q       0 0x3c200834 -Rrw--w--w-      root      root
q       1 0x3e200834 --rw-r--r--      root      root
Shared Memory:
m       0 0x2f1c0002 --rw-------      root       sys
m       1 0x41200579 --rw-rw-rw-      root      root
m   48005 0x00000000 --rw-rw-rw-   rifflet     users
m   43409 0x41440014 --rw-rw-rw-   rifflet     users
Semaphores:
s       0 0x2f1c0002 --ra-ra-ra-      root       sys
s       1 0x41200579 --ra-ra-ra-      root      root
s   18192 0x41440014 --ra-ra-ra-   rifflet     users
-->
```

IPC anonyme créé avec la clé IPC_PRIVATE

Envoi d'un message

```
int msgsnd (int id, void *pt_msg,
            int long, int option)
```

- `id` : identificateur de la file de message
- `pt_msg`: pointeur sur le message à envoyer
- `long`: longueur du message
- `option`: permet par exemple de ne pas bloquer l'émetteur si la file de messages est pleine
- l'appel retourne 0 si ok



Réception d'un message

```
int msgrcv (int id, void *pt_msg, int longmax,
            int option, long type)
```

- `id`: identificateur de la file de messages
- `pt_msg`: adresse pour stocker le message reçu
- `longmax`: taille max de l'emplacement de stockage du message
- `option`: permet par exemple de ne pas bloquer si la file est vide
- `type`:
  - `type` = 0: le message en tête de la file est reçu
  - `type` > 0: le premier message dans la file du type donné est reçu
  - `type` < 0: le premier message dans la file du plus petit type T t.q. T ≤ |`type`| est reçu
- l'appel retourne la longueur du message reçu

Exemple d'utilisation du `type` d'un message

On considère 3 types:

- `type` = 1: messages les plus prioritaires
- `type` = 2: messages de priorité moyenne
- `type` = 3: messages les moins prioritaires

- `msgrcv( , , , , 1)` permet de recevoir un message de la priorité max
- `msgrcv( , , , , -2)` permet de recevoir un message de priorité 1 (si disponible), sinon un message de priorité 2
- `msgrcv( , , , , -3)` permet de recevoir un message de priorité 1 (si disponible), sinon un message de priorité 2 (si disponible), sinon un message de priorité 3

Destruction d'une file de messages

```
int msgctl (int id, int cmd, … arg)
```

- `msgctl`: fournit plusieurs commandes de contrôle, dont une commande permettant de détruire une file de messages
- `id`: identificateur de la file de messages;
- `cmd`: commande; pour détruire, la commande est `IPC_RMID`
- `arg`: arguments pour certaines commandes (pour détruire: `NULL`)

Files de messages Posix.4 (message queues)

Une file de messages Posix.4 est désignée comme un fichier (même si une file de messages ne se trouve pas dans l'espace des noms des fichiers, c-à-d n'est pas forcément visible par la commande `ls`);

- le nom doit commencer par `/`
- création (si n'existe pas) et ouverture:

```
mqd_t mq_open(const char *nom,
              int option, ... );
```

- envoi d'un message: `mq_send`
- réception d'un message: `mq_receive`
- fermeture: `mq_close`
- destruction: `mq_unlink`
- réception asynchrone: `mq_notify`

  Si la file est vide, permet à un processus d'être notifié de la réception d'un message. Utilise le mécanisme des signaux.