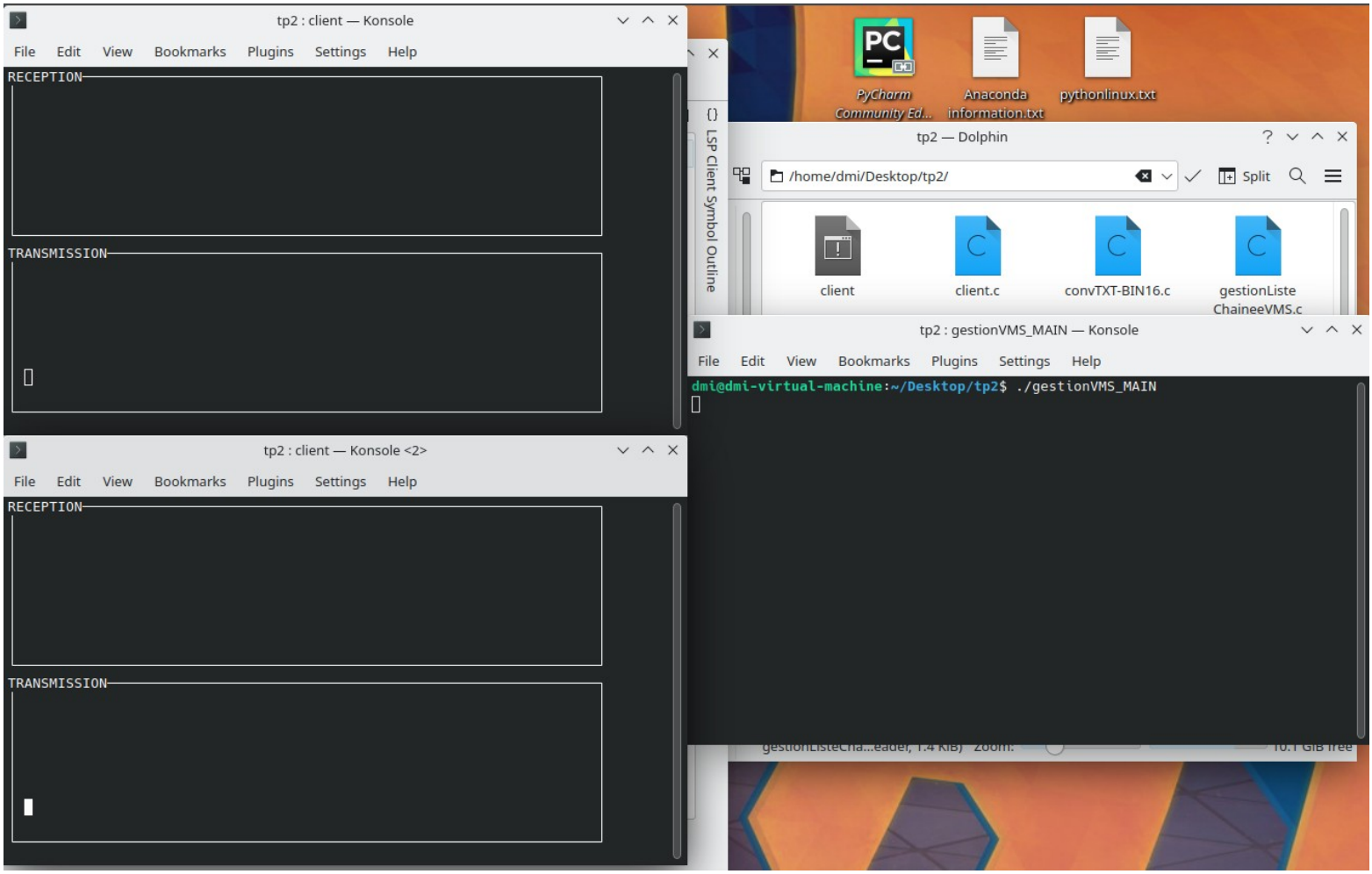
Systèmes d’exploitation (SIF1015)

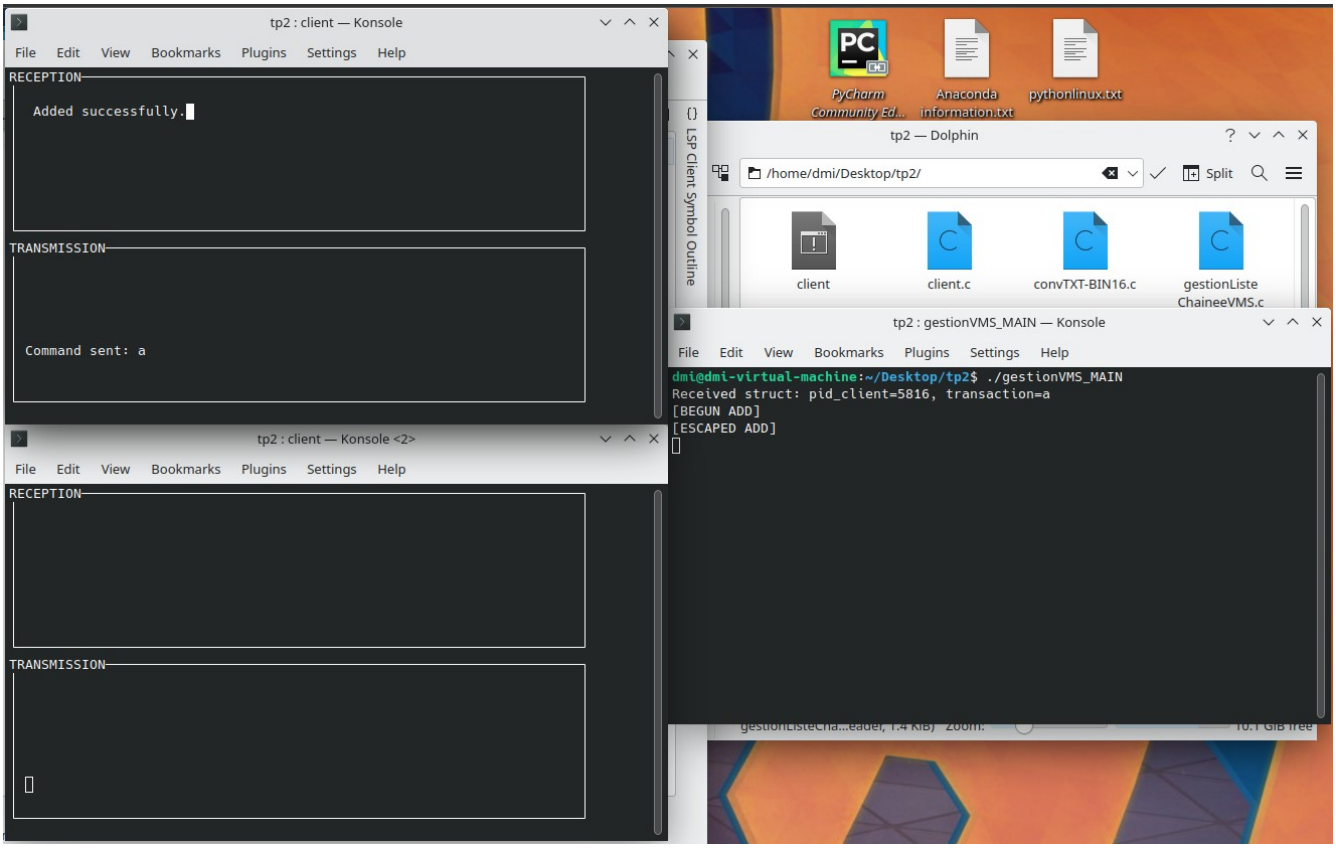
TP 2

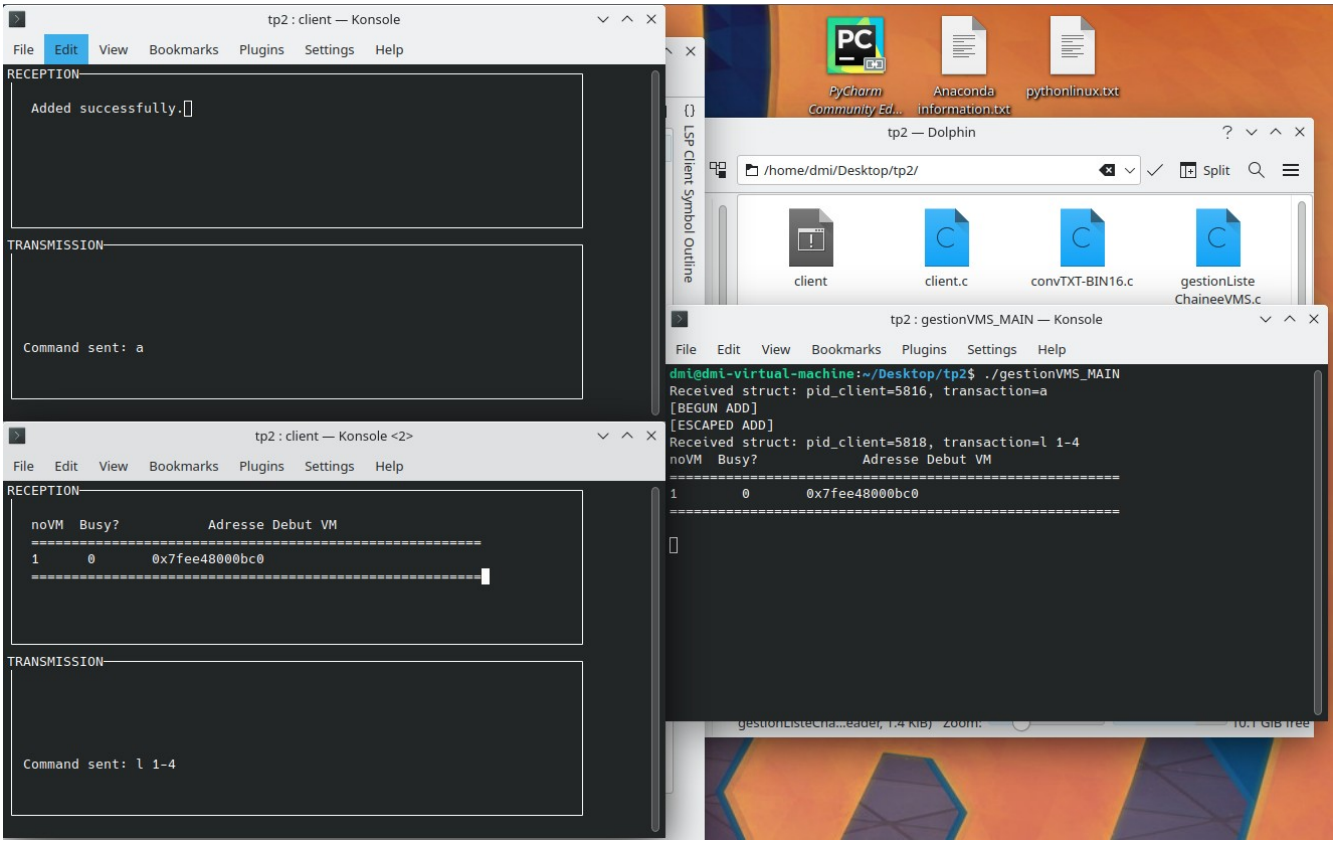
Christian Robert

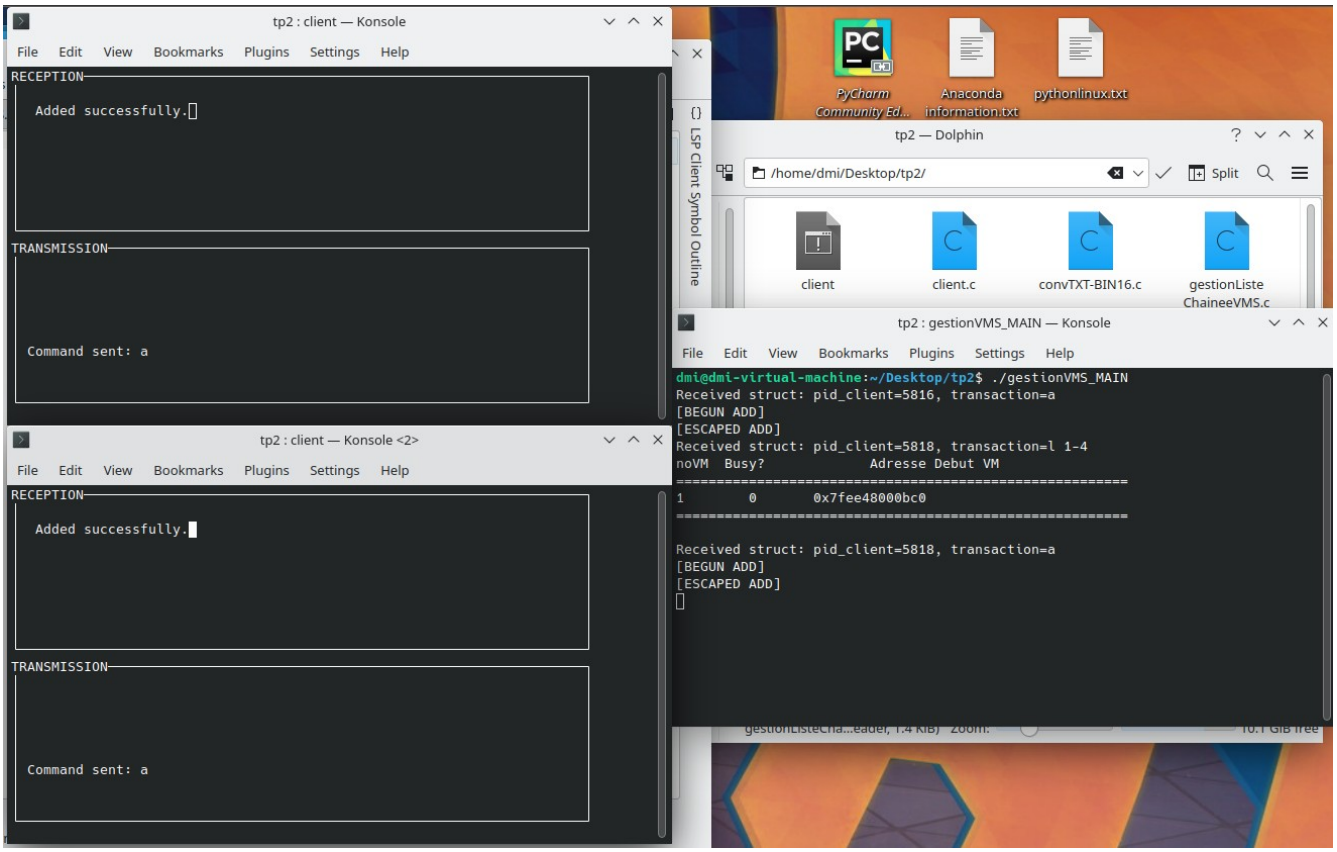
11 novembre 2023

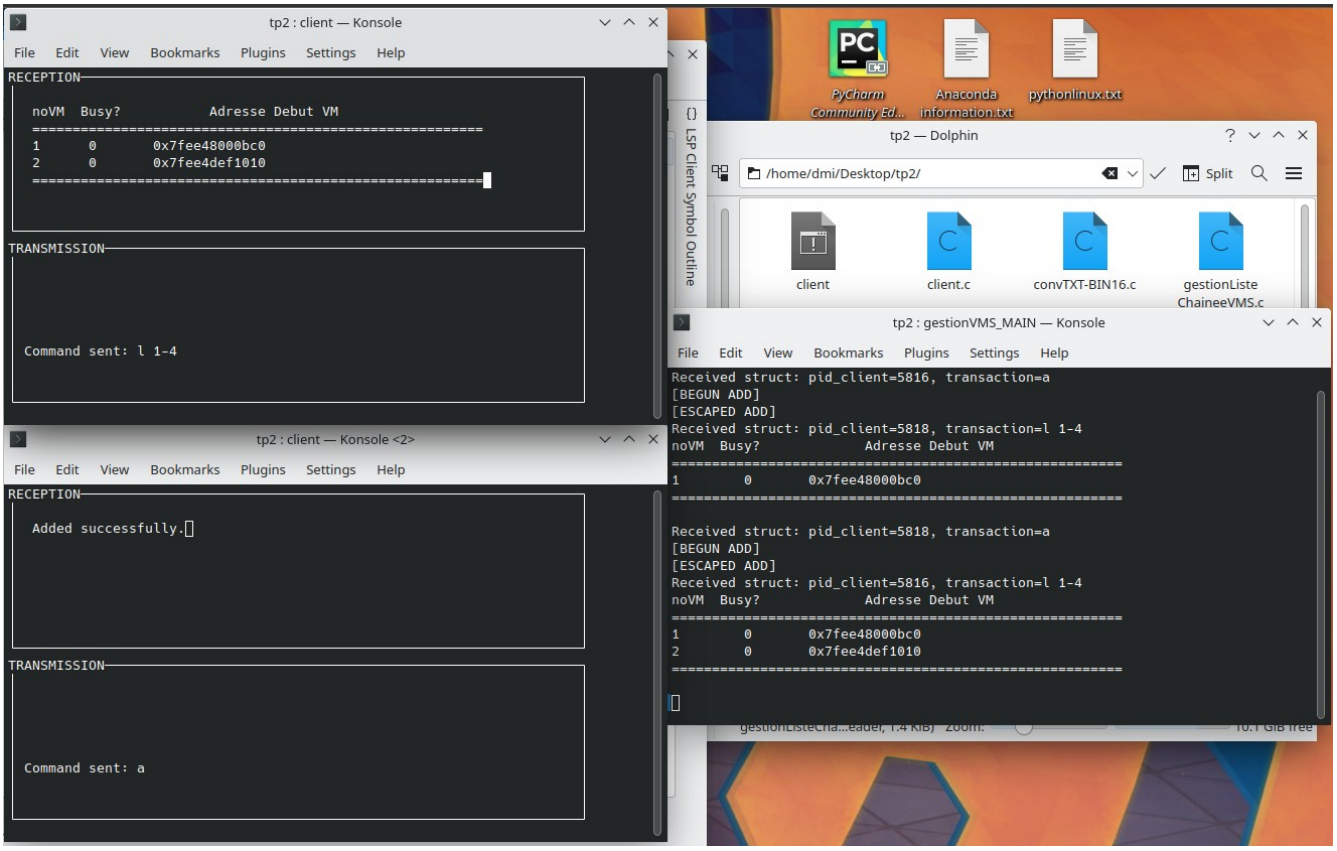
Exécution :
Le document du tp2 a mentionné « l'exécution du fichier de transactions trans2.txt. » C'était peut-être une erreur, mais bon : j'ai reproduit cette séquence d'opérations manuellement.

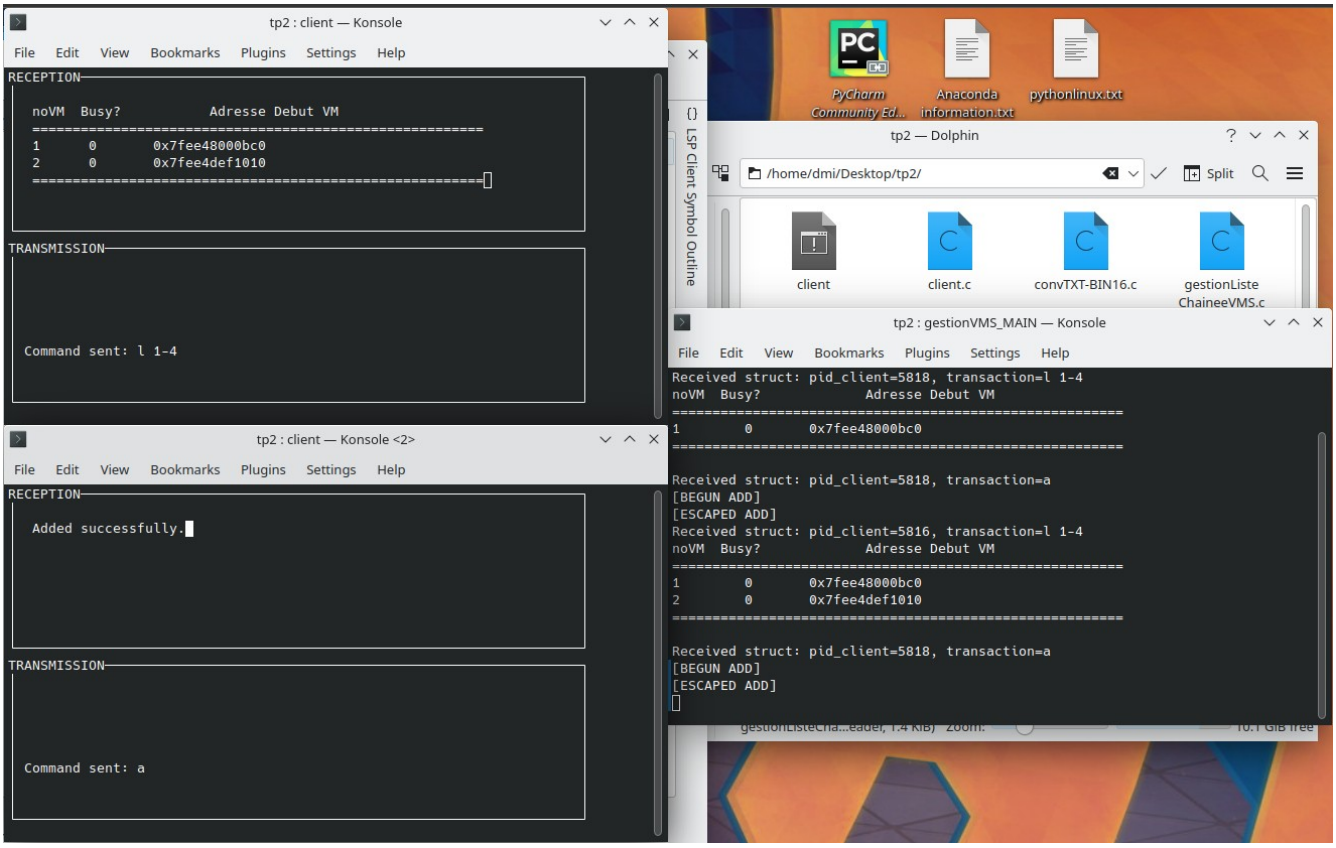


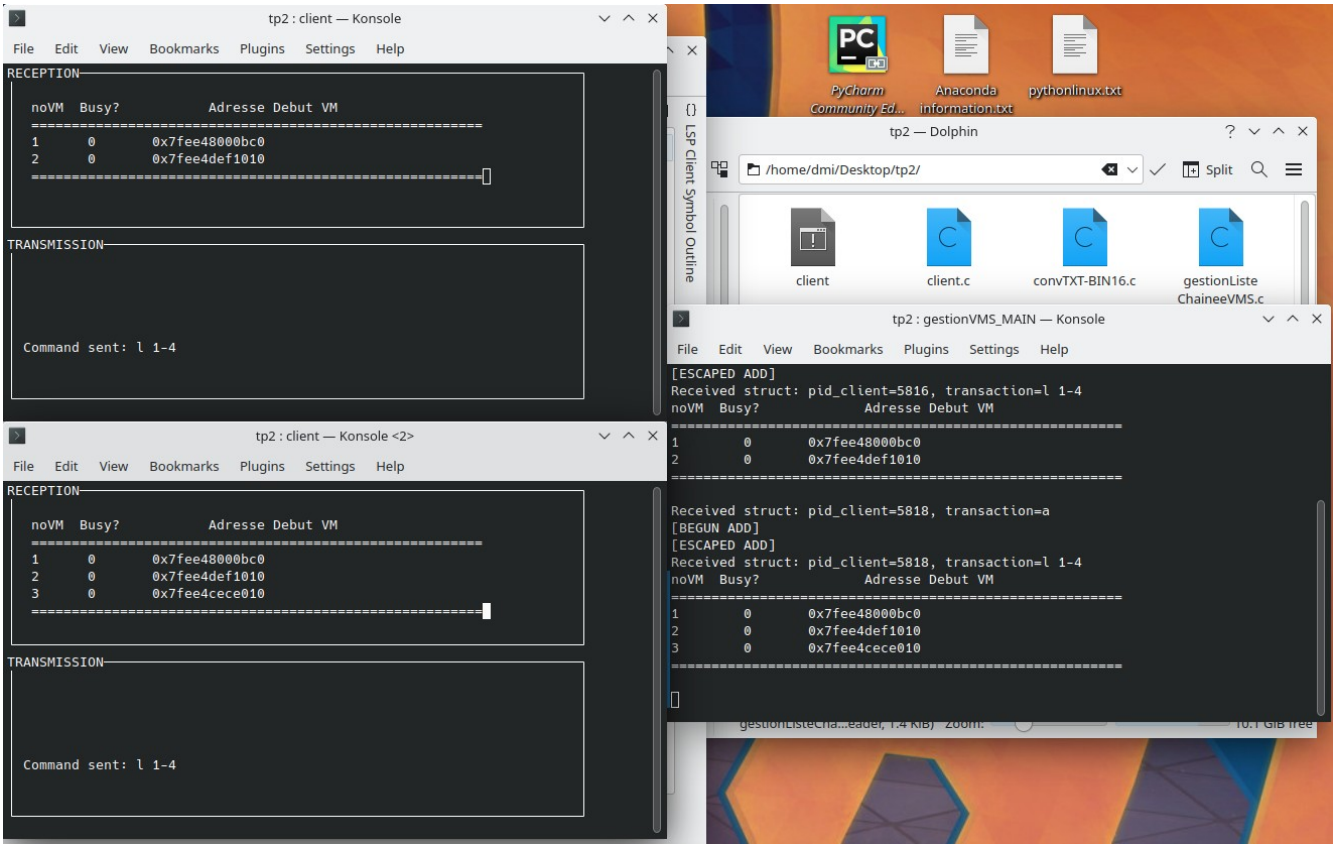

tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
TRANSMISSION
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
dmi@dmi-virtual-machine:~/Desktop/tp2$ ./gestionVMS_MAIN
Received struct: pid_client=5816, transaction=a
[BEGUN ADD]
[ESCAPED ADD]

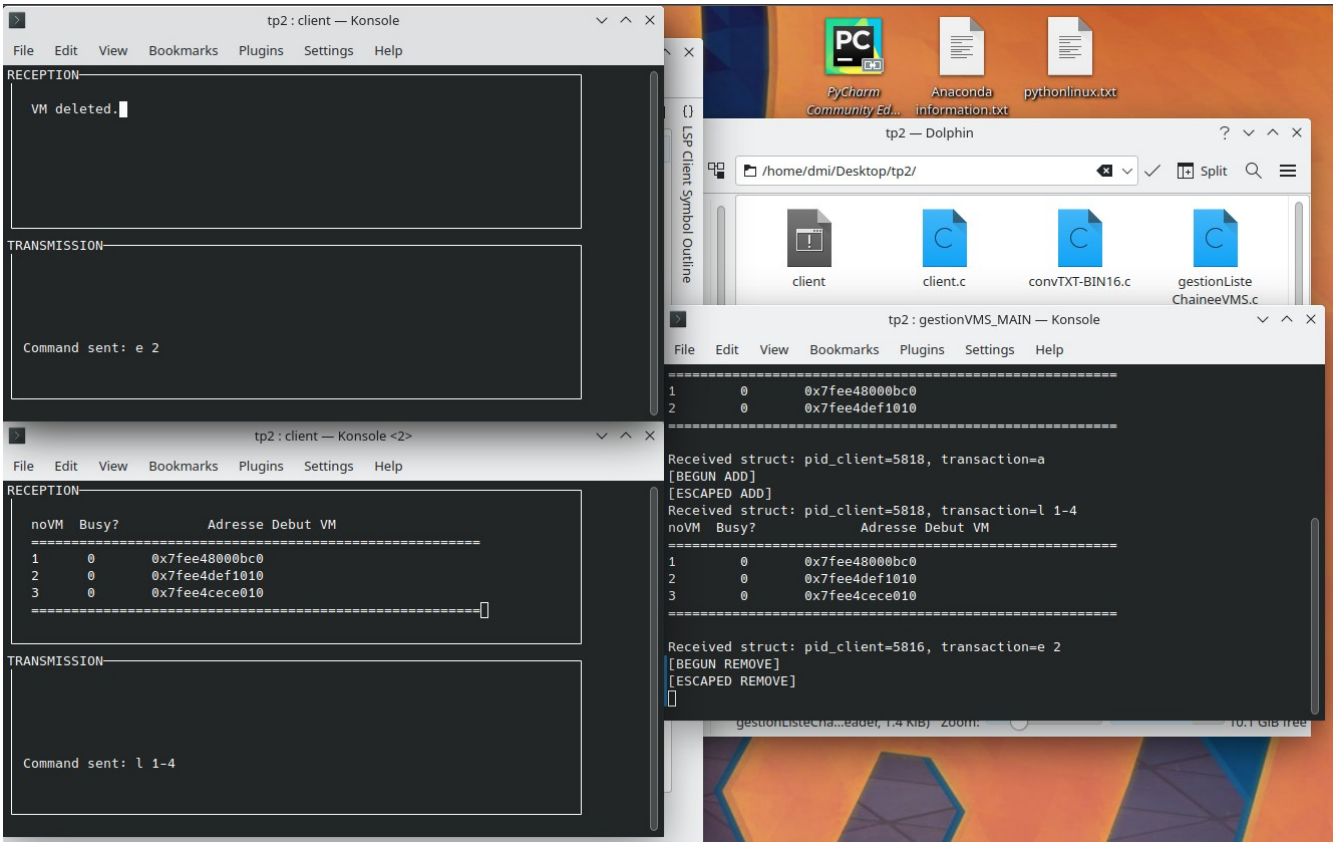

tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
TRANSMISSION
Command sent: l 1-4
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
dmi@dmi-virtual-machine:~/Desktop/tp2$ ./gestionVMS_MAIN
Received struct: pid_client=5816, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5818, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0

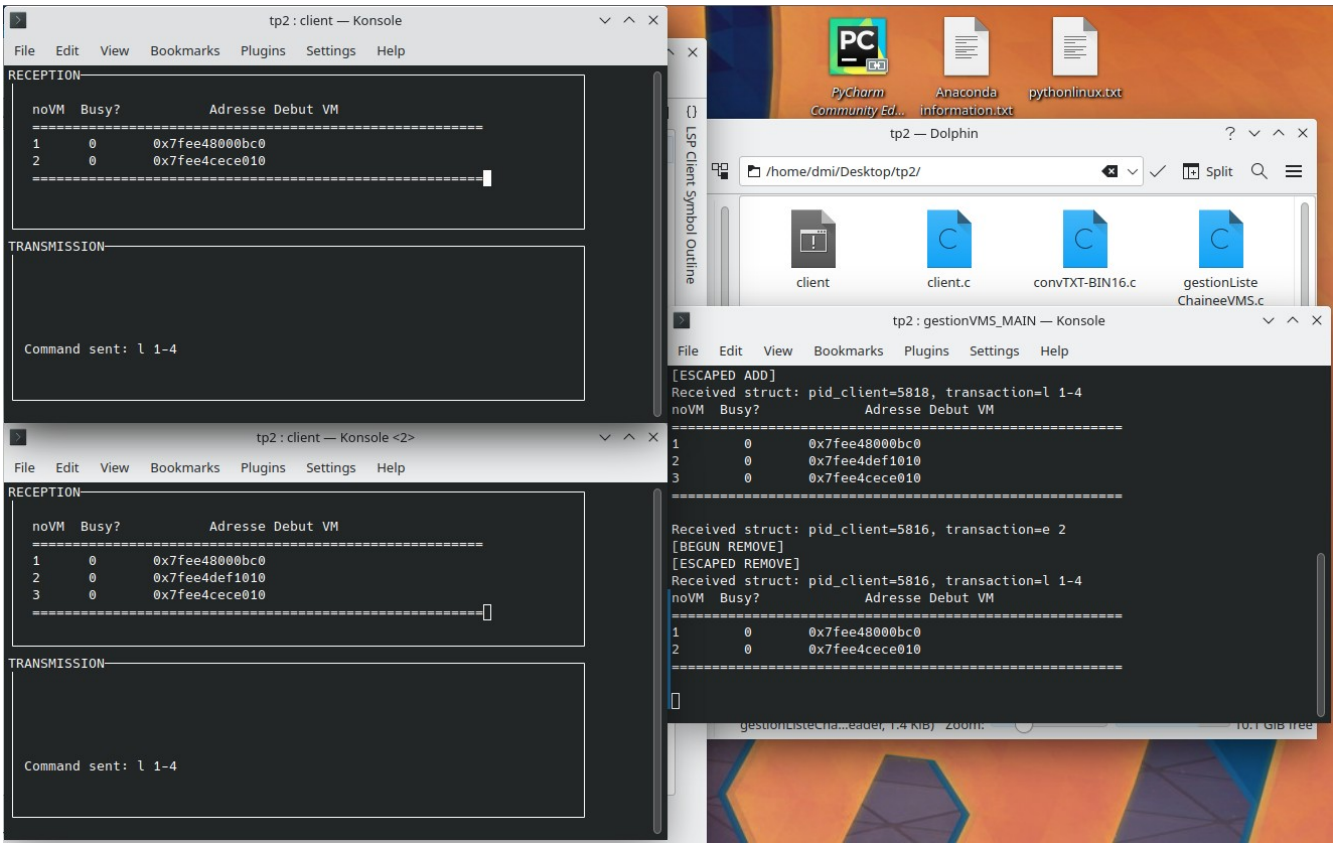

tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
dmi@dmi-virtual-machine:~/Desktop/tp2$ ./gestionVMS_MAIN
Received struct: pid_client=5816, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5818, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]

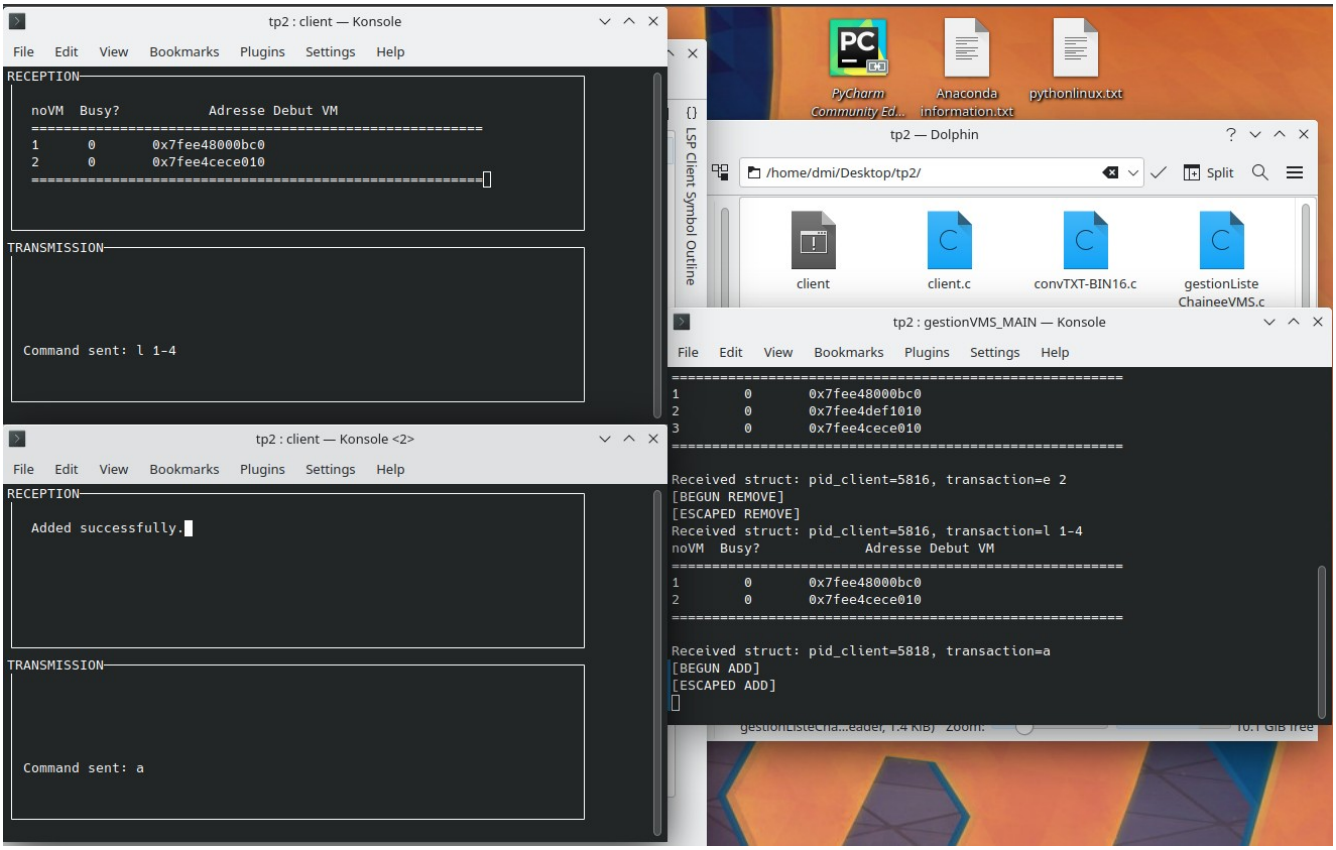

tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
TRANSMISSION
Command sent: l 1-4
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
Received struct: pid_client=5816, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5818, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5816, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010

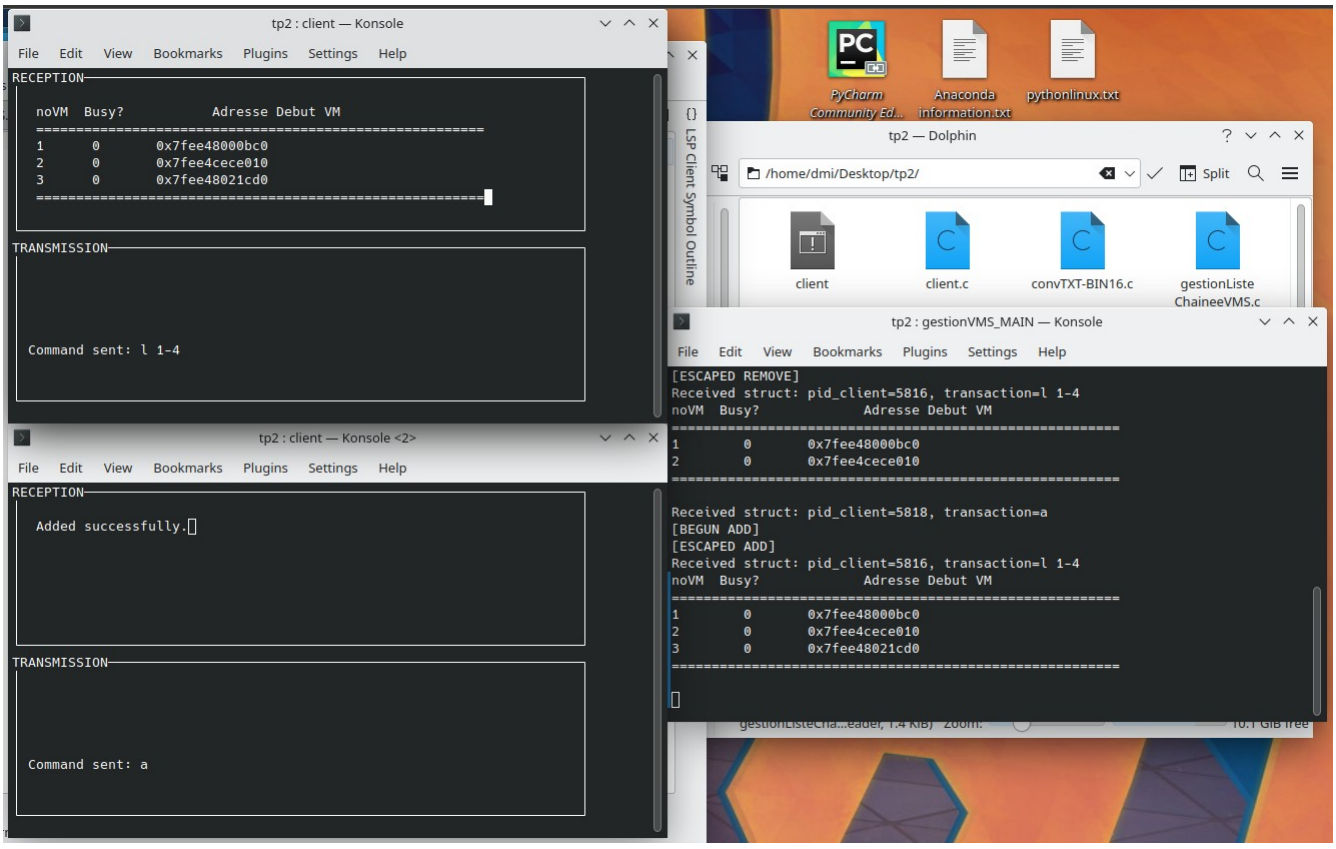

tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
TRANSMISSION
Command sent: l 1-4
tp2 : client — Konsole <2>
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
Received struct: pid_client=5818, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5816, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]

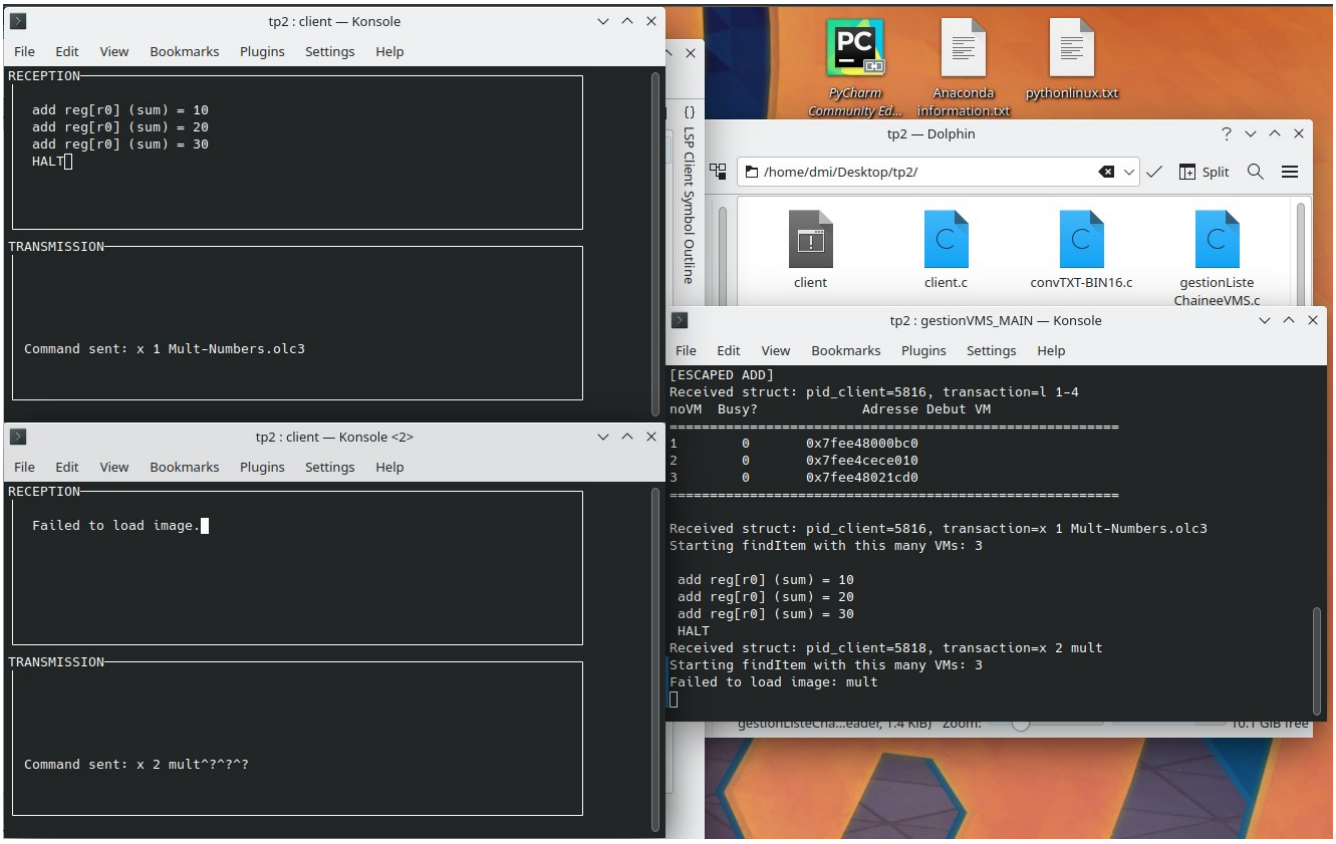

tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
TRANSMISSION
Command sent: l 1-4
tp2 : client — Konsole <2>
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
3 0 0x7fee4cece010
TRANSMISSION
Command sent: l 1-4
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
[ESCAPED ADD]
Received struct: pid_client=5816, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5818, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
3 0 0x7fee4cece010


tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
VM deleted.
TRANSMISSION
Command sent: e 2
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
3 0 0x7fee4cece010
TRANSMISSION
Command sent: l 1-4
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5818, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
3 0 0x7fee4cece010
Received struct: pid_client=5816, transaction=e 2
[BEGUN REMOVE]
[ESCAPED REMOVE]


tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4cece010
TRANSMISSION
Command sent: l 1-4
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
3 0 0x7fee4cece010
TRANSMISSION
Command sent: l 1-4
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
[ESCAPED ADD]
Received struct: pid_client=5818, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
3 0 0x7fee4cece010
Received struct: pid_client=5816, transaction=e 2
[BEGUN REMOVE]
[ESCAPED REMOVE]
Received struct: pid_client=5816, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4cece010


tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4cece010
TRANSMISSION
Command sent: l 1-4
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
1 0 0x7fee48000bc0
2 0 0x7fee4def1010
3 0 0x7fee4cece010
Received struct: pid_client=5816, transaction=e 2
[BEGUN REMOVE]
[ESCAPED REMOVE]
Received struct: pid_client=5816, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4cece010
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]


tp2 : client — Konsole
File Edit View Bookmarks Plugins Settings Help
RECEPTION
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4cece010
3 0 0x7fee48021cd0
TRANSMISSION
Command sent: l 1-4
tp2 : client — Konsole <2>
File Edit View Bookmarks Plugins Settings Help
RECEPTION
Added successfully.
TRANSMISSION
Command sent: a
PyCharm Community Ed...
Anaconda information.txt
pythonlinux.txt
tp2 — Dolphin
/home/dmi/Desktop/tp2/
Split
client
client.c
convTXT-BIN16.c
gestionListe ChaineeVMS.c
LSP Client Symbol Outline
tp2 : gestionVMS_MAIN — Konsole
File Edit View Bookmarks Plugins Settings Help
[ESCAPED REMOVE]
Received struct: pid_client=5816, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4cece010
Received struct: pid_client=5818, transaction=a
[BEGUN ADD]
[ESCAPED ADD]
Received struct: pid_client=5816, transaction=l 1-4
noVM Busy? Adresse Debut VM
1 0 0x7fee48000bc0
2 0 0x7fee4cece010
3 0 0x7fee48021cd0



J’ai écrit la mauvaise chose pour exécuter le script dans la deuxième fenêtre, et j’ai reçu l’erreur connexe.

Code :

client.c

```
 #include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdbool.h>
#include <pthread.h>
#include <curses.h>

#define FIFO_NAME "/tmp/FIFO_TRANSACTIONS"
#define BUFFER_SIZE PIPE_BUF
#define ONE_MEG (1024 * 1024)
#define PREV_INPUTS_LIST_SIZE 5

void readProcess(void *args);
void printCooldown(void *args);
char** splitString(char* input, int* numLines);
void addToPreviousStrings(char* input);
```

```
struct Info_FIFO_Transaction {
       int  pid_client;
       char transaction[1024];
};

WINDOW* contain_read_window_ptr;
WINDOW* read_window_ptr;
WINDOW* contain_write_window_ptr;
WINDOW* write_window_ptr;
int readWindowYsize;
int readWindowXsize;
int writeWindowYsize;
int writeWindowXsize;
bool recentlyPrinted;
int previousInputsIndex = 0;
int previousInputsListSize = PREV_INPUTS_LIST_SIZE;
char previousInputs[PREV_INPUTS_LIST_SIZE][50];

int main()
{
    int pipe_fd;
    int res;
    int open_mode = O_WRONLY;
    char buffer[BUFFER_SIZE + 1];
    char input[1024];
    bool closeCondition;
    pthread_t read_tid;
    int x_loop;
    int y_loop;
    char a_letter = 'a';

    readWindowYsize = 8;
    readWindowXsize = 70;
    writeWindowYsize = 8;
    writeWindowXsize = 70;

    initscr();
    noecho();

    //fillScreen();
    refresh();

    // Start reception window
    contain_read_window_ptr = newwin(readWindowYsize + 2, readWindowXsize + 2, 0, 0);
    read_window_ptr = newwin(readWindowYsize, readWindowXsize, 1, 1);
    scrollok(read_window_ptr, TRUE);
    box(contain_read_window_ptr, 0, 0);
    mvwprintw(contain_read_window_ptr, 0, 0, "%s", "RECEPTION");
    wrefresh(contain_read_window_ptr);
    wrefresh(read_window_ptr);

    // Start transmission window
    contain_write_window_ptr = newwin(writeWindowYsize + 2, writeWindowXsize + 2,
readWindowYsize + 2, 0);
    write_window_ptr = newwin(writeWindowYsize, writeWindowXsize, readWindowYsize + 3, 1);
    scrollok(write_window_ptr, TRUE);
    box(contain_write_window_ptr, 0, 0);
```

```
    mvwprintw(contain_write_window_ptr, 0, 0, "%s", "TRANSMISSION");
    wrefresh(contain_write_window_ptr);
    wrefresh(write_window_ptr);

    closeCondition = false;
    recentlyPrinted = false;

    pthread_create(&read_tid, NULL, readProcess, NULL);

    //printf("Process %d opening FIFO O_WRONLY\n", getpid());
    pipe_fd = open(FIFO_NAME, open_mode);
    //printf("Process %d result %d\n", getpid(), pipe_fd);

    if (pipe_fd != -1) {

        int ch;
        int y = writeWindowYsize - 2, x = 1;
        int index;
        char inputString[1024];
        wmove(write_window_ptr, y, x);
        touchwin(read_window_ptr);
        wrefresh(write_window_ptr);

        do {

            index = 0;
            while ((ch = getch()) != 27) { // Exit on ESC key
                // Check for Enter key
                if (ch == '\n') {
                    inputString[index] = '\0'; // Null-terminate the string

                    // if q, end program
                    if (inputString[0] == 'q' && inputString[1] == '\0') {

                        // delete windows
                        delwin(contain_read_window_ptr);
                        delwin(read_window_ptr);
                        delwin(contain_write_window_ptr);
                        delwin(write_window_ptr);

                        // send q as junk data (prevents dumb bug)
                        struct Info_FIFO_Transaction transac;
                        transac.pid_client = getpid();
                        snprintf(transac.transaction, sizeof(transac.transaction),
inputString);
                        memset(inputString, '\0', sizeof(inputString));
                        write(pipe_fd, &transac, sizeof(struct Info_FIFO_Transaction));

                        // unlink reception fifo
                        char reception_fifo[100];
                        sprintf(reception_fifo, "/tmp/FIFO%d", getpid());
                        unlink(reception_fifo);

                        // clear console
                        system("clear");

                        // exit
                        exit(EXIT_SUCCESS);
```

```
                }

                break; // Exit the loop on Enter
            } else {
                // Store the character in the string
                inputString[index] = ch;
                index++;

                // Display the character in the window
                mvwaddch(write_window_ptr, y, x, ch);
                x++;

                // Check for newline and reset the x position
                if (x > writeWindowXsize - 1) {
                    x = 1;
                    y++;
                }
            }

            wrefresh(write_window_ptr);
        }

        //clear write window
        int yClear, xClear;
        for (yClear = 1; yClear < writeWindowYsize; yClear++) {
            for (xClear = 1; xClear < writeWindowXsize; xClear++) {
                mvwprintw(write_window_ptr, yClear, xClear, "%s", " ");
            }
        }

        sleep(0.1);

        //reset input line
        x = 1;

        //write previous input upwards
        mvwprintw(write_window_ptr, y - 1, x, "Command sent: %s", inputString);

        wrefresh(write_window_ptr);

        struct Info_FIFO_Transaction transac;
               transac.pid_client = getpid();
        //transac.transaction = input;
        snprintf(transac.transaction, sizeof(transac.transaction), inputString);

        memset(inputString, '\0', sizeof(inputString));

        //res = write(pipe_fd, input, strlen(input) + 1);
        res = write(pipe_fd, &transac, sizeof(struct Info_FIFO_Transaction));
        if (res == -1) {
            //fprintf(stderr, "Write error on pipe\n");
            exit(EXIT_FAILURE);
        }

    } while (closeCondition == false);

    (void)close(pipe_fd);
}
```

```
    else {
        exit(EXIT_FAILURE);
    }

    //printf("Process %d finished\n", getpid());
    exit(EXIT_SUCCESS);
}

void readProcess(void *args) {

    char reception_fifo[100];
    int resRead, fd;
    bool closeConditionRead;
    char receivedContent[1024];
    pthread_t cool_tid;

    closeConditionRead = false;

    // Generate reception FIFO title
    sprintf(reception_fifo, "/tmp/FIFO%d", getpid());

    // Create FIFO if it does not exist
    if (access(reception_fifo, F_OK) == -1) {
        resRead = mkfifo(reception_fifo, 0777);
        if (resRead != 0) {
            //fprintf(stderr, "Could not create fifo %s\n", reception_fifo);
            exit(EXIT_FAILURE);
        }
    }

    do {

        // Open FIFO
        if ((fd = open(reception_fifo, O_RDONLY)) < 0) {
            //perror("Error opening FIFO for reading");
            exit(EXIT_FAILURE);
        }

        memset(receivedContent, '\0', sizeof(receivedContent));

        // Read data from the FIFO
        read(fd, receivedContent, sizeof(receivedContent) - 1);
        receivedContent[sizeof(receivedContent) - 1] = '\0';  // Ensure null-termination

        //clear reception window
        int yClear, xClear;
        for (yClear = 1; yClear < readWindowYsize; yClear++) {
            for (xClear = 1; xClear < readWindowXsize; xClear++) {
                //mvwaddch(read_window_ptr, yClear, xClear, ' ');
                mvwprintw(read_window_ptr, yClear, xClear, "%s", " ");
            }
        }

        sleep(0.1);

        wrefresh(read_window_ptr);

        //split message into lines, then write each one
```

```
        int numLines;
        char** lines = splitString(receivedContent, &numLines);
        // Print the split strings
        for (int i = 0; i < numLines; i++) {
            //printf("Line %d: %s\n", i + 1, lines[i]);
            mvwprintw(read_window_ptr, i + 1, 2, "%s", lines[i]);
        }

        // Display the read message
        //printf("%s\n", receivedContent);
        //mvwprintw(read_window_ptr, 1, 2, "%s", receivedContent);
        //box(read_window_ptr, 0, 0);
        //mvwprintw(read_window_ptr, 0, 0, "%s", "RECEPTION");
        touchwin(read_window_ptr);
        wrefresh(read_window_ptr);
        /*
        if (!recentlyPrinted) {
            printf("%s\n", receivedContent);
            recentlyPrinted = true;
            pthread_create(&cool_tid, NULL, printCooldown, NULL);
        }
        */

        // Close the FIFO file descriptor
        close(fd);

    } while (closeConditionRead == false);

    pthread_exit(1);
}

// permet d'éviter un bug étrange ou l'affichage d'un message recu se dupliquait
void printCooldown(void *args) {
    sleep(0.1);
    recentlyPrinted = false;
}

void fillScreen() {
    int x_loop;
    int y_loop;
    char a_letter = 'a';

    for (x_loop = 0; x_loop < COLS - 1; x_loop++) {
        for (y_loop = 0; y_loop < LINES - 1; y_loop++) {
            mvwaddch(stdscr, y_loop, x_loop, a_letter);
            a_letter++;
            if (a_letter > 'z') a_letter = 'a';
        }
    }

    refresh();
}

// Function to split a string into multiple strings using newline as delimiter
char** splitString(char* input, int* numLines) {

    *numLines = 0;
    char **result = NULL;
```

```
    int count = 0;
    int i;
    char *pch;

    // split
    pch = strtok (input,"\n");
    while (pch != NULL)
    {
        result = (char*)realloc(result, sizeof(char*)*(count+1));
        result[count] = (char*)malloc(strlen(pch)+1);
        strcpy(result[count], pch);
        count++;
        (*numLines)++;
        pch = strtok (NULL, "\n");
    }

    return result;
}
```

gestionListeChaineeVMS.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdbool.h>

#include <stdint.h>
#include <signal.h>
/* unix */
#include <unistd.h>
#include <fcntl.h>

#include <sys/time.h>
#include <sys/types.h>
#include <sys/termios.h>
#include <sys/mman.h>

struct infoVM{
       int          noVM;
       unsigned char      busy;
       uint16_t *   ptrDebutVM;
       uint16_t     offsetDebutCode; // region memoire ReadOnly
       uint16_t     offsetFinCode;
       };

struct noeudVM{
       struct infoVM      VM;
       struct noeudVM            *suivant;
       sem_t semVM;
       };

struct listFuncThreadArgs {
    int nstart;
```

```
    int nend;
       char client_fifo[100];
       };

struct exeFuncThreadArgs {
    int noVM;
       char nomfich[100];
       char client_fifo[100];
       };

struct removeFuncThreadArgs {
    int _noVM;
       char client_fifo[100];
       };

struct addFuncThreadArgs {
       char client_fifo[100];
       };

struct Info_FIFO_Transaction {
       int  pid_client;
       char transaction[1024];
       };

void cls(void);
void error(const int exitcode, const char * message);

struct noeudVM * findItem(const int no);
struct noeudVM * findPrev(const int no);

void addItem(struct addFuncThreadArgs* structParam);
void removeItem(struct removeFuncThreadArgs* structParam);
void listItems(struct listFuncThreadArgs* structParam);
void saveItems(const char* sourcefname);
void executeFile(struct exeFuncThreadArgs* structParam);

void* readTrans();
```

gestionVMS_MAIN.c

```
//#############################################################
//#
//# Titre :  UTILITAIRES (MAIN) TP1 LINUX Automne 21
//#                       SIF-1015 - Système d'exploitation
//#                       Université du Québec à Trois-Rivières
//#
//# Auteur :        Francois Meunier
//#   Date :        Septembre 2022
//#
//# Langage :       ANSI C on LINUX
//#
//#######################################

#include "gestionListeChaineeVMS.h"
#include "gestionVMS.h"

//Pointeur de tête de liste
```

```
struct noeud* head;
//Pointeur de queue de liste pour ajout rapide
struct noeud* queue;
// nombre de VM actives
int nbVM;
// sémaphores
sem_t semH, semQ, semNbVM, semConsole, semAdd;


int main(){

	//Initialisation des pointeurs
	head = NULL;
	queue = NULL;
	nbVM = 0;

	// initialisation des sémaphores
	sem_init(&semH, 0, 1);
	sem_init(&semQ, 0, 1);
	sem_init(&semConsole, 0, 1);
	sem_init(&semNbVM, 0, 1);
	sem_init(&semAdd, 0, 1);

	//"Nettoyage" de la fenêtre console
	//cls();

	readTrans();


	//Fin du programme
	exit( 0);
}
```

gestionListeChaineeVMS.c

```
//#########################################################
//#
//# Titre :  Utilitaires Liste Chainee et CVS LINUX Automne 21
//#                       SIF-1015 - Système d'exploitation
//#                       Université du Québec à Trois-Rivières
//#
//# Auteur :        Francois Meunier
//#   Date :        Septembre 2022
//#
//# Langage :       ANSI C on LINUX
//#
//#########################################

#include "gestionListeChaineeVMS.h"
#include "gestionVMS.h"

//Pointeur de tête de liste
extern struct noeudVM* head;
//Pointeur de queue de liste pour ajout rapide
extern struct noeudVM* queue;
```

```
// nombre de VM actives
extern int nbVM;
//sémaphores
extern sem_t semH;
extern sem_t semQ;
extern sem_t semConsole;
extern sem_t semNbVM;
extern sem_t semAdd;

//#######################################
//# Recherche un item dans la liste chaînée
//# ENTREE: Numéro de la ligne
//# RETOUR:  Un pointeur vers l'item recherché
//#          Retourne NULL dans le cas où l'item
//#                est introuvable
//#
struct noeudVM * findItem(const int no){

	printf("Starting findItem with this many VMs: %d\n", nbVM);

	sem_wait(&semH);
	sem_wait(&semQ);

	//La liste est vide
	if ((head==NULL)&&(queue==NULL)) {
		sem_post(&semQ);
		sem_post(&semH);
		printf("[EMPTY LIST FOUND BRUH]\n");
		return NULL;
	}

	//Pointeur de navigation
	//sem_wait(&(head->semVM)); // verrouille noeud de tete
	struct noeudVM * ptr = head;
	sem_post(&semQ);
	sem_post(&semH);

	if(ptr->VM.noVM==no) // premier noeudVM
		return ptr;

	//Tant qu'un item suivant existe
	while (ptr->suivant!=NULL){

		//Déplacement du pointeur de navigation
		struct noeudVM* optr = ptr;
		ptr=ptr->suivant;
		//sem_post(&(optr->semVM));

		//Est-ce l'item recherché?
		if (ptr->VM.noVM==no){
			return ptr;
		}

		/*
		if (ptr->suivant!=NULL){
			sem_wait(&(ptr->suivant->semVM));
		}
		else { // ptr->suivant==NULL no invalide
```

```
                sem_post(&(ptr->semVM)); // deverrouille dernier noeud verrouille
            }
            */

      }
      //On retourne un pointeur NULL
      printf("End of findItem reached.\n");
      return NULL;
}

//#######################################
//#
//# Recherche le PRÉDÉCESSEUR d'un item dans la liste chaînée
//# ENTREE: Numéro de la ligne a supprimer
//# RETOUR:  Le pointeur vers le prédécesseur est retourné
//#          Retourne NULL dans le cas où l'item est introuvable
//#
struct noeudVM * findPrev(const int no){

      sem_wait(&semH);
      sem_wait(&semQ);

      //La liste est vide
      if ((head==NULL)&&(queue==NULL)) {
            sem_post(&semQ);
            sem_post(&semH);
            return NULL;
      }

      //sem_wait(&(head->semVM)); // verrouille noeud de tete
      struct noeudVM * ptr = head;
      sem_post(&semQ);
      sem_post(&semH);

      //Tant qu'un item suivant existe
      while (ptr->suivant!=NULL){

            //Est-ce le prédécesseur de l'item recherché?
            if (ptr->suivant->VM.noVM==no){
                  //sem_post(&(ptr->suivant->semVM));
                  //On retourne un pointeur sur l'item précédent
                  return ptr;
            }

            //Déplacement du pointeur de navigation
            struct noeudVM* optr = ptr;
            ptr=ptr->suivant;
            //sem_post(&(optr->semVM));
      }

      //On retourne un pointeur NULL
      return NULL;
}

//#####################################################
//# Ajoute un item a la fin de la liste chaînée de VM
//# ENTREE:
//#   RETOUR:
```

```
void addItem(struct addFuncThreadArgs* structParam){

	char my_client_fifo[100];
	strcpy(my_client_fifo,(const char*)structParam->client_fifo);
	//printf("Test: %s\n", my_client_fifo);

	sem_wait(&semAdd);

	printf("[BEGUN ADD]\n");

	//Création de l'enregistrement en mémoire
	struct noeudVM* ni = (struct noeudVM*)malloc(sizeof(struct noeudVM));
//printf("\n noVM=%d busy=%d adr ni=%p", ni->VM.noVM, ni->VM.busy, ni);
//printf("\n noVM=%d busy=%d adrQ deb=%p", ni->VM.noVM, ni->VM.busy,queue);

	//Affectation des valeurs des champs
	sem_wait(&semNbVM);
	ni->VM.noVM  = ++nbVM;
	sem_post(&semNbVM);
	//printf("\n noVM=%d", ni->VM.noVM);
	ni->VM.busy  = 0;
	//printf("\n busy=%d", ni->VM.busy);
	ni->VM.ptrDebutVM   = (unsigned short*)malloc(sizeof(unsigned short)*65536);
//printf("\n noVM=%d busy=%d adrptr VM=%p", ni->VM.noVM, ni->VM.busy, ni->VM.ptrDebutVM);
//printf("\n noVM=%d busy=%d adrQ=%p", ni->VM.noVM, ni->VM.busy, queue);

	sem_init(&(ni->semVM), 0, 1);

	sem_wait(&semH);
	sem_wait(&semQ);

	if ((head == NULL) && (queue == NULL)){//liste vide
	  ni->suivant= NULL;
	  queue = head = ni;

	  sem_post(&semQ);
	  sem_post(&semH);

	  printf("[ESCAPED ADD]\n");
	  sem_post(&semAdd);

	  sendMsgToClient(my_client_fifo, "Added successfully.");

	  //return NULL;
	  pthread_exit(1);
	}

	sem_wait(&(queue->semVM));

	struct noeudVM* tptr = queue;
	ni->suivant = NULL;
	queue = ni;
//printf("\n noVM=%d busy=%d adrQ=%p", ni->VM.noVM, ni->VM.busy, queue);
	tptr->suivant = ni;
//printf("\n noVM=%d busy=%d adr Queue=%p", ni->VM.noVM, ni->VM.busy,queue);

	sem_post(&(tptr->semVM));
```

```
	sem_post(&semQ);
	sem_post(&semH);

	printf("[ESCAPED ADD]\n");
	sem_post(&semAdd);

	sendMsgToClient(my_client_fifo, "Added successfully.");

	//return NULL;
	pthread_exit(1);
}

//#######################################
//# Retire un item de la liste chaînée
//# ENTREE: noVM: numéro du noeud a retirer
void removeItem(struct removeFuncThreadArgs* structParam){

	int noVM = structParam->_noVM;
	char my_client_fifo[100];
	strcpy(my_client_fifo,(const char*)structParam->client_fifo);

	struct noeudVM * ptr;
	struct noeudVM * tptr;
	struct noeudVM * optr;

	sem_wait(&semH);
	sem_wait(&semQ);

	printf("[BEGUN REMOVE]\n");

	//Vérification sommaire (noVM>0 et liste non vide)
	if ((noVM<1)||((head==NULL)&&(queue==NULL))) {
		sem_post(&semQ);
		sem_post(&semH);
		printf("[ESCAPED REMOVE (LIST EMPTY)]\n");
		//sem_post(&semAdd);

		sendMsgToClient(my_client_fifo, "List is empty.");

		//return NULL;
		pthread_exit(0);
	}

	//Pointeur de recherche
	if(noVM==1){
		ptr = head; // suppression du premier element de la liste
	}
	else{
		sem_post(&semQ);
		sem_post(&semH);

		sem_wait(&semAdd);
		ptr = findPrev(noVM);
		sem_post(&semAdd);

		sem_wait(&semH);
		sem_wait(&semQ);
	}
```

```
bool itemFound = false;
//L'item a été trouvé
if (ptr!=NULL){

	itemFound = true;
	//sem_post(&(ptr->semVM));

	sem_wait(&semNbVM);
	nbVM--;
	sem_post(&semNbVM);

	// Memorisation du pointeur de l'item en cours de suppression
	// Ajustement des pointeurs
	if((head == ptr) && (noVM==1)) // suppression de l'element de tete
	{
		if(head==queue) // un seul element dans la liste
		{
			sem_wait(&(ptr->semVM));
			sem_post(&(ptr->semVM));
			sem_destroy(&(ptr->semVM));
			free(ptr->VM.ptrDebutVM);
			free(ptr);
			queue = head = NULL;
			sem_post(&semQ);
			sem_post(&semH);
			printf("[ESCAPED REMOVE]\n");
			//sem_post(&semAdd);

			sendMsgToClient(my_client_fifo, "VM deleted.");

			//return NULL;
			pthread_exit(1);
		}
		tptr = ptr->suivant;
		head = tptr;
		sem_wait(&(ptr->semVM));
		sem_post(&(ptr->semVM));
		sem_destroy(&(ptr->semVM));
		free(ptr->VM.ptrDebutVM);
		free(ptr);
	}
	else if (queue==ptr->suivant) // suppression de l'element de queue
	{
		queue=ptr;
		sem_wait(&(ptr->suivant->semVM));
		sem_post(&(ptr->suivant->semVM));
		sem_destroy(&(ptr->suivant->semVM));
		free(ptr->suivant->VM.ptrDebutVM);
		free(ptr->suivant);
		ptr->suivant=NULL;
		sem_post(&semQ);
		sem_post(&semH);
		printf("[ESCAPED REMOVE]\n");
		//sem_post(&semAdd);

		sendMsgToClient(my_client_fifo, "VM deleted.");
```

```
                //return NULL;
                pthread_exit(1);
            }
            else // suppression d'un element dans la liste
            {
                optr = ptr->suivant;
                ptr->suivant = ptr->suivant->suivant;
                tptr = ptr->suivant;
                sem_wait(&(optr->semVM));
                sem_post(&(optr->semVM));
                sem_destroy(&(optr->semVM));
                free(optr->VM.ptrDebutVM);
                free(optr);
            }


            while (tptr!=NULL){ // ajustement des numeros de VM

            //Est-ce le prédécesseur de l'item recherché?
                tptr->VM.noVM--;
                //On retourne un pointeur sur l'item précédent

            //Déplacement du pointeur de navigation
                tptr=tptr->suivant;
            }
        }

        sem_post(&semQ);
        sem_post(&semH);

        if (itemFound) {
            printf("[ESCAPED REMOVE]\n");
            sendMsgToClient(my_client_fifo, "VM deleted.");
        }
        else {
            printf("[ESCAPED REMOVE (ITEM NOT FOUND)]\n");
            sendMsgToClient(my_client_fifo, "VM not found.");
        }

        //sem_post(&semAdd);

        //return NULL;
        pthread_exit(1);
}

//#######################################
//#
//# Affiche les items dont le numéro séquentiel est compris dans une plage
//#
void listItems(struct listFuncThreadArgs* structParam){

        const int start = structParam->nstart;
        const int end = structParam->nend;
        char my_client_fifo[100];
        char message[1024] = {'\0'};
        strcpy(my_client_fifo,(const char*)structParam->client_fifo);
```

```
	sem_wait(&semConsole);

	//Affichage des entêtes de colonnes
	printf("noVM  Busy?       Adresse Debut VM                        \n");
	printf("=========================================================\n");
	strcat(message, "noVM  Busy?           Adresse Debut VM                          \n");
	strcat(message, "=========================================================\n");

	sem_wait(&semH);

	struct noeudVM * ptr = head;                 //premier element
	//sem_wait(&(ptr->semVM));

	sem_post(&semH);

	while (ptr!=NULL){

		//L'item a un numéro séquentiel dans l'interval défini
		if ((ptr->VM.noVM>=start)&&(ptr->VM.noVM<=end)){
			printf("%d \t %d \t %p\n",
				ptr->VM.noVM,
				ptr->VM.busy, ptr->VM.ptrDebutVM);
			char line[1024];
			sprintf(line, "%d \t %d \t %p\n", ptr->VM.noVM, ptr->VM.busy, ptr-
>VM.ptrDebutVM);
			strcat(message, line);
		}

		if (ptr->VM.noVM>end){
			//L'ensemble des items potentiels sont maintenant passés
			//Déplacement immédiatement à la FIN de la liste
			//Notez que le pointeur optr est toujours valide
			ptr=NULL;
			//sem_post(&(ptr->semVM));
		}
		else{
			sem_wait(&semAdd);

			struct noeudVM* optr = ptr;
			if (ptr->suivant!=NULL) {
				//sem_wait(&(ptr->suivant->semVM));
			}
			ptr = ptr->suivant;
			//sem_post(&(optr->semVM));

			sem_post(&semAdd);
		}

	}

	//Affichage des pieds de colonnes
	printf("=========================================================\n\n");
	strcat(message, "=========================================================");
	sendMsgToClient(my_client_fifo, message);

	sem_post(&semConsole);

	//printf("[ESCAPED LISTING]");
```

```
	//return NULL;
	pthread_exit(1);
}

void sendMsgToClient(char client_fifo[100], char message[1024]) {

	int pipe_fd;
    int res;

	if ((pipe_fd = open(client_fifo, O_WRONLY)) < 0) {
		perror("Error opening FIFO for reading");
	}
	else {
		res = write(pipe_fd, message, strlen(message));
		if (res == -1) {
			fprintf(stderr, "Write error on pipe\n");
		}
		(void)close(pipe_fd);
	}

}
```

gestionVMS.c

```
//#########################################################
//#
//# Titre :  Utilitaires CVS LINUX Automne 21
//#                       SIF-1015 - Système d'exploitation
//#                       Université du Québec à Trois-Rivières
//#
//# Auteur :        Francois Meunier
//#   Date :        Septembre 2022
//#
//# Langage :       ANSI C on LINUX
//#
//#######################################

#include "gestionListeChaineeVMS.h"
#include "gestionVMS.h"
#include <pthread.h>

#define FIFO_NAME "/tmp/FIFO_TRANSACTIONS"

//Pointeur de tête de liste
extern struct noeud* head;
//Pointeur de queue de liste pour ajout rapide
extern struct noeud* queue;

// nombre de VM actives
extern int nbVM;

//sémaphores
extern sem_t semConsole;

//#######################################
//#
//# Affiche une série de retour de ligne pour "nettoyer" la console
```

```
//#
void cls(void){
       printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
       }

//#######################################
//#
//# Affiche un messgae et quitte le programme
//#
void error(const int exitcode, const char * message){
       printf("\n------------------------\n%s\n",message);
       exit(exitcode);
       }

/* Sign Extend */
uint16_t sign_extend(uint16_t x, int bit_count)
{
    if ((x >> (bit_count - 1)) & 1) {
        x |= (0xFFFF << bit_count);
    }
    return x;
}

/* Swap */
uint16_t swap16(uint16_t x)
{
    return (x << 8) | (x >> 8);
}

/* Update Flags */
void update_flags(uint16_t reg[R_COUNT], uint16_t r)
{
    if (reg[r] == 0)
    {
        reg[R_COND] = FL_ZRO;
    }
    else if (reg[r] >> 15) /* a 1 in the left-most bit indicates negative */
    {
        reg[R_COND] = FL_NEG;
    }
    else
    {
        reg[R_COND] = FL_POS;
    }
}


/* Validation of an adress generated by a program  */
int validAdress(uint16_t * memory,uint16_t  offset, struct noeudVM * ptr, char accessType)
{

       if(((memory+offset) > (memory+65535)) || ((memory) > (memory+offset)) )
       {
              printf("\n Adresse invalide");
              return 0;
       }
    if (accessType == 'W') { // validate if in code region
```

```
                    if(((memory+offset) > (memory+ptr->VM.offsetDebutCode)) &&
((memory+ptr->VM.offsetFinCode) > (memory+offset)))
                    {
                           printf("\n Adresse invalide en Write");
                           return 0;
                    }
      }
      return 1;
}

/* Read Image File */
int read_image_file(uint16_t * memory, char* image_path,uint16_t * origin,  struct noeudVM
* ptr)
{
       char fich[200];
       strcpy(fich,image_path);
       FILE* file = fopen(fich, "rb");

    if (!file) { return 0; }
    /* the origin tells us where in memory to place the image */
      *origin=0x3000;

    /* we know the maximum file size so we only need one fread */
    uint16_t max_read = UINT16_MAX - *origin;
    uint16_t* p = memory + *origin;
    ptr->VM.offsetDebutCode = *origin;
    size_t read = fread(p, sizeof(uint16_t), max_read, file);
    ptr->VM.offsetFinCode = *origin+read-1;
    /* swap to little endian ???? */
    while (read-- > 0)
    {
    // printf("\n p * BIG = %x",*p);
       // *p = swap16(*p);
             // printf("\n p * LITTLE = %x",*p);
        ++p;
    }
    return 1;
}


/* Check Key */
uint16_t check_key()
{
    fd_set readfds;
    FD_ZERO(&readfds);
    FD_SET(STDIN_FILENO, &readfds);

    struct timeval timeout;
    timeout.tv_sec = 0;
    timeout.tv_usec = 0;
    return select(1, &readfds, NULL, NULL, &timeout) != 0;
}

/* Memory Access */
void mem_write(uint16_t * memory, uint16_t address, uint16_t val)
{
    memory[address] = val;
}
```

```
uint16_t mem_read( uint16_t * memory, uint16_t address)
{
    if (address == MR_KBSR)
    {
        if (check_key())
        {
            memory[MR_KBSR] = (1 << 15);
            memory[MR_KBDR] = getchar();
        }
        else
        {
            memory[MR_KBSR] = 0;
        }
    }
    return memory[address];
}

/* Input Buffering */
struct termios original_tio;

void disable_input_buffering()
{
    tcgetattr(STDIN_FILENO, &original_tio);
    struct termios new_tio = original_tio;
    new_tio.c_lflag &= ~ICANON & ~ECHO;
    tcsetattr(STDIN_FILENO, TCSANOW, &new_tio);
}

void restore_input_buffering()
{
    tcsetattr(STDIN_FILENO, TCSANOW, &original_tio);
}

/* Handle Interrupt */
void handle_interrupt(int signal)
{
    restore_input_buffering();
    printf("\n");
    exit(-2);
}



//#######################################
//#
//# Execute le fichier de code .obj
//#
void executeFile(struct exeFuncThreadArgs* structParam){

    /*
    struct exeFuncThreadArgs* myExeStruct = (struct exeFuncThreadArgs*) structParam;
      int noVM = myExeStruct->noVM;
      char* sourcefname = myExeStruct->nomfich;
    */

    char sourcefname[100];
      int noVM;
```

```
      noVM = structParam->noVM;
      strcpy(sourcefname,(const char*)structParam->nomfich);
    char my_client_fifo[100];
      strcpy(my_client_fifo,(const char*)structParam->client_fifo);
      free(structParam);

/* Memory Storage */
/* 65536 locations */
      uint16_t *  memory;
      uint16_t origin;
      uint16_t PC_START;

/* Register Storage */
      uint16_t reg[R_COUNT];

    sem_wait(&semConsole);

    char message[1024] = {'\0'};
    char line[1024];

    struct noeudVM * ptr = findItem(noVM);

    /*
    if (ptr == NULL)
    {
        //sem_wait(&semConsole);
        printf("Virtual Machine unavailable\n");
        //sem_post(&semConsole);
        //return NULL; //bad outcome
        //sem_post(&(ptr->semVM));
        sem_post(&semConsole);
        //printf("[ESCAPED EXEC]");
        pthread_exit(0);
    }
    */

    if (ptr == NULL)
    {
        printf("Virtual Machine unavailable, trying again shortly...\n");

        sleep(0.5);

        ptr = findItem(noVM);
        if (ptr == NULL) {
            printf("Virtual Machine not found.\n");
            sem_post(&semConsole);

            sendMsgToClient(my_client_fifo, "VM not found.\n");

            //printf("[ESCAPED EXEC]");
            pthread_exit(0);
        }
    }

    sem_wait(&(ptr->semVM));

      memory = ptr->VM.ptrDebutVM;
```

```
    if (!read_image_file(memory, sourcefname, &origin, ptr))
    {
        //sem_wait(&semConsole);
        printf("Failed to load image: %s\n", sourcefname);
        //return NULL; //bad outcome
        sem_post(&(ptr->semVM));
        sem_post(&semConsole);

        sendMsgToClient(my_client_fifo, "Failed to load image.\n");

        //printf("[ESCAPED EXEC]");
        pthread_exit(0);
    }

    while(ptr->VM.busy != 0){ // wait for the VM
    }
      // Acquiring access to the VM
    ptr->VM.busy = 1;

    /* Setup */
    signal(SIGINT, handle_interrupt);
    disable_input_buffering();

    /* set the PC to starting position */
    /* at  ptr->VM.ptrDebutVM + 0x3000 is the default  */
    //enum { PC_START = origin };
    PC_START = origin;
    reg[R_PC] = PC_START;
    uint16_t instr;
    uint16_t op;

    int running = 1;
    while (running)
    {
        /* FETCH */
             if(validAdress(memory, reg[R_PC], ptr, 'R'))
             {
                    instr = mem_read(memory, reg[R_PC]++);
// printf("\n instr = %x", instr);

             }
      else //invalid adress
      {
                    running = 0;
                    printf("\n Program abort memory problem");
            strcat(message, "Program abort memory problem\n");
                    continue;
             }
             op = instr >> 12;
// printf("\n exe op = %x", op);

        switch (op)
        {
            case OP_ADD:
                /* ADD */
                {
                    /* destination register (DR) */
                    uint16_t r0 = (instr >> 9) & 0x7;
```

```
                    /* first operand (SR1) */
                    uint16_t r1 = (instr >> 6) & 0x7;
                    /* whether we are in immediate mode */
                    uint16_t imm_flag = (instr >> 5) & 0x1;

                    if (imm_flag)
                    {
                        uint16_t imm5 = sign_extend(instr & 0x1F, 5);
                        reg[r0] = reg[r1] + imm5;
                    }
                    else
                    {
                        uint16_t r2 = instr & 0x7;
                        reg[r0] = reg[r1] + reg[r2];

                   sprintf(line, "add reg[r0] (sum) = %d\n", reg[r0]);
                   strcat(message, line);
printf("\n add reg[r0] (sum) = %d", reg[r0]);
//printf("\t add reg[r1] (sum avant) = %d", reg[r1]);
//printf("\t add reg[r2] (valeur ajoutee) = %d", reg[r2]);
                    }

                    update_flags(reg, r0);
                }

                break;
            case OP_AND:
                /* AND */
                {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t r1 = (instr >> 6) & 0x7;
                    uint16_t imm_flag = (instr >> 5) & 0x1;

                    if (imm_flag)
                    {
                        uint16_t imm5 = sign_extend(instr & 0x1F, 5);
                        reg[r0] = reg[r1] & imm5;
                    }
                    else
                    {
                        uint16_t r2 = instr & 0x7;
                        reg[r0] = reg[r1] & reg[r2];
                    }
                    update_flags(reg, r0);
                }

                break;
            case OP_NOT:
                /* NOT */
                {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t r1 = (instr >> 6) & 0x7;

                    reg[r0] = ~reg[r1];
                    update_flags(reg, r0);
                }

                break;
```

```
            case OP_BR:
                /* BR */
                {
                    uint16_t pc_offset = sign_extend(instr & 0x1FF, 9);
                    uint16_t cond_flag = (instr >> 9) & 0x7;
                    if (cond_flag & reg[R_COND])
                    {
                        reg[R_PC] += pc_offset;
                    }
                }

                break;
            case OP_JMP:
                /* JMP */
                {
                    /* Also handles RET */
                    uint16_t r1 = (instr >> 6) & 0x7;
                    reg[R_PC] = reg[r1];
                }

                break;
            case OP_JSR:
                /* JSR */
                {
                    uint16_t long_flag = (instr >> 11) & 1;
                    reg[R_R7] = reg[R_PC];
                    if (long_flag)
                    {
                        uint16_t long_pc_offset = sign_extend(instr & 0x7FF, 11);
                        reg[R_PC] += long_pc_offset;  /* JSR */
                    }
                    else
                    {
                        uint16_t r1 = (instr >> 6) & 0x7;
                        reg[R_PC] = reg[r1]; /* JSRR */
                    }
                    break;
                }

                break;
            case OP_LD:
                /* LD */
           {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t pc_offset = sign_extend(instr & 0x1FF, 9);
                                if(validAdress(memory, reg[R_PC]+ pc_offset, ptr, 'R'))
                                {
                                        reg[r0] = mem_read(memory, reg[R_PC] +
pc_offset); // valider adresse afficher message erreur mem et running = 0
                                        update_flags(reg, r0);
                                }
                                else //invalid adress
                                {
                                        running = 0;
                                        printf("\n Program abort memory out of bound");
                        sprintf(line, "Program abort memory out of bound\n");
                        strcat(message, line);
                                }
```

```
            }

                break;
             case OP_LDI:
                /* LDI */
             {
                    /* destination register (DR) */
                    uint16_t r0 = (instr >> 9) & 0x7;
                    /* PCoffset 9*/
                    uint16_t pc_offset = sign_extend(instr & 0x1FF, 9);
                    /* add pc_offset to the current PC, look at that memory location to
get the final address */
                                    if(validAdress(memory, reg[R_PC]+ pc_offset, ptr, 'R'))
                                    {
                                           reg[r0] = mem_read(memory, mem_read(memory,
reg[R_PC] + pc_offset)); // valider adresse afficher message erreur mem et running = 0
                                           update_flags(reg, r0);
                                    }
                                    else //invalid adress
                                    {
                                           running = 0;
                                           printf("\n Program abort memory out of bound");
                            sprintf(line, "Program abort memory out of bound\n");
                            strcat(message, line);
                                    }
             }

                break;
             case OP_LDR:
                /* LDR */
             {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t r1 = (instr >> 6) & 0x7;
                    uint16_t offset = sign_extend(instr & 0x3F, 6);
                                    if(validAdress(memory, reg[r1]+ offset, ptr, 'R'))
                                    {
                                           reg[r0] = mem_read(memory, reg[r1] + offset); //
valider adresse afficher message erreur mem et running = 0
                                           update_flags(reg, r0);
                                    }
                                    else //invalid adress
                                    {
                                           running = 0;
                                           printf("\n Program abort memory out of bound");
                            sprintf(line, "Program abort memory out of bound\n");
                            strcat(message, line);
                                    }
            }

                break;
             case OP_LEA:
                /* LEA */
                {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t pc_offset = sign_extend(instr & 0x1FF, 9);
                    reg[r0] = reg[R_PC] + pc_offset;
                    update_flags(reg, r0);
                }
```

```
                break;
            case OP_ST:
                /* ST */
            {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t pc_offset = sign_extend(instr & 0x1FF, 9);
                                if(validAdress(memory, reg[R_PC]+ pc_offset, ptr, 'W'))
                                {
                                        mem_write(memory, reg[R_PC] + pc_offset,
reg[r0]); // valider adresse afficher message erreur mem et running = 0
                                }
                                else //invalid adress
                                {
                                        running = 0;
                                        printf("\n Program abort memory out of bound or
access violation");
                            sprintf(line, "Program abort memory out of bound or access
violation\n");
                            strcat(message, line);
                                }
                              }

                break;
            case OP_STI:
                /* STI */
            {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t pc_offset = sign_extend(instr & 0x1FF, 9);
                                if(validAdress(memory, reg[R_PC] + pc_offset, ptr, 'R'))
                                {
                                        if(validAdress(memory, mem_read(memory, reg[R_PC] +
pc_offset), ptr, 'W'))
                                        {
                                                mem_write(memory, mem_read(memory, reg[R_PC]
+ pc_offset), reg[r0]); // valider adresse afficher message erreur mem et running = 0
                                        }
                                        else //invalid adress
                                        {
                                                running = 0;
                                                printf("\n Program abort memory out of bound
or access violation");
                                sprintf(line, "Program abort memory out of bound or access
violation\n");
                                strcat(message, line);
                                        }
                                }
                                else //invalid adress
                                {
                                        running = 0;
                                        printf("\n Program abort memory out of bound");
                            sprintf(line, "Program abort memory out of bound\n");
                            strcat(message, line);
                                }
                              }

                break;
            case OP_STR:
```

```
                /* STR */
            {
                    uint16_t r0 = (instr >> 9) & 0x7;
                    uint16_t r1 = (instr >> 6) & 0x7;
                    uint16_t offset = sign_extend(instr & 0x3F, 6);
                                if(validAdress(memory, reg[r1] + offset, ptr, 'W'))
                                {
                                       mem_write(memory, reg[r1] + offset, reg[r0]); //
valider adresse afficher message erreur mem et running = 0
                                }
                                else //invalid adress
                                {
                                       running = 0;
                                       printf("\n Program abort memory out of bound or
access violation");
                        sprintf(line, "Program abort memory out of bound or access
violation\n");
                        strcat(message, line);
                                }
                          }

                break;
            case OP_TRAP:
                /* TRAP */
                switch (instr & 0xFF)
                {
                    case TRAP_GETC:
                        /* TRAP GETC */
                        /* read a single ASCII char */
                        reg[R_R0] = (uint16_t)getchar();

                        break;
                    case TRAP_OUT:
                        /* TRAP OUT */
                        putc((char)reg[R_R0], stdout);
                        fflush(stdout);

                        break;
                    case TRAP_PUTS:
                        /* TRAP PUTS */
                    {
                            /* one char per word */
                            uint16_t* c = memory + reg[R_R0]; // valider adresse afficher
message erreur mem et running = 0
                                                    if(validAdress(c, 0, ptr, 'R'))
                                                    {
                                                           while (*c)
                                                           {
                                                                  putc((char)*c, stdout);
                                                                  ++c; // valider adresse afficher
message erreur mem et running = 0
                                                                  if(!validAdress(c, 0, ptr, 'R'))
                                                                  {
                                                                         running = 0;
                                                                         printf("\n Program abort
memory out of bound");
                                                    sprintf(line, "Program abort memory out of bound\
n");
```

```
                                    strcat(message, line);
                                                *c = 0;
                                          }
                                    }
                                    fflush(stdout);
                              }
                              else //invalid adress
                              {
                                    running = 0;
                                    printf("\n Program abort memory out of
bound");
                        sprintf(line, "Program abort memory out of bound\n");
                        strcat(message, line);
                              }
                  }

                      break;
                   case TRAP_IN:
                      /* TRAP IN */
                      {
                          printf("Enter a character: ");
                          char c = getchar();
                          putc(c, stdout);
                          reg[R_R0] = (uint16_t)c;
                      }

                      break;
                   case TRAP_PUTSP:
                      /* TRAP PUTSP */
                   {
                           /* one char per byte (two bytes per word)
                              here we need to swap back to
                              big endian format */
                           uint16_t* c = memory + reg[R_R0]; // valider adresse afficher
message erreur mem et running = 0
                                    if(validAdress(c, 0, ptr, 'R'))
                                    {
                                          while (*c)
                                          {
                                                char char1 = (*c) & 0xFF;
                                                putc(char1, stdout);
                                                char char2 = (*c) >> 8;
                                                /*
                                                char char1 = (*c) >> 8;
                                                putc(char1, stdout);
                                                char char2 = (*c) & 0xFF;
                                                */
                                                if (char2) putc(char2, stdout);
                                                ++c; // valider adresse afficher
message erreur mem et running = 0
                                                if(!validAdress(c, 0, ptr, 'R'))
                                                {
                                                      running = 0;
                                                      printf("\n Program abort
memory out of bound");
                                    sprintf(line, "Program abort memory out of bound\
n");
                                    strcat(message, line);
```

```
                                                                        *c = 0;
                                                                }
                                                        }
                                                        fflush(stdout);
                                                }
                                                else //invalid adress
                                                {
                                                        running = 0;
                                                        printf("\n Program abort memory out of
bound");
                                        sprintf(line, "Program abort memory out of bound\n");
                                        strcat(message, line);
                                                }
                    }

                        break;
                     case TRAP_HALT:
                        /* TRAP HALT */
                        puts("\n HALT");
                        fflush(stdout);
                        running = 0;

                        strcat(message, "HALT\n");

                        break;
                }

                break;
            case OP_RES:
            case OP_RTI:
            default:
                /* BAD OPCODE */
                abort();

                break;
        }
    }
    ptr->VM.busy = 0;
    /* Shutdown */
    restore_input_buffering();
    sem_post(&(ptr->semVM));
    sem_post(&semConsole);
    //return NULL; //good outcome

    sendMsgToClient(my_client_fifo, message);

    //printf("[ESCAPED EXEC]");
    pthread_exit(1);
}

//#######################################
//#
//# fonction utilisée pour le traitement  des transactions
//# ENTREE: Nom de fichier de transactions
//# SORTIE:
void* readTrans(){

    int fd;
```

```
    int res;
    char fifoBuffer[1024];
    char client_fifo[100];
    bool closeCondition;
    closeCondition = false;
    struct Info_FIFO_Transaction receivedData;

    pthread_t tid[1000];
    int nbThread = 0;
       char *tok, *sp;

    if (access(FIFO_NAME, F_OK) == -1) {
        res = mkfifo(FIFO_NAME, 0777);
        if (res != 0) {
            fprintf(stderr, "Could not create fifo %s\n", FIFO_NAME);
            exit(EXIT_FAILURE);
        }
    }

       do {

        // Open the FIFO file for reading
        if ((fd = open(FIFO_NAME, O_RDONLY)) < 0) {
            perror("Error opening FIFO for reading");
            exit(EXIT_FAILURE);
        }

        // Read data from the FIFO
        read(fd, &receivedData, sizeof(struct Info_FIFO_Transaction));

        printf("Received struct: pid_client=%d, transaction=%s\n",
receivedData.pid_client, receivedData.transaction);

        sprintf(client_fifo, "/tmp/FIFO%d", receivedData.pid_client);
        snprintf(fifoBuffer, sizeof(fifoBuffer) - 1, receivedData.transaction);

        //read(fd, fifoBuffer, sizeof(fifoBuffer) - 1);
        fifoBuffer[sizeof(fifoBuffer) - 1] = '\0';  // Ensure null-termination

        // Display the read message
        //printf("Received message: %s\n", fifoBuffer);

        // Close the FIFO file descriptor
        close(fd);

             //Extraction du type de transaction
             tok = strtok_r(fifoBuffer, " ", &sp);

             //Branchement selon le type de transaction
             switch(tok[0]){
                    case 'A':
                    case 'a':{
                           //Appel de la fonction associée
                // addItem(); // Ajout de une VM

                struct addFuncThreadArgs* ptr = (struct addFuncThreadArgs*)
malloc(sizeof(struct addFuncThreadArgs));
                           strcpy(ptr->client_fifo, (const char*)client_fifo);
```

```
                        pthread_create(&tid[nbThread++], NULL, addItem, ptr);
                        break;
            }
                case 'E':
                case 'e':{
                        //Extraction du paramètre
                        int noVM = atoi(strtok_r(NULL, " ", &sp));
                        //Appel de la fonction associée
                //removeItem(noVM); // Eliminer une VM
                struct removeFuncThreadArgs* ptr = (struct removeFuncThreadArgs*)
malloc(sizeof(struct removeFuncThreadArgs));
                ptr->_noVM = noVM;
                        strcpy(ptr->client_fifo, (const char*)client_fifo);

                        pthread_create(&tid[nbThread++], NULL, removeItem, ptr);
                        break;
            }
                case 'L':
                case 'l':{
                        //Extraction des paramètres
                        int nstart = atoi(strtok_r(NULL, "-", &sp));
                        int nend = atoi(strtok_r(NULL, " ", &sp));
                        //Appel de la fonction associée
                //listItems(nstart, nend); // Lister les VM

                struct listFuncThreadArgs* ptr = (struct listFuncThreadArgs*)
malloc(sizeof(struct listFuncThreadArgs));
                        ptr->nstart = nstart;
                        ptr->nend = nend;
                        strcpy(ptr->client_fifo, (const char*)client_fifo);

                        pthread_create(&tid[nbThread++], NULL, listItems, ptr);

                        break;
            }
                case 'X':
                case 'x':{
                        //Appel de la fonction associée
                        int noVM = atoi(strtok_r(NULL, " ", &sp));
                        char *nomfich = strtok_r(NULL, "\n", &sp);
                //printf("here is noVM %d\n", noVM);

                //executeFile(noVM, nomfich); // Executer le code binaire du fichier
nomFich sur la VM noVM

                struct exeFuncThreadArgs* ptr = (struct exeFuncThreadArgs*)
malloc(sizeof(struct exeFuncThreadArgs));
                        ptr->noVM = noVM;
                        strcpy(ptr->nomfich, (const char*)nomfich);
                        strcpy(ptr->client_fifo, (const char*)client_fifo);
                        pthread_create(&tid[nbThread++], NULL, executeFile, ptr);
                        break;
            }
          }

        } while (closeCondition == false);
```

```
        for(int i=0; i<nbThread;i++)
          pthread_join(tid[i], NULL);

        //Retour
        return NULL;
}
```