Réseau - Rapport du TP n° 6

Foucault Antoine

18 décembre 2013

Sommaire

Introduction

L'objectif de ce TP est de s'initier à la programmation de RPC (Remote Procedure Call). Nous allons tout d'abord reproduire des exemples de cours avant de réaliser un programme de « traitement d'image ». Ces exercices nous permettront également d'analyser la manière dont sont transmises les informations entre l'application cliente et le serveur. La programmation des applications sera facilitée par l'utilisation de rpcgen, un logiciel permettant de générer des squelettes de programme en C à partir de la définition des procédures distantes à réaliser.

Chapitre 1

Reprise des exemples de cours

La première partie du TP s'est composée de la réalisation de programmes vus en cours ainsi que de leurs analyses à l'aide de la commande rpcinfo.

1.1 Application cliente/serveur

Notre application se décompose en deux parties :
— Le serveur qui contiendra le code des procédures distantes
— Le client qui se chargera d'apeller les procédures du serveur

Les deux procédures réalisées en cours sont un additionneur et un multiplicateur. Ces deux procédures prennent donc deux « int » en paramètres et retournent un « int » en résultat. La figure 1.1 présente l'interaction entre les deux applications.

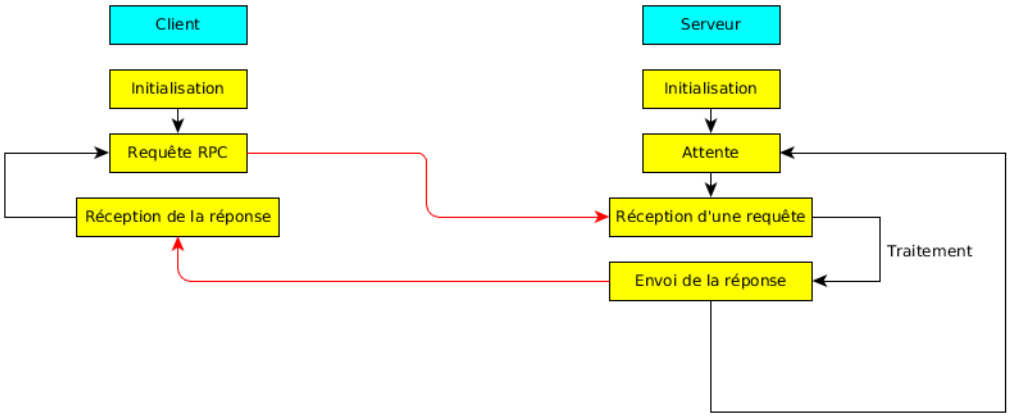

FIGURE 1.1 – Interaction entre une partie cliente et serveur lors de la mise en place d'une application RPC

Un exemple d'utilisation de l'application cliente est disponible section A.3 page 25.

1.2 La commande rpcinfo

La commande « rpcinfo » sert à récupérer les informations sur les procédures RPC du serveur local ou d'un serveur distant.

Les différents paramètres sont :

-p [**serveur** :] Retourne tous les programmes RPC enregistrés sur le serveur définit, le serveur local sinon

-u [**serveur** [programme] [version](optionnel) :] Effectue un appel à la procédure 0 du programme spécifié sur le serveur spécifié en utilisant le protocole UDP

-t [**serveur** [programme] [version](optionnel) :] Effectue un appel à la procédure 0 du programme spécifié sur le serveur spécifié en utilisant le protocole TCP

-n [**port** :] Sert à définir un port de connexion au serveur autre que celui assigné par défaut

-b [**programme** [version] :] Envoi en broadcast la réponse de la procédure 0 du programme spécifié avec sa version via le protocole UDP

-d [**programme** [version] :] Enlève le programme spécifié de version spécifiée de la table des services RPC disponibles (nécéssite les droits superutilisateur)

Nous avons testé cette commande sur nos applications avec les différents commutateurs, les résultats sont disponibles dans la figure A.2 page 15. On remarque que l'application n° 805306401 correspond au numéro hexadécimal de programme dans le fichier .x : 0x30000021.

Chapitre 2

Réalisation d'une application « Traitement d'image »

La seconde partie du TP a consisté à programmer une application client/-serveur RPC chargée de traiter des images NVG (Niveaux de gris) à l'aide de différentes procédures :

Histogramme : Calcule le nombre d'occurences de chaque niveau de gris dans l'image

Filtre moyenne : Effectue une moyenne localisée des niveaux de gris d'une image à l'aide d'une matrice 3*3 pour chaque point de l'image (sauf les bords)

Filtre médian : Affecte à chaque pixel (sauf les bords) le niveau médian localisé à l'aide d'une matrice 3*3

Le fonctionnement de l'application client/serveur sera le même que celui présenté dans la figure 1.1 page 4.

2.1 Le fichier générateur pour rpcgen

Afin de générer le squelette de nos programmes il est nécessaire de créer un fichier « générateur » pour rpcgen. Ce fichier possédant généralement l'extension .x est écrit dans le langage RPCL et définit nos procédures distantes avec leurs paramètres et valeurs de retour (comme les prototypes en C).

Le listing 2.1 présente le fichier générateur de notre application.

```
struct nvg{

        int tab[2500];
        int m;
        int n;
};

struct histogr{

        int tab[255];
};

program TP6{

        version HISTOVERS{

                histogr HISTO(nvg) = 1;
                nvg  MOY(nvg) = 2;
                nvg  MED(nvg) = 3;
        }= 1;

}=0x30000021;
```

Listing 2.1 – Fichier de génération de notre programme

Une image étant de taille variable (maximum 50*50) on a définit une structure regroupant les informations brutes (tab[2500]) de l'image ainsi que ses dimensions (m et n). L'histogramme, nécessaire seulement pour une procédure est un tableau de la taille des différentes valeurs que peut prendre un pixel (ici 255 valeurs)

2.2 Analyse de la fonction d'encodage

Afin de transmettre les paramètres/le retour des fonctions, les informations transmises ont besoin d'être encodées.

rpcgen utilise XDR (eXternal Data Representation) un standard de la couche présentation pour transférer les données. XDR définit un encodage pour les principaux types de données (int, float, string...) qu'il utilise pour encoder des plus grosses données tel que les structures. Le listing 2.2 montre bien ce découpage des structure en entitées de bases, connues et encodables.

```
1  buf = XDR_INLINE (xdrs, (2 + 2500 )*
       BYTES_PER_XDR_UNIT);
2  if (buf == NULL) {
3    if (!xdr_vector (xdrs, (char *)objp->tab, 2500,
4      sizeof (int), (xdrproc_t) xdr_int))
5      return FALSE;
6    if (!xdr_int (xdrs, &objp->m))
7      return FALSE;
8    if (!xdr_int (xdrs, &objp->n))
9      return FALSE;
10   } else {
11
12     register int *genp;
13
14     for (i = 0, genp = objp->tab;
15     i < 2500; ++i) {
16     IXDR_PUT_LONG(buf, *genp++);
17   }
18
19   IXDR_PUT_LONG(buf, objp->m);
20   IXDR_PUT_LONG(buf, objp->n);
21 }
22 return TRUE;
```

Listing 2.2 – Fonction d'encodage de notre structure nvg

2.3 Choix du protocole et test de l'application

Nous avons choisi d'utiliser le protocole TCP dans notre application afin d'assurer l'intégrité des données et de permettre la transmission d'images de taille importante en toute sécurité.
Un exemple d'utilisation de l'application est disponible en figure A.3 page 25.

Conclusion

Ce TP nous aura permis de nous initier à la programmation d'applications client/serveur RPC. Nous avons également pu voir quelques intéractions du système avec ces programmes. L'utilisation de rpcgen présente un gain considérable de temps pour la programmation d'une application distribuée, il est cependant préférable de bien connaître le mécanisme interne des squelettes générés afin de maîtriser l'ensemble du flot d'exécution de notre programme.

Annexe A

Codes Sources et utilisation des applications

A.1 Application de cours

A.1.1 Code source

```
1  /*
2   * This is sample code generated by rpcgen.
3   * These are only templates and you can use them
4   * as a guideline for developing your own functions.
5   */
6
7  #include "exo1.h"
8
9  void vider_stdin(){
10
11     int c;
12
13     do {
14         c = getchar();
15     } while (c != '\n' && c != EOF);
16 }
17
18 void
19 exo1_1(char *host)
20 {
21   CLIENT *clnt;
```

```
22    int  *resultat;
23    int argument1, argument2;
24
25    char choix;
26
27  #ifndef DEBUG
28    clnt = clnt_create (host, EXO1, EXO1VERS, "udp");
29    if (clnt == NULL) {
30      clnt_pcreateerror (host);
31      exit (1);
32    }
33  #endif  /* DEBUG */
34
35    while(1){
36
37      printf("Options :\n\n\t1. Additionner 2 nombres\n\t2
          . Multiplier deux nombres\n\nVotre choix (q pour
          quitter) : ");
38
39      scanf("%c", &choix);
40      vider_stdin();
41
42      if(choix != 'q'){
43
44        printf("\nEntrez l'argument 1 : ");
45        scanf("%i", &argument1);
46
47        printf("Entrez l'argument 2 : ");
48        scanf("%i", &argument2);
49      }
50
51      vider_stdin();
52
53      switch(choix){
54
55        case '1': resultat = add_1(argument1, argument2,
            clnt);
56              if (resultat == (int *) NULL) {
57                clnt_perror (clnt, "call failed");
58              }
59              else{
```

```
60
61                printf("\nR sultat : %i\n", *resultat);
62              }
63              break;
64
65        case '2': resultat = mult_1(argument1, argument2,
              clnt);
66              if (resultat == (int *) NULL) {
67                clnt_perror (clnt, "call failed");
68              }
69              else{
70
71                printf("\nR sultat : %i\n", *resultat);
72              }
73              break;
74
75        case 'q': printf("Fin du programme\n");
76              exit(0);
77      }
78    }
79 #ifndef DEBUG
80    clnt_destroy (clnt);
81 #endif    /* DEBUG */
82 }
83
84
85 int
86 main (int argc, char *argv[])
87 {
88    char *host;
89
90    if (argc < 2) {
91      printf ("usage: %s server_host\n", argv[0]);
92      exit (1);
93    }
94    host = argv[1];
95    exo1_1 (host);
96 exit (0);
97 }
```

Listing A.1 – Sources du client

```c
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "exo1.h"

int *
add_1_svc(int arg1, int arg2,  struct svc_req *rqstp)
{
  static int  result;

  result = arg1 + arg2;

  return &result;
}

int *
mult_1_svc(int arg1, int arg2,  struct svc_req *rqstp)
{
  static int  result;

  result = arg1 * arg2;

  return &result;
}
```

Listing A.2 – Sources du serveur

A.1.2 Utilisation

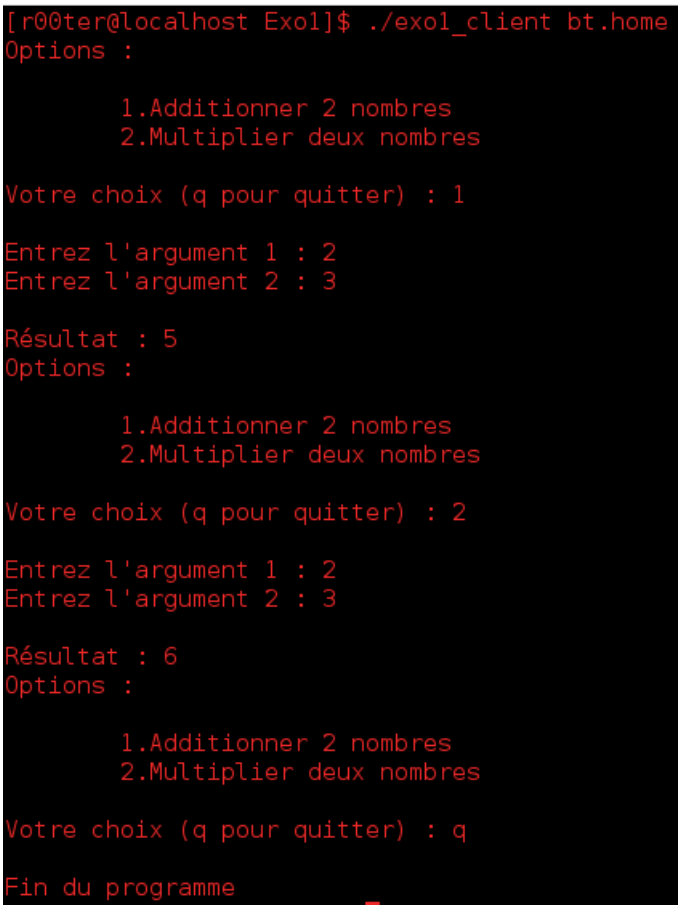

FIGURE A.1 – Exemple d'usage de l'application réalisée en cours

A.1.3 Utilisation de la commande rpcinfo

```
[r00ter@localhost Exo1]$ rpcinfo -p bt.home
   program vers proto   port  service
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  41473  status
    100024    1   tcp  50188  status
 805306400    1   udp    727
 805306400    1   tcp    728
 805306401    1   udp    762
 805306401    1   tcp    763
[r00ter@localhost Exo1]$ rpcinfo -t bt.home 805306401
program 805306401 version 1 ready and waiting
```

FIGURE A.2 – Exemple d'usage de l'application rpcinfo

A.2 Application « Traitement d'image »

A.2.1 Code source

```
1  /*
2   * This is sample code generated by rpcgen.
3   * These are only templates and you can use them
4   * as a guideline for developing your own functions.
5   */
6
7  #include "generateur.h"
8  #include <time.h>
9  #include <unistd.h>
10
11 void vider_stdin(){
12
13     int c;
14
15     do {
16         c = getchar();
17     } while (c != '\n' && c != EOF);
18 }
19
20 void remplir_image(struct nvg *i){
21
```

```
22    int j;
23    int l, h;
24
25    do{
26      printf("\nVeuillez␣entrer␣la␣hauteur␣de␣l'image␣(
          max.␣50)␣:␣");
27      scanf("%i", &h);
28    }while(h>50 || h<0);
29
30    do{
31      printf("Veuillez␣entrer␣la␣largeur␣de␣l'image␣(max.
          ␣50)␣:␣");
32      scanf("%i", &l);
33    }while(l>50 || l<0);
34
35    for(j=0; j<(l*h); j++){
36      i->tab[j] = rand()%256;
37    }
38
39    i->m = h;
40    i->n = l;
41  }
42
43  void afficher_image(nvg *image){
44
45    int i,j;
46
47    printf("\n");
48
49    for(i=0; i < (image->m); i++){
50
51      for(j=0; j < (image->n); j++){
52
53        printf("%i\t", image->tab[(i*(image->n)) +j]);
54      }
55
56      printf("\n");
57    }
58
59    printf("\n\nLongueur␣:␣%i\nHauteur␣:␣%i\n\n", image->
        n, image->m);
```

```
60 }
61
62 //Affiche un histogramme
63 void afficher_histo(histogr *h){
64
65   int i;
66
67   printf("Affichage de l'histogramme : \n\n");
68
69   for(i=0; i<255; i++){
70
71     if(!(i%10)){
72
73       printf("\n");
74     }
75
76     printf("%i ", h->tab[i]);
77   }
78
79   printf("\n");
80 }
81
82 void
83 tp6_1(char *host)
84 {
85   CLIENT *clnt;
86
87   histogr  *histogramme;
88   nvg  *image_retour;
89   nvg image;
90
91   char choix;
92
93 #ifndef DEBUG
94   clnt = clnt_create (host, TP6, HISTOVERS, "tcp");
95   if (clnt == NULL) {
96     clnt_pcreateerror (host);
97     exit (1);
98   }
99 #endif  /* DEBUG */
100
```

```
101   srand(getpid());
102
103   while(1){
104
105     printf("Options␣:\n\n\t1.D terminer␣l'histogramme␣
          de␣votre␣image\n\t2.Appliquer␣un␣filtre␣moyenne␣
          sur␣votre␣image\n\t3.Appliquer␣un␣filtre␣m dian
          ␣sur␣votre␣image\n\nVotre␣choix␣(q␣pour␣quitter)
          ␣:␣");
106
107     scanf("%c", &choix);
108     vider_stdin();
109
110     if(choix != 'q'){
111       remplir_image(&image);
112       afficher_image(&image);
113       vider_stdin();
114     }
115
116     switch(choix){
117
118       case '1': histogramme = histo_1(&image, clnt);
119             if (histogramme == (histogr *) NULL) {
120               clnt_perror (clnt, "call␣failed");
121             }
122             else{
123
124               afficher_histo(histogramme);
125             }
126             break;
127
128       case '2': image_retour = moy_1(&image, clnt);
129             if (image_retour == (nvg *) NULL) {
130               clnt_perror (clnt, "call␣failed");
131             }
132             else{
133
134               printf("\t\t****************Image␣de␣
                    r sultat*****************");
135               afficher_image(image_retour);
136             }
```

```
137             break;
138
139         case '3': image_retour = med_1(&image, clnt);
140             if (image_retour == (nvg *) NULL) {
141               clnt_perror (clnt, "call failed");
142             }
143             else{
144               afficher_image(image_retour);
145             }
146             break;
147
148         case 'q': printf("\n\nFin du programme\n");
149             exit (0);
150     }
151   }
152
153 #ifndef DEBUG
154   clnt_destroy (clnt);
155 #endif    /* DEBUG */
156 }
157
158
159 int
160 main (int argc, char *argv[])
161 {
162   char *host;
163
164   if (argc < 2) {
165     printf ("usage: %s server_host\n", argv[0]);
166     exit (1);
167   }
168   host = argv[1];
169   tp6_1 (host);
170   exit (0);
171 }
```

Listing A.3 – Sources du client

```
1 /*
2  * This is sample code generated by rpcgen.
3  * These are only templates and you can use them
```

```
4   * as a guideline for developing your own functions.
5   */
6
7  #include "generateur.h"
8
9  //Fonction utilis e dans qsort pour trier un tableau
10 int cmp_int(const void *a, const void *b){
11
12   int *c = (int *)a;
13   int *d = (int *)b;
14
15   return *c-*d;
16 }
17
18 //Recopie les bords d'une matrice sur une autre
19 void recopier_bords(nvg *image, nvg *image_res){
20
21   int i, j;
22
23   for(i=0; i < (image->m); i++){
24     image_res->tab[i*(image->n)] = image->tab[i*(image
         ->n)];
25     image_res->tab[(i+1)*(image->n) - 1] = image->tab[(
         i+1)*(image->n) - 1];
26   }
27
28   for(j=0; j < (image->n); j++){
29
30     image_res->tab[j] = image->tab[j];
31     image_res->tab[(image->n)*(image->m - 1)+j] = image
         ->tab[(image->n)*(image->m - 1)+j];
32   }
33 }
34
35 //Affiche une image (pour DEBUG)
36 void afficher_image(nvg *image){
37
38   int i,j;
39
40   printf("\n");
41
```

```
42    for(i=0; i< (image->m); i++){
43
44      for(j=0; j< (image->n); j++){
45
46        printf("%i\t", image->tab[(i*(image->n)) +j]);
47      }
48
49      printf("\n");
50    }
51
52    printf("\n\nLongueur : %i\nHauteur : %i\n\n", image->
        m, image->n);
53  }
54
55  //Calcule la moyenne d'une matrice 3*3
56  int moyenne_matrice_33(int *mat){
57
58    int moyenne = 0, i;
59
60    for(i=0; i<10; i++){
61
62      moyenne += mat[i];
63    }
64
65    return (moyenne/9);
66  }
67
68  //Retourne le m dian d'une matrice 3*3
69  int median_matrice_33(int *mat){
70
71    //Triage du tableau
72    qsort(mat, 9, sizeof(int), cmp_int);
73
74    //Retour de la valeur m diane
75    return(mat[4]);
76  }
77
78  //Rempli une matrice 3*3    l'aide d'une image nvg et d
      'un indice de cette image
79  void remplir_matrice_33(nvg *image, int i, int *matrice
      ){
```

```
80
81    int x,y;
82
83    for(x = (i-1), y = 0; x <= (i+1); x++, y++){
84
85      matrice[y] = image->tab[x];
86    }
87
88    for(x = ((i-1) - (image->n)); x <= ((i+1) - (image->n
          )); x++, y++){
89
90      matrice[y] = image->tab[x];
91    }
92
93    for(x = ((i-1) + (image->n)); x <= ((i+1) + (image->n
          )); x++, y++){
94
95      matrice[y] = image->tab[x];
96    }
97  }
98
99  //Renvoi l'histogramme d'une image
100 histogr *
101 histo_1_svc(nvg *image,  struct svc_req *rqstp)
102 {
103   static histogr  histo;
104   int i, j;
105
106   printf("\n\nAppel de la fonction histogramme en RPC\n
          ");
107
108   for(i=0; i<255; i++){
109
110     histo.tab[i] = 0;
111   }
112
113   for(i=0; i<(image->n); i++){
114
115     for(j=0; j<(image->m); j++){
116
117       histo.tab[image->tab[(i*(image->n)+j)]] += 1;
```

```
118       }
119     }
120
121     printf("Retour des r sultats\n");
122
123     return &histo;
124 }
125
126 //Renvoi une image    laquelle on a appliqu  un filtre
      moyenne
127 nvg *
128 moy_1_svc(nvg *image,  struct svc_req *rqstp)
129 {
130     static nvg image_res;
131
132     printf("\n\nAppel de la fonction moyenne en RPC\n");
133
134     int matrice33[9], i, j;
135
136     image_res.m = image->m;
137     image_res.n = image->n;
138
139     //On recopie les bords de l'image
140     recopier_bords(image, &image_res);
141
142     //On applique le filtre
143     for(i=1; i < ((image->m) - 1); i++){
144
145       for(j=1; j < ((image->n) - 1); j++){
146
147         remplir_matrice_33(image, i*(image->n) + j,
            matrice33);
148         image_res.tab[i*(image->n) + j] =
            moyenne_matrice_33(matrice33);
149       }
150     }
151
152     printf("Retour des r sultats");
153
154     return &image_res;
155 }
```

```
156
157 //Renvoi une image    laquelle on a appliqu  un filtre
        m dian
158 nvg *
159 med_1_svc(nvg *image,  struct svc_req *rqstp)
160 {
161   static nvg image_res;
162
163   printf("\n\nAppel de la fonction m dian en RPC\n");
164
165   int matrice33[9], i, j;
166
167   image_res.m = image->m;
168   image_res.n = image->n;
169
170   //On recopie les bords de l'image
171   recopier_bords(image, &image_res);
172
173   //On applique le filtre
174   for(i=1; i < ((image->m) - 1); i++){
175
176     for(j=1; j < ((image->n) - 1); j++){
177
178       remplir_matrice_33(image, i*(image->n) + j,
            matrice33);
179       image_res.tab[i*(image->n) + j] =
            median_matrice_33(matrice33);
180     }
181   }
182
183   printf("Retour des r sultats");
184
185   return &image_res;
186 }
```

Listing A.4 – Sources du serveur

A.2.2 Utilisation

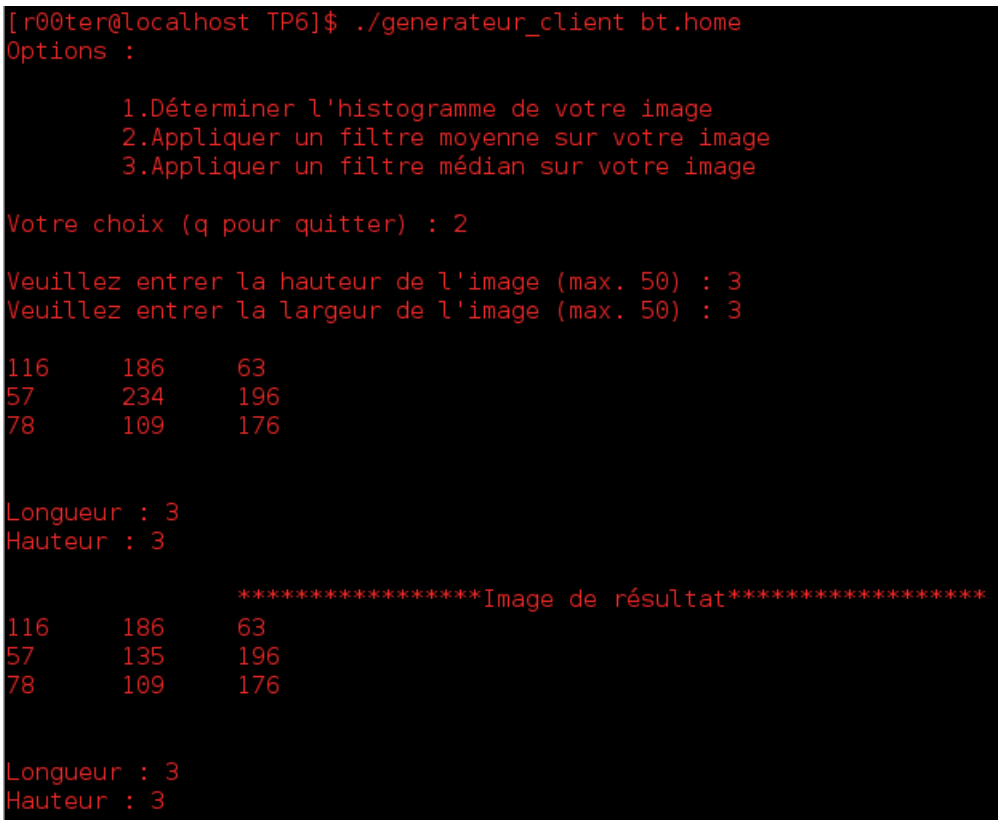

FIGURE A.3 – Exemple d'usage de l'application traitement d'image