Exercice 2 : TP RPC

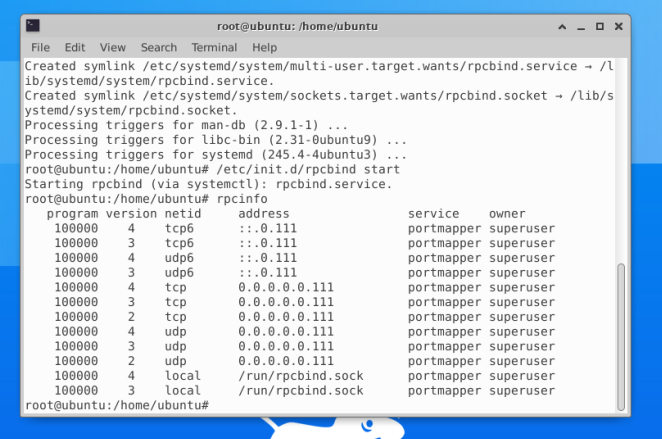
Installation des dependances

Essai de RPC pour hello world code predefine : <https://github.com/riyazathali/RPC-HelloWorld>

REPONSE

**sudo apt install rpcbind**

**/etc/init.d/rpcbind start**



**sudo apt update**

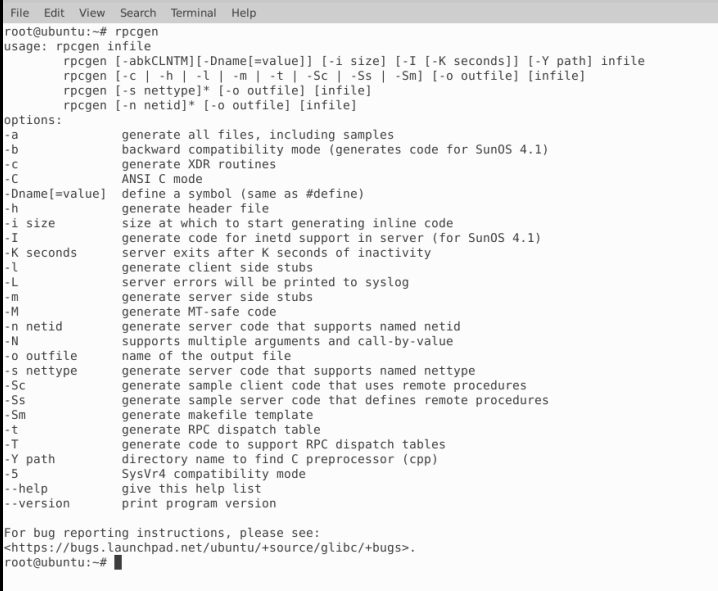
**sudo apt install build-essential**

**sudo apt-get install manpages-dev**

**sudo apt install libc-dev-bin**

**apt install zip**

* rpcgen devrait marcher correctement :



**git clone** [**https://github.com/riyazathali/RPC-HelloWorld**](https://github.com/riyazathali/RPC-HelloWorld)

**cd RPC-HelloWorld/**

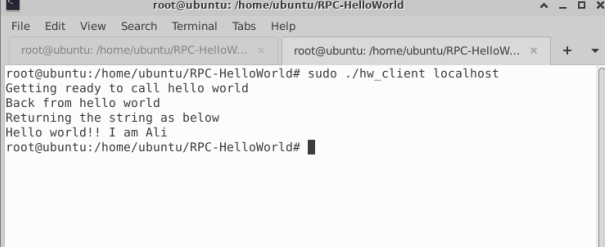
**rpcgen -a -C hw.x**

**make -f Makefile.hw**

**sudo ./hw\_server**

**tapez ctrl-shift T**

**sudo ./hw\_client localhost**



Exercice 3 : Entrainnement C,

3.1 -> Affichage Hello world

3.2 -> Affichage d’un variable ayant comme contenu 10

3.3 -> Afffichage de texte entrée comme paramètre en ligne de commande

3.4 -> Affichage de texte avec plusieurs arguments en ligne de commande

3.5 -> Déclaration d’un variable, saisie d’un nombre et affichage de ce nombre

3.6 -> Creation de fonction somme (entier)

3.7 -> Creation d’une fonction somme avec verification overflow

3.8 -> Creation de fonction somme de deux structures points en utilisant une passage par copie

3.9 -> passage par pointeur de la function somme

3.10 -> Création d’une de la function somme C en utilisant une passage par pointeur

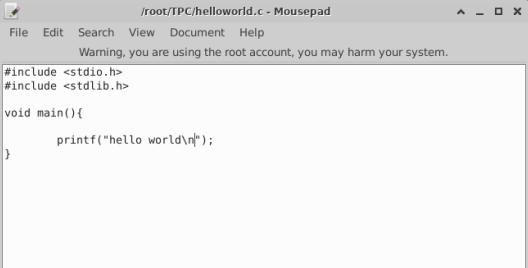
REPONSE 3.1

3.1 -> Affichage Hello world

**mkdir TPC**

**cd TPC/**

**mousepad helloworld.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*void main(){*

*printf("hello world\n");*

*}*

**gcc -c helloworld.c**

**gcc -o helloworld helloworld.o**

**chmod +x helloworld**

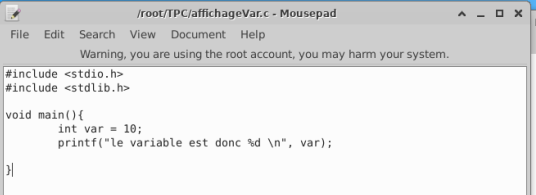
**./helloworld**



REPONSE 3.2

3.2 -> Affichage d’un variable ayant comme contenu 10

**mousepad affichageVar.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*void main(){*

*int var = 10;*

*printf("le variable est donc %d \n", var);*

*}*

**gcc -c affichageVar.c**

**gcc -o affichageVar affichageVar.o**

**chmod +x affichageVar**

**./affichageVar**



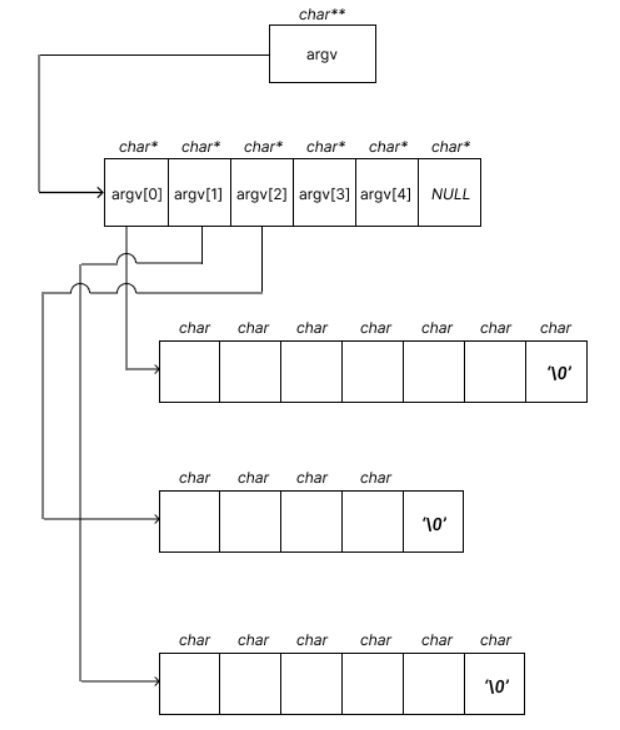
REPONSE 3.3

3.3 -> Afffichage de texte entrée comme paramètre en ligne de commande

La declaration de main deviant c.f <https://stackoverflow.com/questions/3024197/what-does-int-argc-char-argv-mean> :

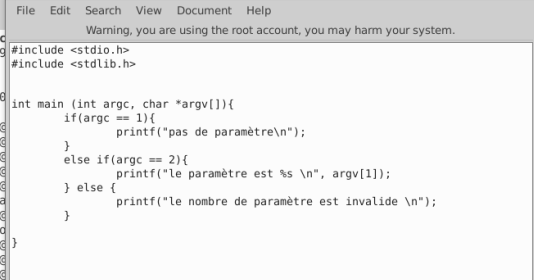
int main (int argc, char \*argv[]); //first declaration

int main (int argc, char \*\*argv); //RE-DECLARATION. Equivalent to the above declaration



* Le code final sera donc :

**mousepad affichage.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*int main (int argc, char \*argv[]){*

*if(argc == 1){*

*printf("pas de paramètre\n");*

*}*

*else if(argc == 2){*

*printf("le paramètre est %s \n", argv[1]);*

*} else {*

*printf("le nombre de paramètre est invalide \n");*

*}*

*}*

**gcc -o affichage affichage.o**

**gcc -c affichage.c**

**gcc -o affichage affichage.o**

**chmod +x affichage**

Test après :

**./affichage**



**./affichage test**



**./affichage "test is ok"**



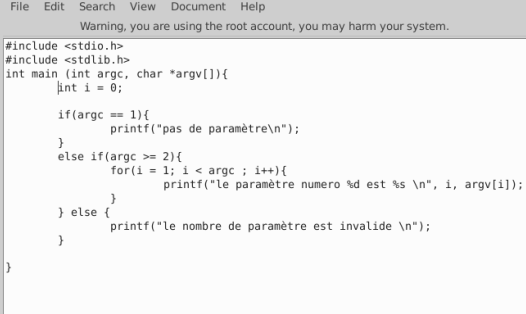
**./affichage test is ok**



REPONSE 3.4

3.4 -> Affichage de texte avec plusieurs arguments en ligne de commande

**mousepad affichageMultiple.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*int main (int argc, char \*argv[]){*

*int i = 0;*

*if(argc == 1){*

*printf("pas de paramè?tre\n");*

*}*

*else if(argc >= 2){*

*for(i = 1; i < argc ; i++){*

*printf("le paramè?tre numero %d est %s \n", i, argv[i]);*

*}*

*} else {*

*printf("le nombre de paramè?tre est invalide \n");*

*}*

*}*

**gcc -c affichageMultiple.c**

**gcc -o affichageMultiple affichageMultiple.o**

**chmod +x affichageMultiple**

**./ affichageMultiple**



**./affichageMultiple test**



**./affichage "test is ok"**



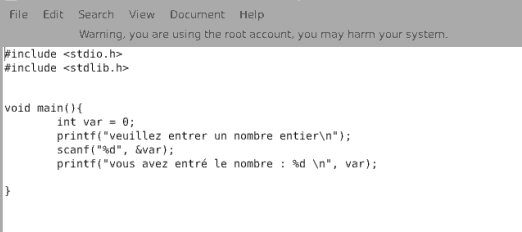
**./affichage test is ok**



REPONSE 3.5

3.5 -> Déclaration d’un variable, saisie d’un nombre et affichage de ce nombre

**mousepad saisie.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*void main(){*

*int var = 0;*

*printf("veuillez entrer un nombre entier\n");*

*scanf("%d", &var);*

*printf("vous avez entré? le nombre : %d \n", var);*

*}*

**gcc -c saisie.c**

**gcc -o saisie saisie.o**

**chmod +x saisie**

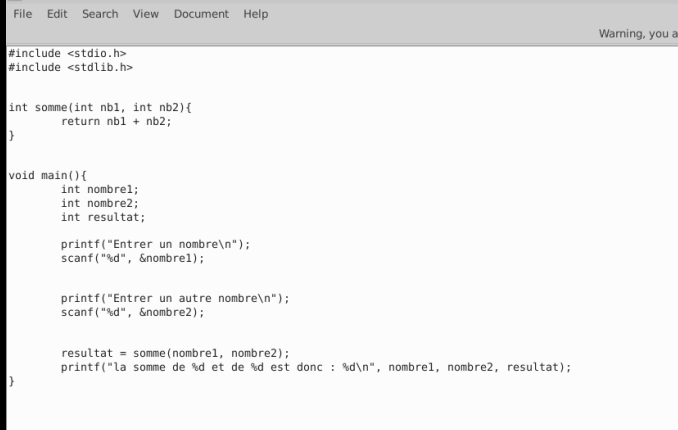
**./saisie**



REPONSE 3.6

3.6 -> Creation de fonction somme (entier)

**mousepad fonc\_somme.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*int somme(int nb1, int nb2){*

*return nb1 + nb2;*

*}*

*void main(){*

*int nombre1;*

*int nombre2;*

*int resultat;*

*printf("Entrer un nombre\n");*

*scanf("%d", &nombre1);*

*printf("Entrer un autre nombre\n");*

*scanf("%d", &nombre2);*

*resultat = somme(nombre1, nombre2);*

*printf("la somme de %d et de %d est donc : %d\n", nombre1, nombre2, resultat);*

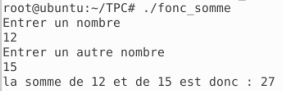
*}*

**gcc -c fonc\_somme.c**

**gcc -o fonc\_somme fonc\_somme.o**

**chmod +x fonc\_somme**

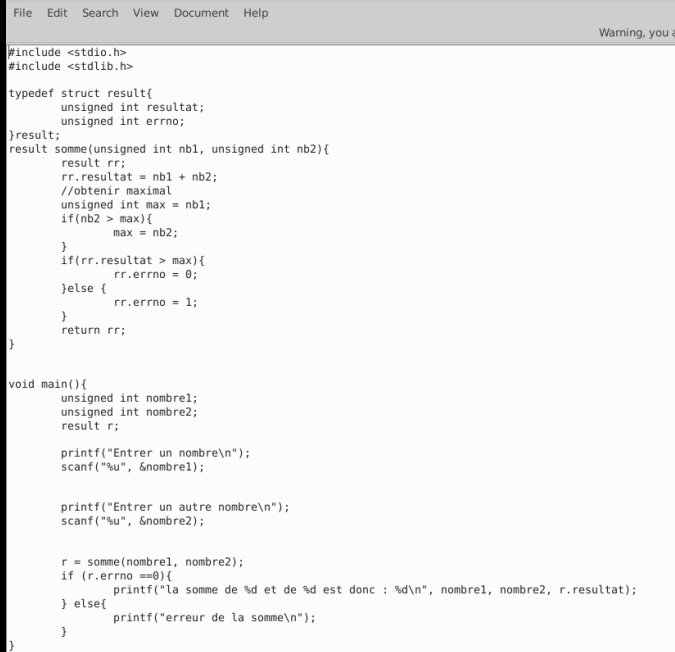
**./fonc\_somme**



REPONSE 3.7

3.7 -> Creation d’une fonction somme avec verification overflow

**mousepad overf\_somme.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*typedef struct result{*

*unsigned int resultat;*

*unsigned int errno;*

*}result;*

*result somme(unsigned int nb1, unsigned int nb2){*

*result rr;*

*rr.resultat = nb1 + nb2;*

*//obtenir maximal*

*unsigned int max = nb1;*

*if(nb2 > max){*

*max = nb2;*

*}*

*if(rr.resultat > max){*

*rr.errno = 0;*

*}else {*

*rr.errno = 1;*

*}*

*return rr;*

*}*

*void main(){*

*unsigned int nombre1;*

*unsigned int nombre2;*

*result r;*

*printf("Entrer un nombre\n");*

*scanf("%u", &nombre1);*

*printf("Entrer un autre nombre\n");*

*scanf("%u", &nombre2);*

*r = somme(nombre1, nombre2);*

*if (r.errno ==0){*

*printf("la somme de %d et de %d est donc : %d\n", nombre1, nombre2, r.resultat);*

*} else{*

*printf("erreur de la somme\n");*

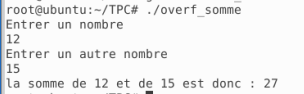
*}*

*}*

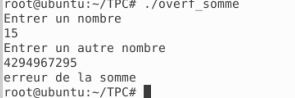
**gcc -c overf\_somme.c**

**gcc -o overf\_somme overf\_somme.o**

**chmod +x overf\_somme**



Le nombre maximal unsigned int est : 4294967295

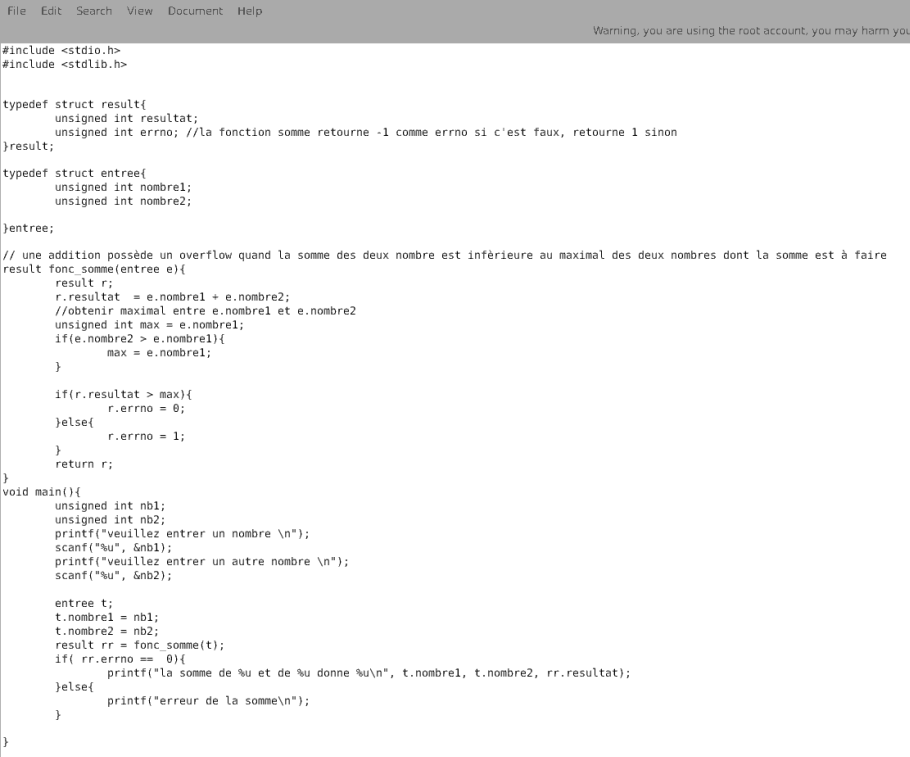


Note bonus pour la verification overf\_somme en utilisant int et non unsigned int ?

REPONSE 3.8 : cf. <https://www.youtube.com/watch?v=cByXgJXaTB4>

3.8 -> Creation d’une structure pour faire une function somme avec verification de overflow

**mousepad somme\_struct.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*typedef struct result{*

*unsigned int resultat;*

*unsigned int errno; //la fonction somme retourne -1 comme errno si c'est faux, retourne 1 sinon*

*}result;*

*typedef struct entree{*

*unsigned int nombre1;*

*unsigned int nombre2;*

*}entree;*

*// une addition possè?de un overflow quand la somme des deux nombre est infè?rieure au maximal des deux nombres dont la somme est à faire*

*result fonc\_somme(entree e){*

*result r;*

*r.resultat = e.nombre1 + e.nombre2;*

*//obtenir maximal entre e.nombre1 et e.nombre2*

*unsigned int max = e.nombre1;*

*if(e.nombre2 > e.nombre1){*

*max = e.nombre1;*

*}*

*if(r.resultat > max){*

*r.errno = 0;*

*}else{*

*r.errno = 1;*

*}*

*return r;*

*}*

*void main(){*

*unsigned int nb1;*

*unsigned int nb2;*

*printf("veuillez entrer un nombre \n");*

*scanf("%u", &nb1);*

*printf("veuillez entrer un autre nombre \n");*

*scanf("%u", &nb2);*

*entree t;*

*t.nombre1 = nb1;*

*t.nombre2 = nb2;*

*result rr = fonc\_somme(t);*

*if( rr.errno == 0){*

*printf("la somme de %u et de %u donne %u\n", t.nombre1, t.nombre2, rr.resultat);*

*}else{*

*printf("erreur de la somme\n");*

*}*

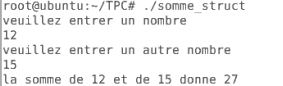
*}*

**gcc -c somme\_struct.c**

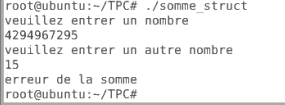
**gcc -o somme\_struct somme\_struct.o**

**chmod +x somme\_struct**

**./somme\_struct**



Le nombre maximal unsigned int est : 4294967295

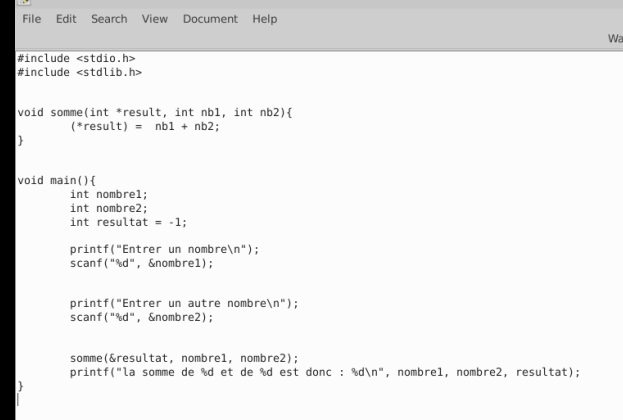


* Note BONUS : Pour un code en C, non signé avec verification overflow ?

REPONSE 3.9

3.9 -> passage par pointeur de la function somme

**mousepad pointeur\_somme.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*void somme(int \*result, int nb1, int nb2){*

*(\*result) = nb1 + nb2;*

*}*

*void main(){*

*int nombre1;*

*int nombre2;*

*int resultat = -1;*

*printf("Entrer un nombre\n");*

*scanf("%d", &nombre1);*

*printf("Entrer un autre nombre\n");*

*scanf("%d", &nombre2);*

*somme(&resultat, nombre1, nombre2);*

*printf("la somme de %d et de %d est donc : %d\n", nombre1, nombre2, resultat);*

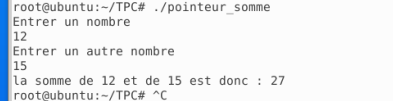
*}*

**gcc -c pointeur\_somme.c**

**gcc -o pointeur\_somme pointeur\_somme.o**

**chmod +x pointeur\_somme**

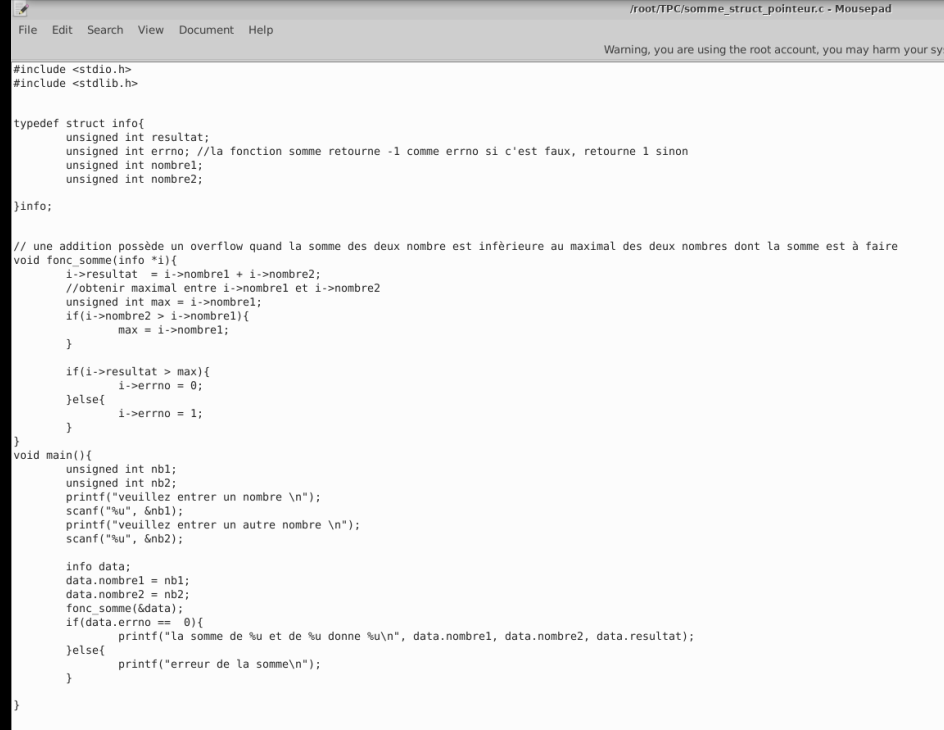
**./pointeur\_somme**



REPONSE 3.10 :

3.10 -> Création d’une de la function somme C en utilisant une passage par pointeur

**mousepad somme\_struct\_pointeur.c**



*#include <stdio.h>*

*#include <stdlib.h>*

*typedef struct info{*

*unsigned int resultat;*

*unsigned int errno; //la fonction somme retourne -1 comme errno si c'est faux, retourne 1 sinon*

*unsigned int nombre1;*

*unsigned int nombre2;*

*}info;*

*// une addition possè?de un overflow quand la somme des deux nombre est infè?rieure au maximal des deux nombres dont la somme est à? faire*

*void fonc\_somme(info \*i){*

*i->resultat = i->nombre1 + i->nombre2;*

*//obtenir maximal entre i->nombre1 et i->nombre2*

*unsigned int max = i->nombre1;*

*if(i->nombre2 > i->nombre1){*

*max = i->nombre1;*

*}*

*if(i->resultat > max){*

*i->errno = 0;*

*}else{*

*i->errno = 1;*

*}*

*}*

*void main(){*

*unsigned int nb1;*

*unsigned int nb2;*

*printf("veuillez entrer un nombre \n");*

*scanf("%u", &nb1);*

*printf("veuillez entrer un autre nombre \n");*

*scanf("%u", &nb2);*

*info data;*

*data.nombre1 = nb1;*

*data.nombre2 = nb2;*

*fonc\_somme(&data);*

*if(data.errno == 0){*

*printf("la somme de %u et de %u donne %u\n", data.nombre1, data.nombre2, data.resultat);*

*}else{*

*printf("erreur de la somme\n");*

*}*

*}*

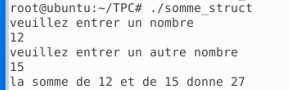
**gcc -c somme\_struct\_pointeur.c**

**gcc -c somme\_struct\_pointeur.c**

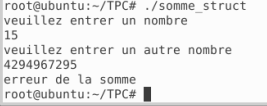
**gcc -o somme\_struct\_pointeur somme\_struct\_pointeur.o**

**chmod +x somme\_struct\_pointeur**

**./somme\_struct**



Le nombre maximal unsigned int est : 4294967295



TP RPC cf. lecons :

**mkdir RPC**

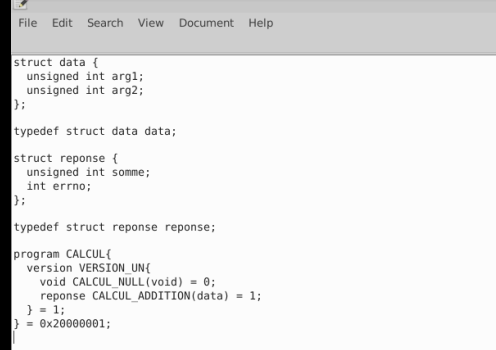
**cd RPC/**

**ls**

**mkdir somme**

**cd somme/**

**mousepad calcul.x**



*struct data {*

*unsigned int arg1;*

*unsigned int arg2;*

*};*

*typedef struct data data;*

*struct reponse {*

*unsigned int somme;*

*int errno;*

*};*

*typedef struct reponse reponse;*

*program CALCUL{*

*version VERSION\_UN{*

*void CALCUL\_NULL(void) = 0;*

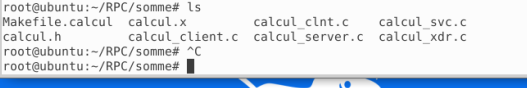
*reponse CALCUL\_ADDITION(data) = 1;*

*} = 1;*

*} = 0x20000001;*

**rpcgen -a calcul.x**

**ls**



* rpcgen a généré des code c et Makefile pour nous

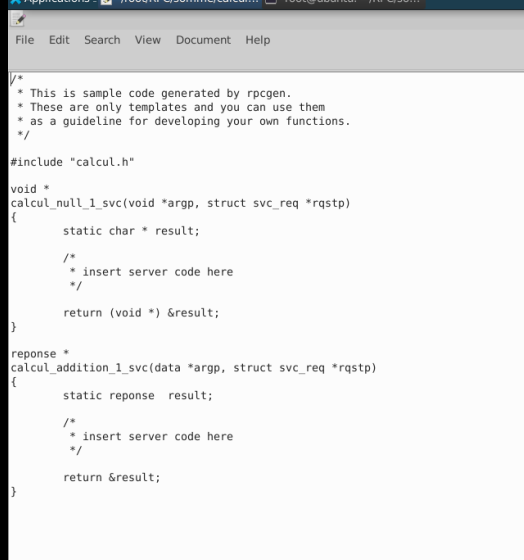
**gcc -c calcul\_xdr.c**

**gcc -c calcul\_clnt.c**

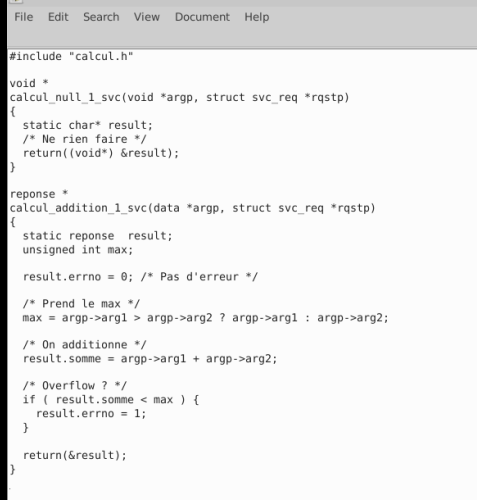
**gcc -c calcul\_svc.c**

* En regardant dans le fichier :

**mousepad calcul\_server.c**



* Il faut modifier le code en implementant l’addition à faire au niveau de serveur en particulier dans la function calcul\_addition\_1\_svc



*#include "calcul.h"*

*void \**

*calcul\_null\_1\_svc(void \*argp, struct svc\_req \*rqstp)*

*{*

*static char\* result;*

*/\* Ne rien faire \*/*

*return((void\*) &result);*

*}*

*reponse \**

*calcul\_addition\_1\_svc(data \*argp, struct svc\_req \*rqstp)*

*{*

*static reponse result;*

*unsigned int max;*

*result.errno = 0; /\* Pas d'erreur \*/*

*/\* Prend le max \*/*

*max = argp->arg1 > argp->arg2 ? argp->arg1 : argp->arg2;*

*/\* On additionne \*/*

*result.somme = argp->arg1 + argp->arg2;*

*/\* Overflow ? \*/*

*if ( result.somme < max ) {*

*result.errno = 1;*

*}*

*return(&result);*

*}*

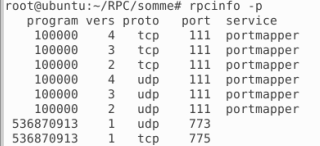
**gcc -c calcul\_server.c**

**gcc -o server calcul\_svc.o calcul\_server.o calcul\_xdr.o**

**./server &**



**rpcinfo -p**



**mkdir client**

**cp calcul.x client/**

**cd client**

**rpcgen -a calcul.x**

**ls**

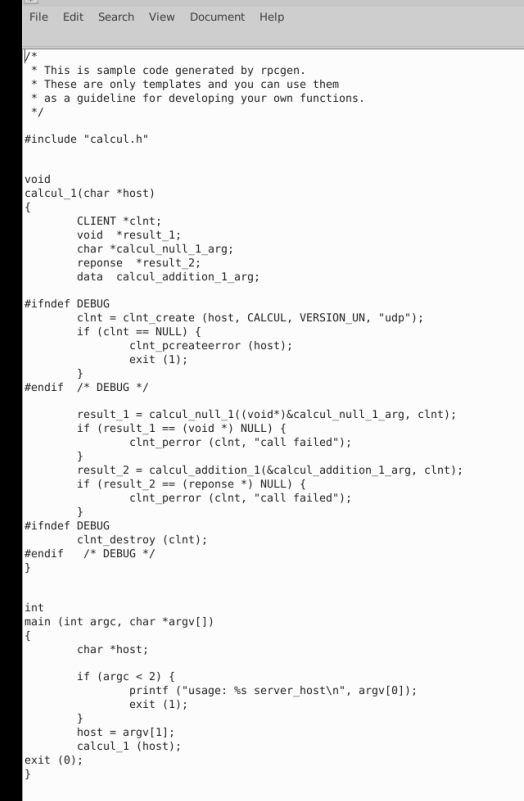
**gcc -c calcul\_xdr.c**

**gcc -c calcul\_clnt.c**

**gcc -c calcul\_svc.c**

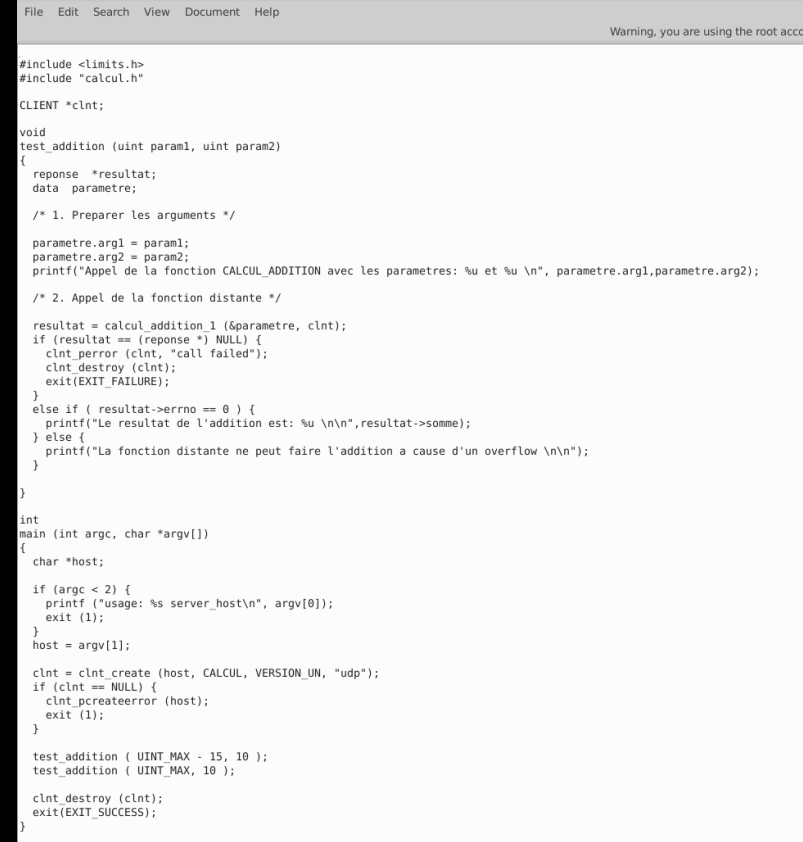
* Regardons d’abord le code calcul\_client.c par :

**mousepad calcul\_client.c**



* Changeons par le code adequate en particulier dans la fonction main tout en créant la function test\_addition. Aprés modification, le code sera :

mousepad calcul\_client.c



*#include <limits.h>*

*#include "calcul.h"*

*CLIENT \*clnt;*

*void*

*test\_addition (uint param1, uint param2)*

*{*

*reponse \*resultat;*

*data parametre;*

*/\* 1. Preparer les arguments \*/*

*parametre.arg1 = param1;*

*parametre.arg2 = param2;*

*printf("Appel de la fonction CALCUL\_ADDITION avec les parametres: %u et %u \n", parametre.arg1,parametre.arg2);*

*/\* 2. Appel de la fonction distante \*/*

*resultat = calcul\_addition\_1 (&parametre, clnt);*

*if (resultat == (reponse \*) NULL) {*

*clnt\_perror (clnt, "call failed");*

*clnt\_destroy (clnt);*

*exit(EXIT\_FAILURE);*

*}*

*else if ( resultat->errno == 0 ) {*

*printf("Le resultat de l'addition est: %u \n\n",resultat->somme);*

*} else {*

*printf("La fonction distante ne peut faire l'addition a cause d'un overflow \n\n");*

*}*

*}*

*int*

*main (int argc, char \*argv[])*

*{*

*char \*host;*

*if (argc < 2) {*

*printf ("usage: %s server\_host\n", argv[0]);*

*exit (1);*

*}*

*host = argv[1];*

*clnt = clnt\_create (host, CALCUL, VERSION\_UN, "udp");*

*if (clnt == NULL) {*

*clnt\_pcreateerror (host);*

*exit (1);*

*}*

*test\_addition ( UINT\_MAX - 15, 10 );*

*test\_addition ( UINT\_MAX, 10 );*

*clnt\_destroy (clnt);*

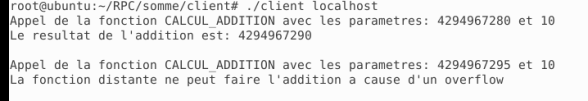
*exit(EXIT\_SUCCESS);*

*}*

**gcc -c calcul\_client.c**

**gcc -o client calcul\_client.o calcul\_clnt.o calcul\_xdr.o**

**./client localhost**



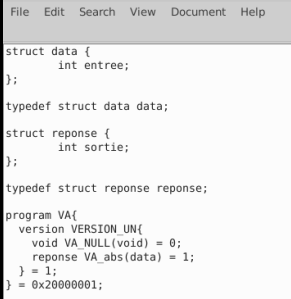
Exercice : Ecrire le code en C pour faire une Valeure absolue distante

**cd RPC**

**mkdir valeurabsolue**

**cd valeurabsolue**

**mousepad va.x**



*struct data {*

*int entree;*

*};*

*typedef struct data data;*

*struct reponse {*

*int sortie;*

*};*

*typedef struct reponse reponse;*

*program VA{*

*version VERSION\_UN{*

*void VA\_NULL(void) = 0;*

*reponse VA\_abs(data) = 1;*

*} = 1;*

*} = 0x20000001;*

**rpcgen -a va.x**



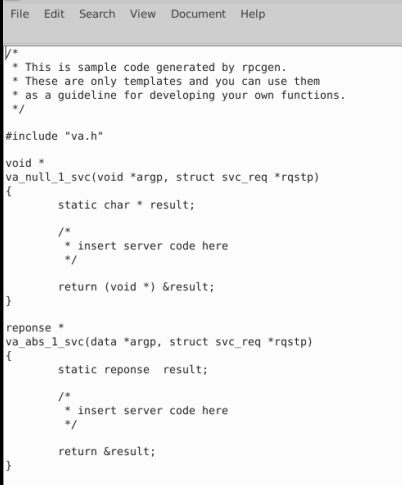
**gcc -c va\_xdr.c**

**gcc -c va\_clnt.c**

**gcc -c va\_svc.c**

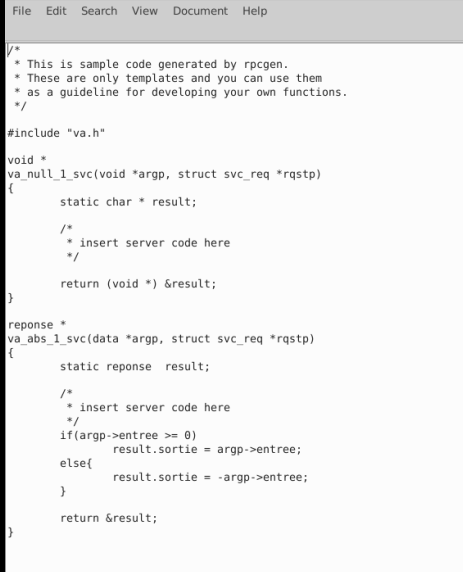
**mousepad va\_server.c**

* Verifions d’abord le code par défaut



* Modifions le code de la function abs

**mousepad va\_server.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 "va.h"*

*void \**

*va\_null\_1\_svc(void \*argp, struct svc\_req \*rqstp)*

*{*

*static char \* result;*

*/\**

*\* insert server code here*

*\*/*

*return (void \*) &result;*

*}*

*reponse \**

*va\_abs\_1\_svc(data \*argp, struct svc\_req \*rqstp)*

*{*

*static reponse result;*

*/\**

*\* insert server code here*

*\*/*

*if(argp->entree >= 0)*

*result.sortie = argp->entree;*

*else{*

*result.sortie = -argp->entree;*

*}*

*return &result;*

*}*

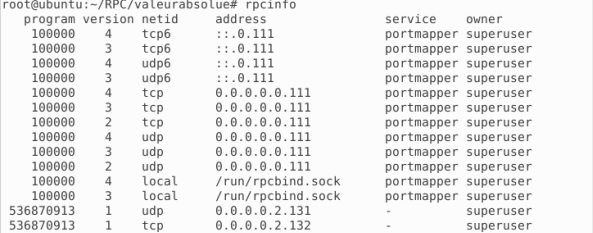
**gcc -c va\_server.c**

**gcc -o server va\_svc.o va\_server.o va\_xdr.o**

**./server &**



**rpcinfo**



**mkdir client**

**cd client**

**cp ../va.x ../client/**

**rpcgen -a va.x**

**ls**



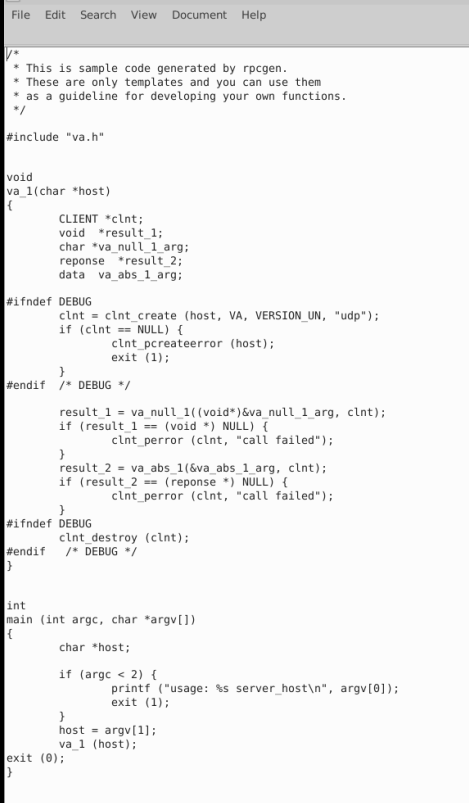
**gcc -c va\_xdr.c**

**gcc -c va\_clnt.c**

**gcc -c va\_svc.c**

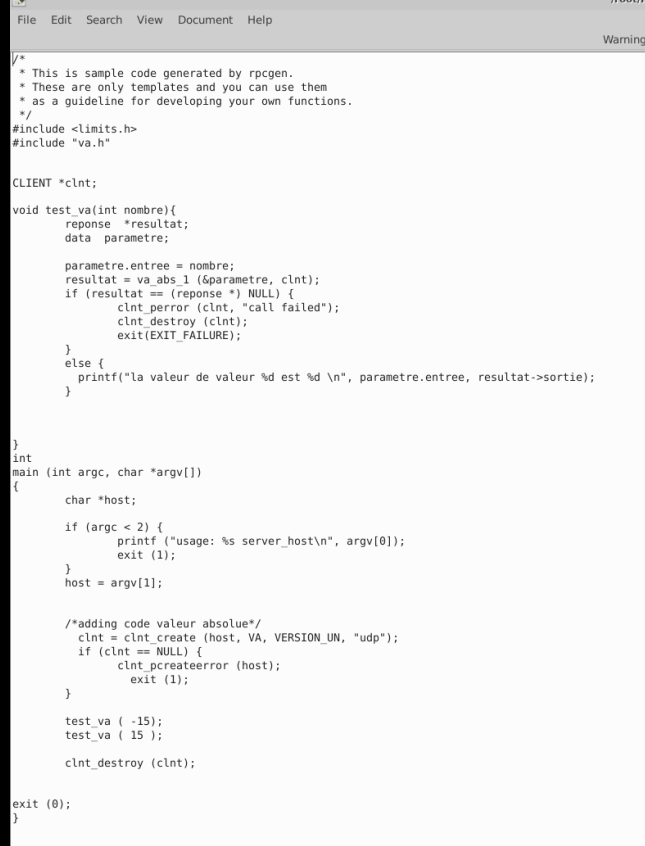
**mousepad va\_client.c**

* Le code initial dans va\_client est:



* Modifions le code dans le main pour faire l’appel de Valeur absolue

**mousepad va\_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 <limits.h>*

*#include "va.h"*

*CLIENT \*clnt;*

*void test\_va(int nombre){*

*reponse \*resultat;*

*data parametre;*

*parametre.entree = nombre;*

*resultat = va\_abs\_1 (&parametre, clnt);*

*if (resultat == (reponse \*) NULL) {*

*clnt\_perror (clnt, "call failed");*

*clnt\_destroy (clnt);*

*exit(EXIT\_FAILURE);*

*}*

*else {*

*printf("la valeur de valeur %d est %d \n", parametre.entree, resultat->sortie);*

*}*

*}*

*int*

*main (int argc, char \*argv[])*

*{*

*char \*host;*

*if (argc < 2) {*

*printf ("usage: %s server\_host\n", argv[0]);*

*exit (1);*

*}*

*host = argv[1];*

*/\*adding code valeur absolue\*/*

*clnt = clnt\_create (host, VA, VERSION\_UN, "udp");*

*if (clnt == NULL) {*

*clnt\_pcreateerror (host);*

*exit (1);*

*}*

*test\_va ( -15);*

*test\_va ( 15 );*

*clnt\_destroy (clnt);*

*exit (0);*

*}*

**gcc -o client va\_client.o va\_clnt.o va\_xdr.o**

**./client localhost**



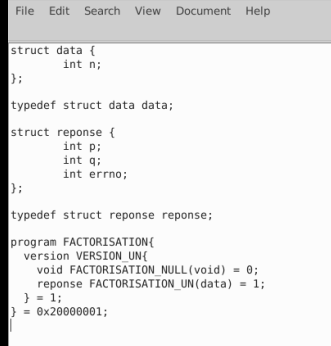
EXERCICE

Faire un code C RPC permettant de factoriser un nombre en produit de deux nombres premiers

**cd RPC**

**mkdir factorisation**

**mousepad factorisation.x**



*struct data {*

*int n;*

*};*

*typedef struct data data;*

*struct reponse {*

*int p;*

*int q;*

*int errno;*

*};*

*typedef struct reponse reponse;*

*program FACTORISATION{*

*version VERSION\_UN{*

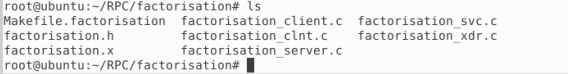
*void FACTORISATION\_NULL(void) = 0;*

*reponse FACTORISATION\_UN(data) = 1;*

*} = 1;*

*} = 0x20000001;*

**rpcgen -a factorisation.x**



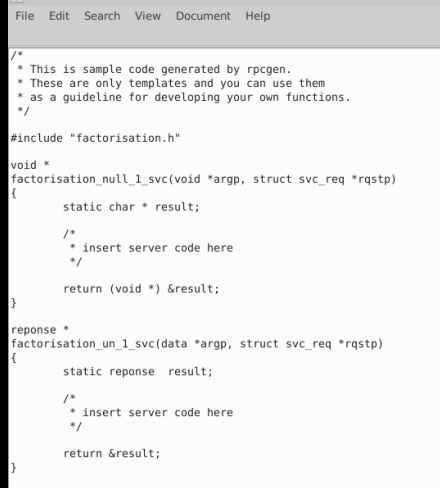
**gcc -c factorisation\_xdr.c**

**gcc -c factorisation\_clnt.c**

**gcc -c factorisation\_svc.c**

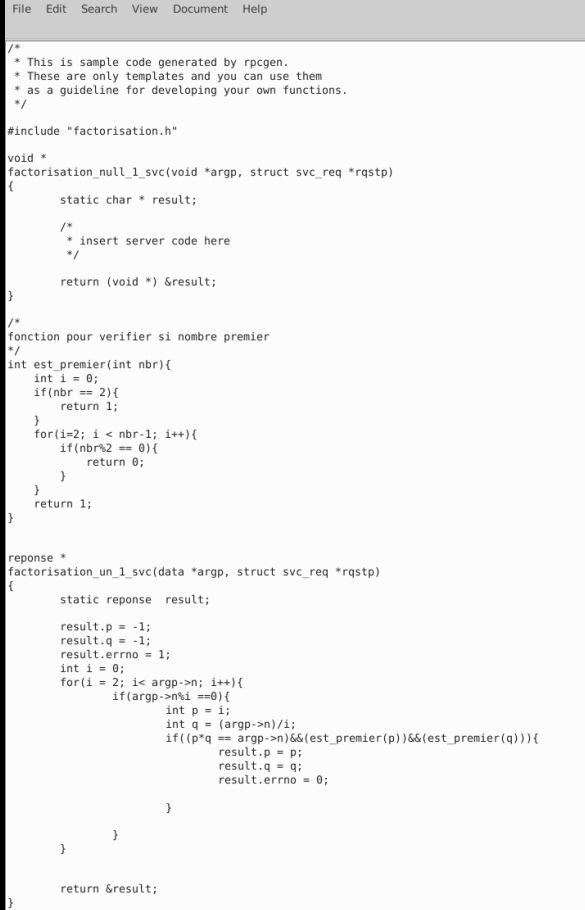
**mousepad factorisation\_server.c**

* Verifions d’abord le code par défaut



* Modifions le code de la function factorisation

**mousepad factorisation\_server.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 "factorisation.h"*

*void \**

*factorisation\_null\_1\_svc(void \*argp, struct svc\_req \*rqstp)*

*{*

*static char \* result;*

*/\**

*\* insert server code here*

*\*/*

*return (void \*) &result;*

*}*

*/\**

*fonction pour verifier si nombre premier*

*\*/*

*int est\_premier(int nbr){*

*int i = 0;*

*if(nbr == 2){*

*return 1;*

*}*

*for(i=2; i < nbr-1; i++){*

*if(nbr%2 == 0){*

*return 0;*

*}*

*}*

*return 1;*

*}*

*reponse \**

*factorisation\_un\_1\_svc(data \*argp, struct svc\_req \*rqstp)*

*{*

*static reponse result;*

*result.p = -1;*

*result.q = -1;*

*result.errno = 1;*

*int i = 0;*

*for(i = 2; i< argp->n; i++){*

*if(argp->n%i ==0){*

*int p = i;*

*int q = (argp->n)/i;*

*if((p\*q == argp->n)&&(est\_premier(p))&&(est\_premier(q))){*

*result.p = p;*

*result.q = q;*

*result.errno = 0;*

*}*

*}*

*}*

*return &result;*

*}*

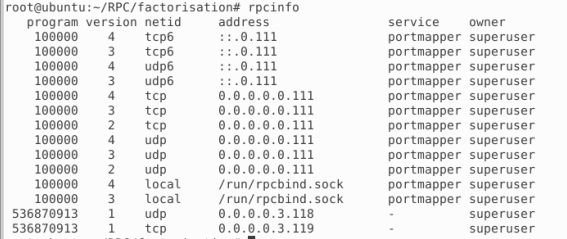
**gcc -c factorisation\_server.c**

**gcc -o server factorisation\_svc.o factorisation\_server.o factorisation\_xdr.o**

**./server &**



**rpcinfo**



**mkdir client**

**cd client**

**cp ../factorisation.x ../client/**

**rpcgen -a factorisation.x**

**ls**



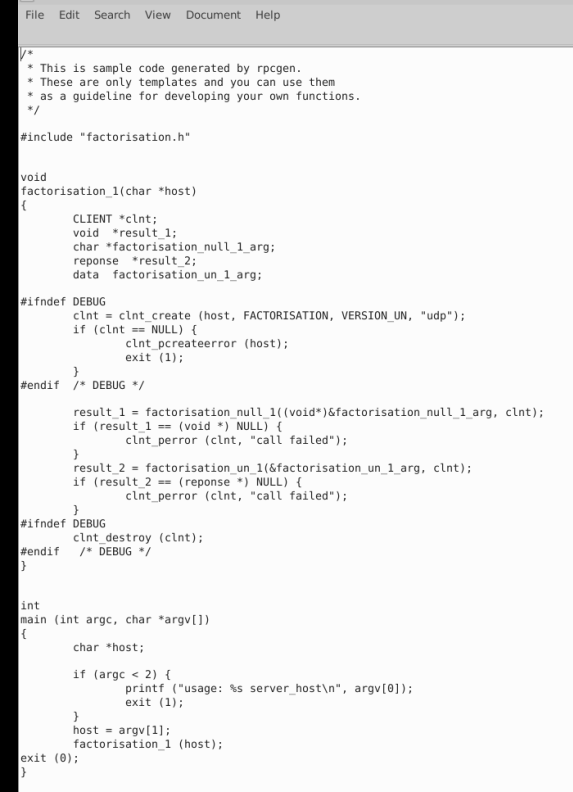
**gcc -c factorisation\_xdr.c**

**gcc -c factorisation\_clnt.c**

**gcc -c factorisation\_svc.c**

**mousepad factorisation\_client.c**

* Le code initial dans factorisation\_client est:



* Modifions le code dans le main pour faire l’appel de la function factorisation

**mousepad factorisation\_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 "factorisation.h"*

*CLIENT \*clnt;*

*void test\_factorisation(int nombre){*

*reponse \*resultat;*

*data parametre;*

*parametre.n = nombre;*

*resultat = factorisation\_un\_1 (&parametre, clnt);*

*if (resultat == (reponse \*) NULL) {*

*clnt\_perror (clnt, "call failed");*

*clnt\_destroy (clnt);*

*exit(EXIT\_FAILURE);*

*}*

*else {*

*if(resultat->errno != 1){*

*printf("Donc %d = %d \* %d \n", parametre.n, resultat->p, resultat->q);*

*}else{*

*printf("Erreur de factorisation\n");*

*}*

*}*

*}*

*int*

*main (int argc, char \*argv[])*

*{*

*char \*host;*

*if (argc < 2) {*

*printf ("usage: %s server\_host\n", argv[0]);*

*exit (1);*

*}*

*host = argv[1];*

*clnt = clnt\_create (host, FACTORISATION, VERSION\_UN, "udp");*

*if (clnt == NULL) {*

*clnt\_pcreateerror (host);*

*exit (1);*

*}*

*test\_factorisation (77);*

*test\_factorisation ( 21 );*

*test\_factorisation ( 12 );*

*clnt\_destroy (clnt);*

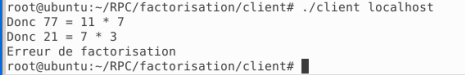
*exit (0);*

*}*

**gcc -c factorisation\_client.c**

**gcc -o client factorisation\_client.o factorisation\_clnt.o factorisation\_xdr.o**

**./client localhost**



Remarques :

LES CODES DE C ET RPC SERONT DANS GITHUB :

* <https://github.com/ImsicatcherBastienbaranoff/RPC>
* <https://github.com/ImsicatcherBastienbaranoff/TPC>

EXERCICE SOCKET :

**mkdir SOCKET**

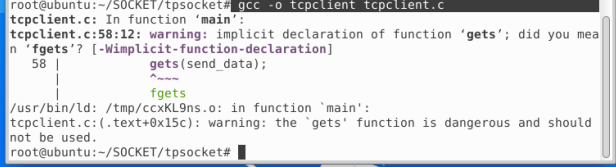
**cd SOCKET/**

**git clone** [**https://github.com/ImsicatcherBastienbaranoff/tp**](https://github.com/ImsicatcherBastienbaranoff/tp)**socket**

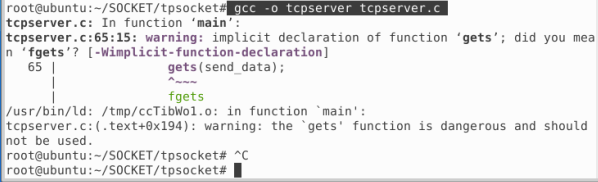
**cd tpsocket**

**unzip client-server.zip**

**gcc -o tcpclient tcpclient.c**



**gcc -o tcpserver tcpserver.c**

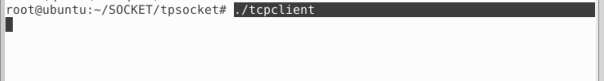


**./tcpserver**



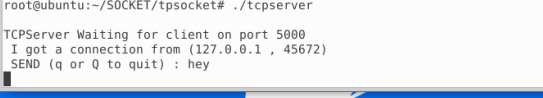
Ctrl + shift + t

**./tcpclient**

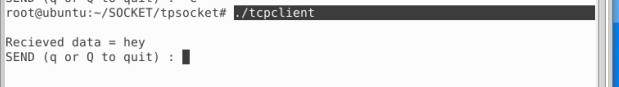


Dans l’autre terminal : 

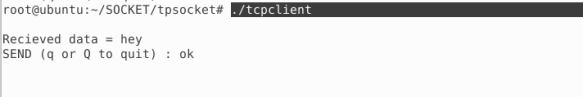
Tapez hey :



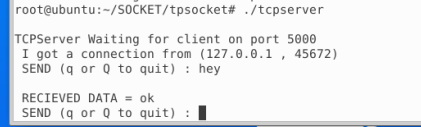
Dans l’autre terminal :



Tapez : ok



Dans l’autre terminal :



POUR PLUS DE PRATIQUE ET COURS: VEUILLEZ FAIRE LA LECTURE DES DEUX CHAINES YOUTUBE:

* <https://www.youtube.com/playlist?list=PLF7E20B552F65C51C>
* <https://www.youtube.com/playlist?list=PLOholRFa862SX9Ypgk3vsByrfMT_hMq-l>