
Compte-rendu TP4 OS302

Dougy Hugo & Creton Pierre

Introduction

Durant ce TP, nous allons nous intéresser aux communications par files de messages. Nous allons notamment utiliser les fonctions pour créer des files de messages, les envoyer et les recevoir.

Exercice 1

Nous avons décidé de remplacer le symbole de la multiplication * par x, car ca nous évite de devoir mettre des quotes lors de l'exécution du client

Code

Structure calcul.h

```
1  #ifndef __CALCUL_H__
2  #define __CALCUL_H__
3
4  #include <unistd.h>
5  #include <sys/types.h>
6
7
8  struct data {
9      int op1;
10     int op2;
11     char operateur;
12     pid_t pid;
13 };
14
15 struct msg_struct {
16     long type;
17     struct data donnee;
18 };
19
20
21
22 #endif /* __CALCUL_H__ */
```

Serveur

```
1  #include "calcul.h"
2
```

```
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <signal.h>
6 #include <unistd.h>
7 #include <sys/ipc.h>
8 #include <sys/msg.h>
9
10
11 int msg_id;
12
13 void raz_msg(int signal) { //suppression de la file de messages des la
    reception d'un signal
14     printf("Suppression de la file de message!\n");
15     msgctl(msg_id,IPC_RMID,NULL);
16 }
17
18
19 int main(int argc, char const *argv[])
20 {
21     struct msg_struct msg;
22     int i_sig;
23     int result;
24
25     /* liberer la zone de messages sur reception de n'importe quel
        signal */
26
27     for (i_sig = 0 ; i_sig < NSIG ; i_sig++) { //pour supprimer la
        file de messages des la reception d'un signal
28         signal(i_sig,raz_msg);
29     }
30     key_t cle = ftok("fich",0); //creation de la cle pour notre file
        de message
31
32     if((msg_id = msgget(cle, IPC_CREAT | 0666 ) ) == -1){ //creation
        de la file de messages grace a la cle
33         perror("ERROR msgget");
34         return EXIT_FAILURE;
35     }
36
37     printf("SERVEUR: pret!\n");
38
39     while (1 == 1) {
40         /* reception */
41         if( msgrcv(msg_id, &msg, sizeof(struct data), 1, 0) == -1){ //
            on attend que le client nous envoie un message de type 1 par
            la file de messages
42             perror("ERROR msgrcv");
43             return EXIT_FAILURE;
44         }
45         printf("PID client: %d\nPID serveur: %d\n",msg.donnee.pid,
            getpid());
```

```
46     printf("SERVEUR: reception d'une requete de la part de: %d\n",
47           msg.donnee.pid);
48     /* preparation de la reponse */
49     printf("Calcul...\n");
50     printf("%d %c %d \n", (msg.donnee).op1, (msg.donnee).operateur,
51           (msg.donnee).op2);
52     msg.type = (msg.donnee).pid; //type du message de reponse =
53     pid du client
54     (msg.donnee).pid = getpid();
55     /*on effectue l'operation adequate*/
56     switch ((msg.donnee).operateur)
57     {
58     case '+':
59         result = (msg.donnee).op1 + (msg.donnee).op2;
60         break;
61
62     case '-':
63         result = (msg.donnee).op1 - (msg.donnee).op2;
64         break;
65     case 'x':
66         result = (msg.donnee).op1 * (msg.donnee).op2;
67         break;
68     case '/':
69         result = (msg.donnee).op1 / (msg.donnee).op2;
70         break;
71     }
72     msg.donnee.op1 = result;
73     printf("result: %d\n", result);
74     /* envoi de la reponse */
75
76     if(msgsnd(msg_id, &msg, sizeof(struct data), 0) == -1){ //on
77         envoie la reponse au client par la file de messages
78         perror("ERROR msgsnd");
79         return EXIT_FAILURE;
80     }
81     printf("Réponse bien envoyé\n");
82     return EXIT_SUCCESS;
83 }
```

Client

```
1  #include "calcul.h"
2
3  #include <stdlib.h>
4  #include <stdio.h>
```

```
5 #include <unistd.h>
6 #include <sys/ipc.h>
7 #include <sys/msg.h>
8
9
10 int main(int argc, char const *argv[])
11 {
12     int msg_id;
13     struct msg_struct msg;
14
15     if (argc != 4) {
16         printf("Usage: %s operande1 {+|-|x|/} operande2\n", argv[0]);
17         //on remplace * par x car ca nous evite de devoir mettre de
18         // ' ' quand on lance le client
19         return EXIT_FAILURE;
20     }
21
22     /* il faut eviter la division par zero */
23     if(argv[2][0] == '/' && atoi(argv[3]) == 0){
24         perror("Division par 0");
25         return EXIT_FAILURE;
26     }
27
28     /*on vient mettre toutes les donnees dans le message a envoyer*/
29     (msg.donnee).op1 = atoi(argv[1]);
30     (msg.donnee).op2 = atoi(argv[3]);
31     (msg.donnee).operateur = argv[2][0];
32     (msg.donnee).pid = getpid();
33     msg.type = 1;
34
35     /* ATTENTION : la file de messages doit avoir ete creee par le
36     serveur. Il
37     * faudrait tester la valeur de retour (msg_id) pour verifier que
38     cette
39     * creation a bien eu lieu. */
40
41     /* On prepare un message de type 1 à envoyer au serveur avec les
42     * informations necessaires */
43
44     key_t cle = ftok("fich",0); //on cree la cle pour notre file de
45     message
46
47     if((msg_id = msgget(cle, 0) ) == -1){ //on vient utiliser la file
48     de message cree par le serveur grace a la cle
49         perror("ERROR msgget");
50         return EXIT_FAILURE;
51     }
52     printf("CLIENT %d: preparation du message contenant l'operation
53     suivante:\n
54     %d %c %d\n", getpid(), atoi(argv[1]), argv[2][0], atoi(argv
```

```

[3])));
49
50
51     /* envoi du message */
52
53     if(msgsnd(msg_id, &msg, sizeof(struct data), 0) == -1){ //on envoie
        notre message au serveur
54         perror("ERROR msgsnd");
55         return EXIT_FAILURE;
56     };
57
58     /* reception de la reponse */
59     printf("J'attend...\n");
60
61
62     if(msgrcv(msg_id, &msg, sizeof(struct data), getpid(), 0) == -1){ //
        on attend la reponse du serveur de type egal au pid du client
63         perror("ERROR msgrcv");
64         return EXIT_FAILURE;
65     }
66     printf("Bien reçu !\n");
67     printf("CLIENT: resultat reçu depuis le serveur %d : %d\n", (msg.
        donnee).pid, (msg.donnee).op1);
68     return EXIT_SUCCESS;
69 }

```

Résultat

On test toutes les opérations sur les clients et on regarde le résultat sur le serveur.

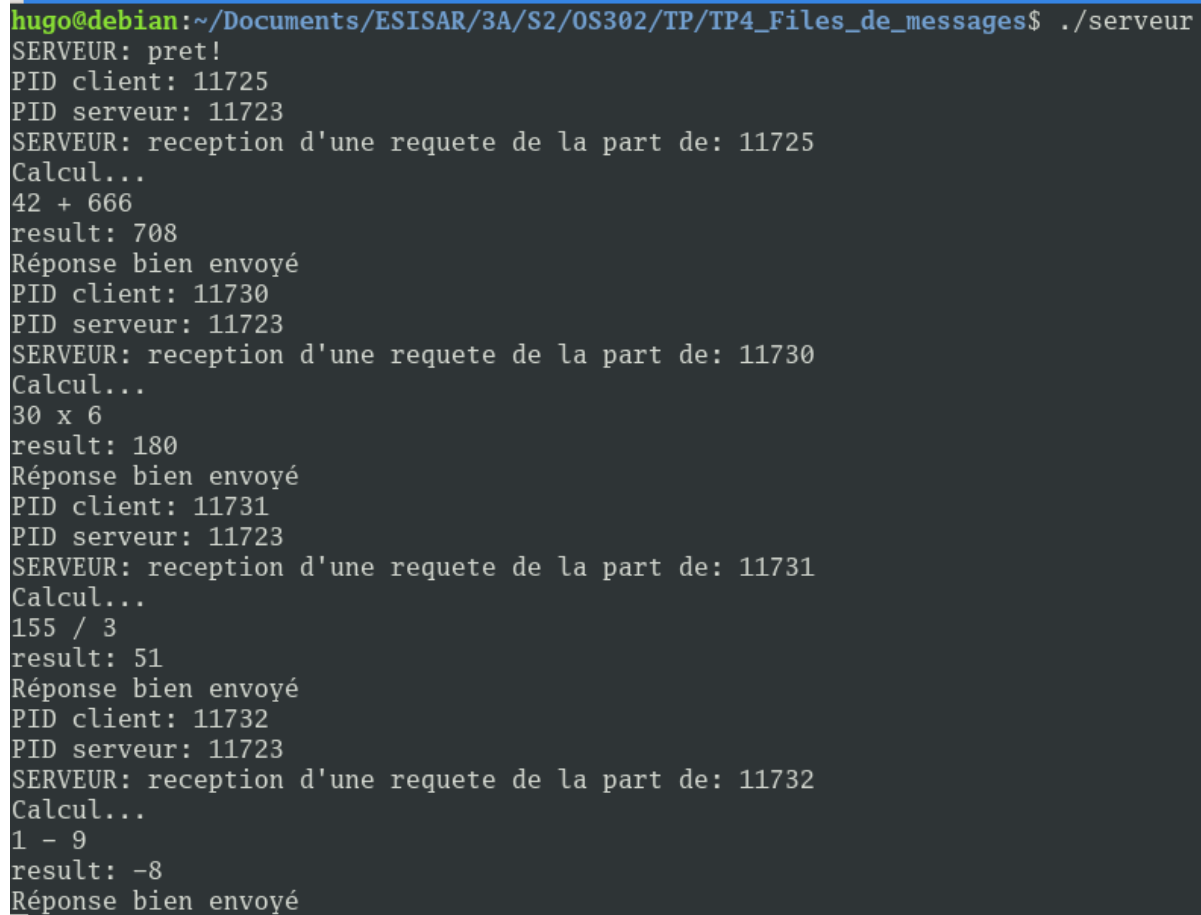
```

hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./client 42 + 666
CLIENT 11725: preparation du message contenant l'operation suivante:          42 + 666
J'attend...
Bien reçu !
CLIENT: resultat reçu depuis le serveur 11723 : 708
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./client 30 x 6
CLIENT 11730: preparation du message contenant l'operation suivante:          30 x 6
J'attend...
Bien reçu !
CLIENT: resultat reçu depuis le serveur 11723 : 180
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./client 155 / 3
CLIENT 11731: preparation du message contenant l'operation suivante:          155 / 3
J'attend...
Bien reçu !
CLIENT: resultat reçu depuis le serveur 11723 : 51
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./client 1 - 9
CLIENT 11732: preparation du message contenant l'operation suivante:          1 - 9
J'attend...
Bien reçu !
CLIENT: resultat reçu depuis le serveur 11723 : -8
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ 

```

Figure 1: Opérations sur le client

On remarque bien que le client reçoit le résultat de l'opération donné par le serveur par la file de message. Sur le serveur, on a le résultat suivant:

A terminal window with a dark background and light green text. The prompt is 'hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages\$'. The user has entered './serveur'. The output shows three sequential requests from clients with PIDs 11725, 11730, and 11731. Each request is received, a calculation is performed, and the result is sent back to the client.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./serveur
SERVEUR: pret!
PID client: 11725
PID serveur: 11723
SERVEUR: reception d'une requete de la part de: 11725
Calcul...
42 + 666
result: 708
Réponse bien envoyé
PID client: 11730
PID serveur: 11723
SERVEUR: reception d'une requete de la part de: 11730
Calcul...
30 x 6
result: 180
Réponse bien envoyé
PID client: 11731
PID serveur: 11723
SERVEUR: reception d'une requete de la part de: 11731
Calcul...
155 / 3
result: 51
Réponse bien envoyé
PID client: 11732
PID serveur: 11723
SERVEUR: reception d'une requete de la part de: 11732
Calcul...
1 - 9
result: -8
Réponse bien envoyé
```

Figure 2: Serveur

Le serveur reçoit bien la demande du client car on affiche le pid de ce dernier. On a fait un printf pour savoir si le serveur répond bien au client.

L'envoi et la réception sont bloquant par défaut donc le serveur attend d'avoir une nouvelle opération, donc de recevoir quelque chose par la file.

Avec la commande `ipcs`, on remarque qu'il y a une file de message créé qui n'a pas encore été supprimée par le serveur, car nous n'avons pas encore arrêté ce dernier.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ipcs
----- Files de messages -----
clef      msqid      propriétaire perms      octets utilisés messages
0x00067f70 32768      hugo      666      0      0
```

Figure 3: résultat de la commande: ipcs

Pour quitter, on fait un CTRL+C sur le serveur. À la fermeture, on remarque bien que le serveur supprime la file de message.

```
Réponse bien envoyé
^C suppression de la file de message!
ERROR msgrcv: Interrupted system call
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$
```

Figure 4: Serveur à la fermeture

Si on exécute la commande `ipcs` sur le shell, on a bien la file de message supprimée.

Exercice 2

Implémentation

Pour implémenter le code, nous avons repris la forme de l'exercice 1 avec un code client et un code serveur ainsi qu'une structure pour le message contenant : le type (mis à 1 par défaut par le client, puis il viendra attendre un message d'un type égal à son pid), le pid (pour différencier les clients) et une chaîne de caractère (pour le message à traduire envoyé par le client, et la message traduit répondu par le serveur).

Comme l'exercice nous impose 2 files de messages, nous en avons créé deux dans le code du serveur. Une pour l'envoi des données par le client et l'autre pour la réponse du serveur. Nous n'avons pas eu de difficultés particulières. Nous avons fait attention à gérer les types des messages pour bien les réceptionner.

Code

Structure struct.h


```
1  #ifndef __STRUCT_H__
2  #define __STRUCT_H__
3
4  #include <unistd.h>
5  #include <sys/types.h>
6
7  #define SIZE 120
8
9  /* On crée notre suture structure msgbuf avec le type, une chaîne de caractères et le pid*/
10
11  struct msgbuf
12  {
13      long mtype;
14      pid_t pid;
15      char mtext[120];
16  };
17
18  //fonction qui traduit les minuscules en majuscules
19  void majuscule(char *chaine)
20  {
21      int i = 0;
22
23      while (chaine[i] != '\0')
24      {
25          if(chaine[i] >= 97 && chaine[i] <= 122){
26              chaine[i] = chaine[i] - 32;
27          }
28          i++;
29      }
30  }
31
32  #endif
```

Serveur

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <signal.h>
4  #include <unistd.h>
5  #include <sys/ipc.h>
6  #include <sys/msg.h>
7  #include <string.h>
8
9  #include "struct.h"
10
11  int msg_id;
12  int msg_retour_id;
```

```
13
14 //fonction pour supprimer les deux files de messages
15 void raz_msg(int signal) {
16     printf("Suppression des files de message!\n");
17     msgctl(msg_id,IPC_RMID,NULL);
18     msgctl(msg_retour_id,IPC_RMID,NULL);
19 }
20
21 int main(){
22
23     struct msgbuf msg;
24
25     // si un signal est reçu, on execute la fonction raz_msg
26     int i_sig;
27     for (i_sig = 0 ; i_sig < NSIG ; i_sig++) {
28         signal(i_sig,raz_msg);
29     }
30
31     key_t cle;
32
33     if((cle = ftok("fich",0)) == -1){
34         perror("ERROR ftok");
35         exit(EXIT_FAILURE);
36     }
37     //comme c'est le serveur qui crée les files de message, on lui passe en
    argument IPC_CREAT et les droits d'accès (en octal)
38     if((msg_id = msgget(cle,IPC_CREAT | 0666)) == -1){
39         perror("ERROR msgget");
40         exit(EXIT_FAILURE);
41     }
42
43     key_t cle_retour;
44
45     if((cle_retour = ftok("fich2",0)) == -1){
46         perror("ERROR ftok retour");
47         exit(EXIT_FAILURE);
48     }
49
50     if((msg_retour_id = msgget(cle_retour,IPC_CREAT | 0666)) == -1){
51         perror("ERROR msgget retour");
52         exit(EXIT_FAILURE);
53     }
54
55
56     printf("SERVEUR PRET: \n");
57
58
59     while (1 == 1)
60     {
61         /* Reception de la demande du client (de type 1) */
62         //on attend la demande d'un client dont le type est 1
```

```
63     if(msgrcv(msg_id,&msg,sizeof(struct msgbuf) - sizeof(long),1,0)
64         == -1){
65         perror("ERROR msgrcv");
66         exit(EXIT_FAILURE);
67     }
68     if(strcmp(msg.mtext,"@") != 0){
69
70         printf("SERVEUR %d: reception d'une requete de la part de:
71             %d\n",getpid(),msg.pid);
72         //le type du message à renvoyer est le pid du client, on affecte le
73         pid du serveur au message de réponse pour que le client puisse
74         savoir qui lui a répondu
75         msg.mtype = msg.pid;
76         msg.pid = getpid();
77
78         //on transforme la chaine en majuscule avec la fonction donnée dans
79         struct.h et on la copie dans le message de reponse
80         majuscule(msg.mtext);
81
82         printf("chaîne de retour: %s\n",msg.mtext);
83
84         /* Renvoie de la réponse */
85
86         //on envoie repond donc au client par la deuxieme file de
87         message
88         if(msgsnd(msg_retour_id, &msg,sizeof(struct msgbuf) -
89             sizeof(long),0) == -1){
90             perror("ERROR msgsnd");
91             exit(EXIT_FAILURE);
92         }
93         printf("Réponse bien envoyé\n");
94     }
95     else{ //on supprime les files de messages
96         printf("Arret avec succès\n");
97         msgctl(msg_id,IPC_RMID,NULL);
98         msgctl(msg_retour_id,IPC_RMID,NULL);
99
100         exit(EXIT_SUCCESS);
101     }
102 }
```

Client

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <signal.h>
4  #include <unistd.h>
5  #include <sys/ipc.h>
6  #include <sys/msg.h>
7  #include <string.h>
8
9  #include "struct.h"
10
11 int main(int argc, char *argv[]){
12
13     //on vérifie que le client n'entre qu'un seul argument
14     if(argc != 2){
15         printf("1 argument demandé\n");
16         exit(EXIT_FAILURE);
17     }
18
19     //on crée le message qui sera utilisé pour l'envoi et le retour
20     struct msgbuf msg;
21
22     //on affecte les valeurs au message à envoyer (on met le type à 1 de
    base)
23     strcpy(msg.mtext,argv[1]);
24     msg.mtype = 1;
25
26     //on crée les identifiants pour les deux files de message créées par le
    serveur
27     int msg_id;
28     int msg_retour_id;
29     //on donne également le pid du client pour que le serveur puisse
    traiter plusieurs demandes
30     msg.pid = getpid();
31
32     //on crée la clé de la file de message à envoyer
33     key_t cle;
34     if((cle = ftok("fich",0)) == -1){
35         perror("ERROR ftok");
36         exit(EXIT_FAILURE);
37     }
38     //on donne 0 en option de msgget car c'est le serveur qui crée les
    files de message, le client vient juste l'utiliser
39     if((msg_id = msgget(cle,0)) == -1){
40         perror("ERROR msgget");
41         exit(EXIT_FAILURE);
42     }
43     //pareil pour la file de message retour
44     key_t cle_retour;
```

```
45     if((cle_retour = ftok("fich2",0)) == -1){
46         perror("ERROR ftok retour");
47         exit(EXIT_FAILURE);
48     }
49
50     if((msg_retour_id = msgget(cle_retour,0)) == -1){
51         perror("ERROR msgget retour");
52         exit(EXIT_FAILURE);
53     }
54
55     /* Envoi du message */
56
57     //on envoie un message par la première file de message avec comme
    identifiant msg_id
58     if(msgsnd(msg_id,&msg,sizeof(struct msgbuf) - sizeof(long),0) ==
        -1){
59         perror("ERROR msgsnd");
60         exit(EXIT_FAILURE);
61     }
62
63     //on regarde si on a le caractère @. Si oui le client s'arrête et n'
    attend pas de réponse
64     if(strcmp(argv[1],"@") != 0){
65
66         printf("CLIENT %d: preparation du message",msg.pid);
67         printf("En attente d'une réponse: \n");
68
69         //on attend la réponse de l'autre file de message dont le type est
        le pid du client
70         if(msgrcv(msg_retour_id,&msg,sizeof(struct msgbuf) - sizeof(
            long),getpid(),0) == -1){
71             perror("ERROR msgrcv retour");
72             exit(EXIT_FAILURE);
73         }
74
75         printf("Bien reçu de la part de: %d\n",msg.pid);
76         printf("Voici la réponse: %s\n",msg.mtext);
77     }
78     else{
79         printf("Arret avec succès\n");
80         exit(EXIT_SUCCESS);
81     }
82     return EXIT_SUCCESS;
83 }
```

Résultats

Comme pour l'exercice précédent, nous avons tester différents cas entre le serveur et le client.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./client2 "allons enfants de la patrie"
CLIENT 5574: preparation du messageEn attente d'une réponse:
Bien reçu de la part de: 5573
Voici la réponse: ALLONS ENFANTS DE LA PATRIE
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./client2 "Le Jour De Gloire Est Arrivee"
CLIENT 5575: preparation du messageEn attente d'une réponse:
Bien reçu de la part de: 5573
Voici la réponse: LE JOUR DE GLOIRE EST ARRIVEE
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./client2 @
Arret avec succès
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ █
```

Figure 5: Demandes des clients

Sur le serveur (avant la réception de “@”) on a bien la réception du message, sa traduction en majuscule, puis son envoi au client avec l’autre file de message.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./serveur2
SERVEUR PRET:
SERVEUR 5573: reception d'une requete de la part de: 5574
chaîne de retour: ALLONS ENFANTS DE LA PATRIE
Réponse bien envoyé
SERVEUR 5573: reception d'une requete de la part de: 5575
chaîne de retour: LE JOUR DE GLOIRE EST ARRIVEE
Réponse bien envoyé
█
```

Figure 6: Serveur avant fermeture

On peut observer avec la commande `ipcs` qu’il y a bien deux files de messages créés.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ipcs

----- Files de messages -----
clef      msqid      propriétaire perms      octets utilisés messages
0x00067f70 327680    hugo      666        0          0
0x00067f71 360449    hugo      666        0          0
```

Figure 7: Résultat de la commande: `ipcs`

Une fois que le client envoie le caractère “@”, le serveur s’arrête et supprime les files de messages.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ ./serveur2
SERVEUR PRET:
SERVEUR 5573: reception d'une requete de la part de: 5574
chaîne de retour: ALLONS ENFANTS DE LA PATRIE
Réponse bien envoyé
SERVEUR 5573: reception d'une requete de la part de: 5575
chaîne de retour: LE JOUR DE GLOIRE EST ARRIVEE
Réponse bien envoyé
Arrêt avec succès
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP4_Files_de_messages$ □
```

Figure 8: Serveur à la fermeture

Travail collaboratif

Pour ce TP, nous nous sommes réparti le travail de la manière suivante:

Exercice 1:

- *Hugo & Pierre*: implémentation & tests
- *Pierre*: commentaires

Exercice 2:

- *Hugo & Pierre*: réflexion
- *Hugo & Pierre*: implémentation & tests
- *Hugo* : commentaires

Conclusion

Pour conclure sur ce TP, nous avons pu utiliser les fonctions vues en cours pour gérer les files de messages.

Nous avons du lire les pages man des fonctions pour pouvoir se les approprier correctement. Nous avons également utilisé la commande `ipcs` pour voir si les files de messages étaient présentes en mémoire.