SY40 TP6 /yann derré

a) Compilez P1.c, P2.c, P3.c pour obtenir les exécutables P1, P2 et P3.

```
(base) yannderre@MacBook-Pro-2 [10:30:53]
-> % gcc P1.c -o P1
```

Utilisation standard de GCC

b) Exécutez P1, puis réessayez une nouvelle fois. Comment expliquez-vous la réponse ?

```
(base) yannderre@MacBook-Pro-2 [10:30:42] [~/Downloads/TP6-Les files de messages/sources/output]
-> % ./"P1"
Pb msgget 1: File exists
(base) yannderre@MacBook-Pro-2 [10:30:43] [~/Downloads/TP6-Les files de messages/sources/output]
```

Le programme ne supprime jamais la file de message créée : le programme ne s'exécutera qu'une seule fois sans la supprimer.

c) Visualisez les tables du système et détruisez l'entrée adéquate (ipcs, ipcrm), puis modifiez P1.c pour que la destruction de la file de messages soit automatique.

On trouve l'ID de la message queue via la commande ipcs, et on rajoute un msgctl pour supprimer l'ancienne fille. Le P1 s'exécute maintenant correctement

```
-> % ipcs
IPC status from <running system> as of Mon May 15 10:30:45 CEST 2023
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
q 131072 0x0000013a --rw------- yannderre    staff

T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:

T     ID     KEY        MODE       OWNER    GROUP
```

```
(base) yannderre@MacBook-Pro-2 [10:37:06] [~/Downloads/TP6-Les files de messages/sources/output]
-> % ipcs -m 196608
IPC status from <running system> as of Mon May 15 10:37:15 CEST 2023
T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
```

On a supprimé les files de messages pré-existantes. On peut maintenant rajouter notre condition en fin de P1 pour supprimer sa file de message pour qu'il puisse automatiquement se réexecuter.

```
83
84      msgctl(msgid, IPC_RMID, NULL);
85
86      printf("P1 termine\n");
87      exit(0);
88  }
89
```

```
(base) yannderre@MacBook-Pro-2 [10:30:53] [~/Downloads/TP6-Les files de messages/sources/output]
-> % ./"P1"
Lancement de P1
Creation de la file de message 196608
Creation de P2 (pid = 20531)
Creation de P3 (pid = 20532)
P2 : mon pid 20531, son pid 1842491184
P3 : mon pid 20532, son pid 1829924672
P2 termine
P3 termine
P1 termine
(base) yannderre@MacBook-Pro-2 [10:30:53] [~/Downloads/TP6-Les files de messages/sources/output]
```

d) Complétez P2.c et P3.c pour que les deux processus P2 et P3 s'écrivent et lisent mutuellement leurs numéros de PID en utilisant la file de message créée par P1.

Dans P2, on envoie le type 2 à P3 et reçoit le type 3. Vice-versa dans P3.

```
(base) yannderre@MacBook-Pro-2 [10:54:02] [~/Downloads/TP6-Les files de messages/sources/output]
-> % ./"P1"
Lancement de P1
Creation de la file de message 1114112
Creation de P2 (pid = 22139)
Creation de P3 (pid = 22140)
P3 : mon pid 22140, son pid 22139
P2 : mon pid 22139, son pid 22140
P3 termine
P2 termine
P1 termine
(base) yannderre@MacBook-Pro-2 [10:54:02] [~/Downloads/TP6-Les files de messages/sources/output]
-> %
```

```
if ((msgid = msgget(cle, 0)) == -1)
    erreur("Pb msgget dans P2");

req.type = 3;
req.numPID = getpid();

if (msgsnd(msgid, &req, tailleMsg, 0) == -1){
    erreur("Pb msgsnd dans P2");
}

if (msgrcv(msgid, &rep, tailleMsg, 2, 0) == -1){
    erreur("Pb msgrcv dans P2");
}


printf("P2 : mon pid %d, son pid %d \n", getpid(), rep.numPID);
exit(0);
}
```

Ecrivez à partir de prg2.c un programme prg3.c comprenant un autre processus serveur (TRIEUR) qui trie les nombres aléatoires générés par le SERVEUR avant de les fournir au CLIENT.

```
typedef struct
{
  long type;
  int tab[NBMAXNB];
} tri_message;
```

```
#define TRI_MSG_TYPE 2
```

```
key_t cle_tri = 1234;
```

J’ai déclaré une structure additionnelle pour envoyer mon message du serveur à la fonction de tri

```
void traitantSIGINT(int s)
{
  msgctl(msgid, IPC_RMID, NULL);
  msgctl(tri_msgid, IPC_RMID, NULL);
  exit(0);
}
```

```
  erreurFin("Pb msgget 1");
if ((tri_msgid = msgget(cle_tri, IPC_CREAT | IPC_EXCL | 0600)) == -1)
  erreurFin("Pb msgget tri");
forkn(nbClients, client);
```

On se charge également de créer nos message queues, et de gérer leur suppression lors de l’arrêt via Ctrl+C (SIGINT signal)

Côté serveur:

```
// Sending the numbers to "tri.c" through the message queue
tri_message tri_msg;
tri_msg.type = TRI_MSG_TYPE;
tailleRep = req.nbNombreDemandes * sizeof(int);

memcpy(tri_msg.tab, rep.tab, req.nbNombreDemandes);

int taillemess = sizeof(tri_msg) - sizeof(long);

// Sending the message to "tri.c"
msgsnd(tri_msgid, &tri_msg, taillemess, 0);

if (fork() == 0)
{
  // Child process - execute "tri" process
  tri(taillemess);
  exit(0);
}
else
{
  wait(NULL);
}

// Receiving the sorted numbers from "tri.c"
msgrcv(tri_msgid, &tri_msg, taillemess, TRI_MSG_TYPE, 0);
//qsort(tri_msg.tab, NBMAXNB, sizeof(int), ordreAscendantInt);

memcpy(rep.tab, tri_msg.tab, req.nbNombreDemandes * sizeof(int));
```

Le serveur va copier la réponse des nombres générer du client dans une structure tri, créer un processus fils qui le triera dans la fonction tri(). Le serveur attend la réponse du fils.

```c
void tri(int tailleRep)
{
  int tri_msgid; // Message queue ID for communication with the server
  int cle = 1234;
  tri_message tri_msg;

  // Connect to the message queue created by the server
  if ((tri_msgid = msgget(cle, 0)) == -1)
  {
    perror("msgget");
    exit(1);
  }

  // Receive the numbers from the server through the message queue
  if (msgrcv(tri_msgid, &tri_msg, tailleRep, TRI_MSG_TYPE, 0) == -1)
  {
    perror("msgrcv");
    exit(1);
  }

  printf("Numbers received from the server\n");

  // Sort the numbers
  qsort(tri_msg.tab, NBMAXNB, sizeof(int), ordreAscendantInt);

  // Send the sorted numbers back to the server through the message queue
  if (msgsnd(tri_msgid, &tri_msg, tailleRep, 0) == -1)
  {
    perror("msgsnd");
    exit(1);
  }
}
```

Le fils appelant la fonction tri va se connecter à la message queue, via la clé prédéfinie. On peut maintenant appliquer notre tri croissant pour nos chiffres entiers générés via qsort, et la fonction ordreAscendantInt, fournie dans tri.c

Tri() renvoie ensuite dans la message queue pour réception par son père.

```c
      memcpy(rep.tab, tri_msg.tab, req.nbNombreDemandes * sizeof(int));

      rep.type = req.pidEmetteur;
      msgsnd(msgid, &rep, tailleRep, 0);
    }
  }
```

Ce dernier va ensuite le replacer dans une structure réponse, et le renvoyer à son client respectif (via son pidEmetteur)

Bien que les structures req et tri_message soient identiques, j'ai préféré créer une autre structure. Elle nous permettrait de s'adapter plus facilement à d'autres contraintes, (e.g. conversion de types…) si nous devions les implémenter, bien que cela consomme davantage de mémoire à l'exécution. Dans notre cas d'utilisation. Nous pourrions nous contenter de réutiliser la même structure dans la message queue de tri.