

Module : Programmation système sous UNIX

Projet de Fin Module : Calculette HPC

Rapport

Réalisé par :

- Soufyan LAARIF

Encadré par :

- Prof. C.EL AMRANI

A.U : 2021/2022

## 1. Objectif :













Développer une application de calcul « calculette HPC » en utilisant les techniques de communication entre les processus. Il s'agit de construire une interface graphique avec un langage front-end et qui contient d'autre interfaces : interface de saisi de données, interface d'affichage de résultats, etc ...

La partie back-end à développer avec le langage C pour gérer les processus et la synchronisation entre eux.

## 2. Techniques utilisées :

Nous allons développer la partie front-end avec le langage java caractérisé par ses multitudes classes, et sa simplicité dans l'implémentation des gui. Le langage C est utilisé pour exploiter sa puissance dans la programmation système sous unix, nous allons créer des sémaphores unix en c, et aussi utiliser les files de messages pour faire passer les données entre processus.

## 3. Hierarchie du projet :

Home Documents calculette2 ▾			
Name	Size	Modifié	
 Calculette.java	9.5 kB	06:22	
 file_message	17.9 kB	05:30	
 file_message.c	5.0 kB	Yesterday	
 InitVal.java	8.7 kB	04:33	
 ipcsem.h	337 bytes	22 Apr	
 MyFile.txt	2 bytes	Yesterday	
 Resultat.java	8.5 kB	06:22	
 sequentiel	16.9 kB	05:49	
 sequentiel.c	2.7 kB	05:20	
 Taches_calcul.java	21.9 kB	06:22	
 temps_execution_ipc.txt	8 bytes	Yesterday	
 temps_execution_seq.txt	8 bytes	05:20	

Le projet est constitué de quatre classes java : Calculette.java, Resultat.java, Taches\_calcul.java, InitVal.java .

La classe Calculette.java : c'est la classe principale qui fait l'appel aux autres classes secondaires, elle représente l'interface principale de l'application.

La classe Resultat pour afficher les résultats de calcul et les temps d'exécution hpc et séquentiel, la classe Taches\_calcul consacré à faire les tâches de calcul avec les opérations arithmétiques « + - \* / ».et finalement la classe InitVal , à travers laquelle on initialise les valeurs des variables a,b,c et d.

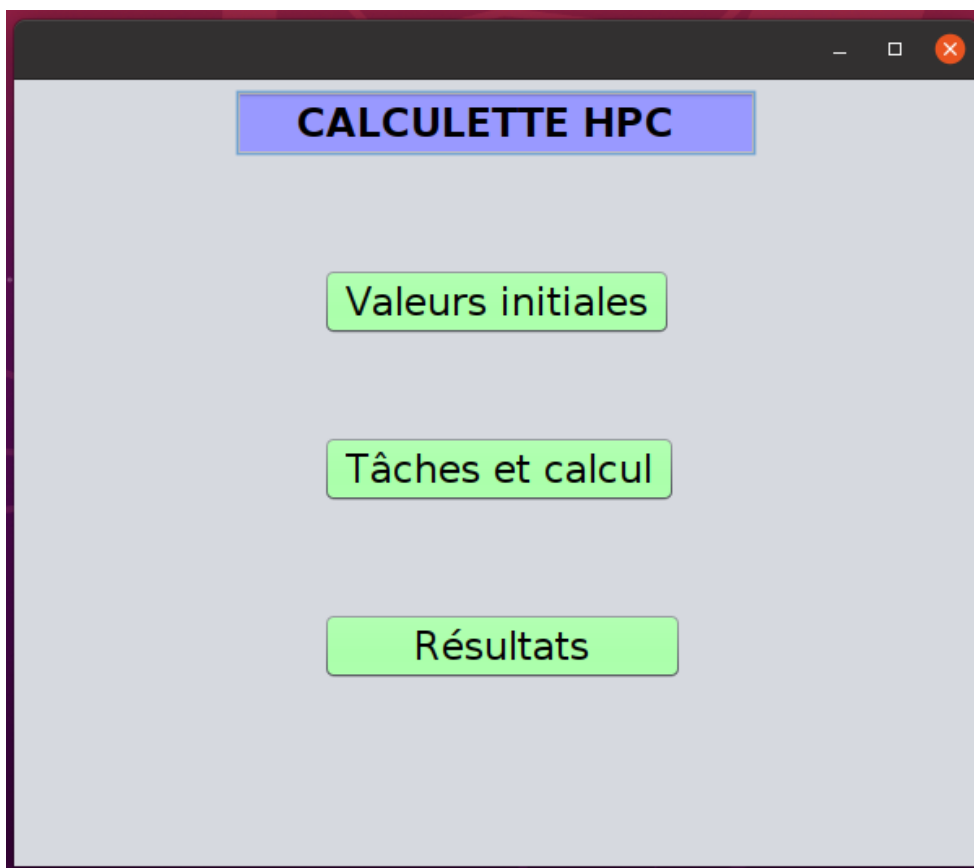
Les fichiers sequentiel.c et file\_message.c sont des fichiers partie backend pour réaliser tout le calcul nécessaire à l'aide des techniques IPC, en particulier les sémaphores unix et files de messages.

Concernant les fichiers format txt sont considérés comme des moyens pour faciliter le transfert de données entre les 2 parties : frontend et backend.

Le package calculette2 contient tous ces fichiers.

#### 4. Explication des éléments du projet :

Classe Calculette.java :



C'est une interface contenant le JFrame Calculette, elle représente le menu principale de l'application, contenant les boutons Valeurs initiales, Tâches et calcul, et Résultats.

Code java :

```
soufyan@ubuntu: ~/Documents
GNU nano 4.8      calculette2/Calculette.java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package calculette2;

import calculette2.InitVal;
import calculette2.Taches_calcul;
import calculette2.Resultat;
/**
 *
 * @author User
 */
public class Calculette extends javax.swing.JFrame {

    private static String a;
    private static String b;
    private static String c;
    private static String d;

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {>
        InitVal t = new InitVal();
        //t.setSize(270, 160);
        t.setVisible(true);
        this.setVisible(false);
        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.dispose();
    } //GEN-LAST:event_jButton1ActionPerformed

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {>

        Taches_calcul tc= new Taches_calcul();

        tc.my_update2(a, b, c, d);
        //tc.setSize(270, 160);
        tc.setVisible(true);
        this.setVisible(false);
        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.dispose();
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {>
        Resultat r= new Resultat();

        //r.my_update2(a, b, c, d);
```

```

        r.my_update2(resultat, temps_ipc, temps_seq);
        //Resultat r = new Resultat();
        //r.setSize(270, 160);
        r.setVisible(true);
        this.setVisible(false);
        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.dispose();
    } //GEN-LAST:event_jButton3ActionPerformed

    public void my_update3(String s1, String s2, String s3, String s4) {
        a = s1;
        b = s2;
        c = s3;
        d = s4;
    }

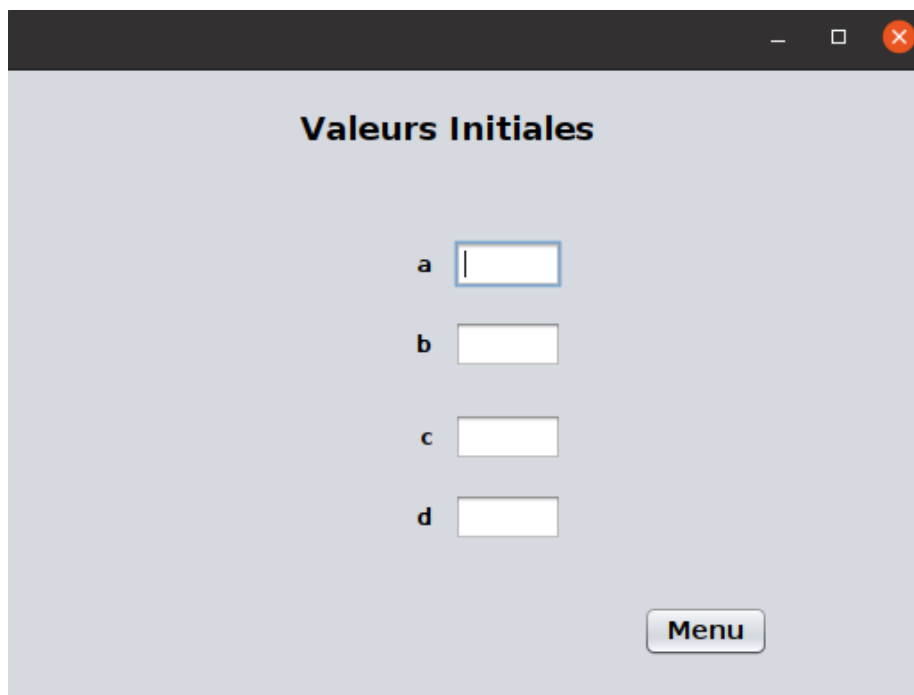
    public void my_update4(String s1, String s2, String s3) {
        resultat = s1;
        temps_ipc = s2;
        temps_seq = s3;
    }
}

```

Nous avons dans cette partie de code trois methodes de type actionPerformed associées à chaque bouton pour rediriger vers les autres interfaces suivant le clique de l'utilisateur. Les methodes my\_update3 et my\_update4 ont pour but de transférer les valeurs a,b,c,d, resultat, temps\_ipc et temps\_seq entre les interfaces pour les utiliser.

### Classe InitVal.java :

Quand l'utilisateur clique sur le bouton Valeurs initiales, il se redirige vers l'interface de l'initialisation des valeurs a,b,c,d.



The screenshot shows a Java Swing window titled "Valeurs Initiales". The window has a light gray background and a dark gray title bar with standard window controls (minimize, maximize, close). Inside the window, there are four text input fields labeled 'a', 'b', 'c', and 'd' arranged vertically. Below these fields is a button labeled "Menu".

Code java :

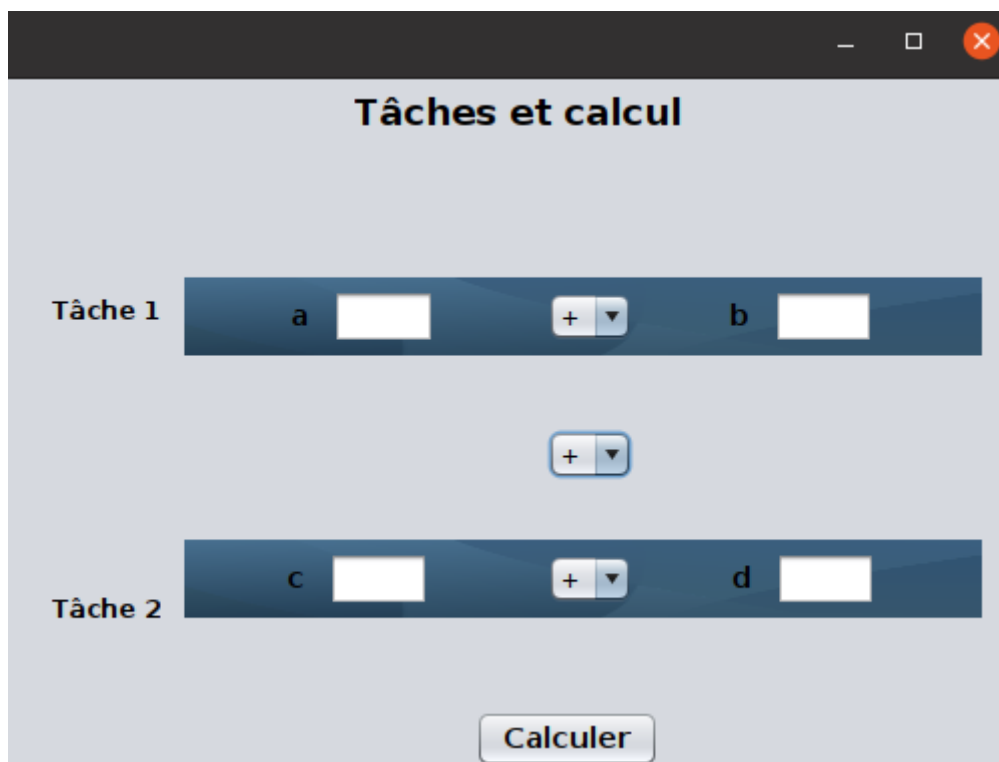
```
soufyan@ubuntu: ~/Documents
GNU nano 4.8      calculette2/InitVal.java

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {>
    String s1 = jTextField1.getText();
    String s2 = jTextField2.getText();
    String s3 = jTextField3.getText();
    String s4 = jTextField4.getText();
    Calculette cl= new Calculette();
    cl.my_update3(s1, s2, s3, s4);
    //cl.setSize(270, 160);
    cl.setVisible(true);
    this.setVisible(false);
    this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
    this.dispose();
} //GEN-LAST:event_jButton1ActionPerformed
```

Cette classe contient la méthode `jButton1ActionPerformed` qui nous permet de se rediriger vers le menu suivant un clique sur le bouton menu, et au même temps de passer ces quatre valeurs à une autre interface qui fait le calcul des tâches.

Classe Taches\_calcul.java :

Tout le calcul et tâches sont affichés sur cette interface qui recoit les valeurs initiales à partir de l'interface Valeurs initiales, l'utilisateur a le choix des opérations à réaliser au niveau du tâche 1 et tâche 2, les deux tâches sont reliés par une autre opération arithmétique. Le clique sur le bouton Calculer nous redirige vers l'interface d'affichage.



Code java :

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_jButton1ActionPerformed
    //----- added -----
    int a = 0, b = 0, c = 0, d = 0;
    a = Integer.parseInt(jTextField1.getText());
    b = Integer.parseInt(jTextField2.getText());
    String op1 = jComboBox1.getSelectedItem().toString();
    c = Integer.parseInt(jTextField3.getText());
    d = Integer.parseInt(jTextField4.getText());
    String op2 = jComboBox3.getSelectedItem().toString();
    String op3 = jComboBox2.getSelectedItem().toString();

    // -----Calculer avec le programme IPC : file_message.c-----
    try {
        String s1=String.valueOf(a);
        String s2=String.valueOf(b);
        String s3=String.valueOf(op1);
        String s4=String.valueOf(c);
        String s5=String.valueOf(d);
        String s6=String.valueOf(op2);
        String s7=String.valueOf(op3);
        // la fonction exec n'accepte que des string
        Process process = Runtime.getRuntime().exec(new String [] { "./file_message.c", s1, s2, s3, s4, s5, s6, s7 });
        StringBuilder output = new StringBuilder();
        BufferedReader reader = new BufferedReader(new InputStreamReader (process.getInputStream()));
        String line, line2;
        while((line = reader.readLine()) != null) {
            output.append(line + "\n");
        }
        System.out.println("Success");
        System.out.println(output);
    } catch (IOException exception1) {
        exception1.printStackTrace();
    }
    // -----Calculer avec le programme séquentiel : sequentiel.c-----
    try {
        String s1=String.valueOf(a);
        String s2=String.valueOf(b);
        String s3=String.valueOf(op1);
        String s4=String.valueOf(c);
        String s5=String.valueOf(d);
        String s6=String.valueOf(op2);
        String s7=String.valueOf(op3);
        Process process = Runtime.getRuntime().exec(new String [] { "./sequentiel.c", s1, s2, s3, s4, s5, s6, s7 });
        StringBuilder output = new StringBuilder();
        BufferedReader reader = new BufferedReader(new InputStreamReader (process.getInputStream()));
        String line, line2;
        while((line = reader.readLine()) != null) {
            output.append(line + "\n");
        }
        System.out.println("Success");
    }
}
```

On a 2 blocs de code try ... catch qui font le transfert des valeurs entrées par l'utilisateur et aussi les opérations arithmétiques vers le code C par l'intermédiaire de la méthode `getRuntime().exec()` dans laquelle on passe les paramètres au code C à exécuter, dans notre projet on a 2 fichiers .c : `sequentiel.c` qui le fait le calcul séquentiel et calcule le temps d'exécution en séquentiel, et le fichier `file_message.c` qui fait le calcul parallèle ou bien concurrent et calcule le temps d'exécution aussi.

## Fichier sequentiel.c :

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/msg.h>
#include<unistd.h>
#include<time.h> // Pour calculer le temps d'exécution

int main(int argc, char* argv){

    clock_t t;
    t = clock();
    // Sauvegarder les variables passé en parametres
    int a      = atoi(argv[1]);
    int b      = atoi(argv[2]);
    char* op1  = argv[3];
    int c      = atoi(argv[4]);
    int d      = atoi(argv[5]);
    char* op2  = argv[6];
    char* op3  = argv[7];
```

Les variables a,b,op1,c,d,op2,op3 sont les variables déjà passés en paramètre depuis le code java.

```
    // Calculer : resultat_a_b
    switch(*op1){
        case ('*'):
            resultat_a_b = a*b;
            break;
        case ('+'):
            resultat_a_b = a+b;
            break;
        case ('-'):
            resultat_a_b = a-b;
            break;
        case ('/'):
            resultat_a_b = a/(float)b;
            break;
    }
    printf("\n---Opération entre a et b : %d---\n", resultat_a_b);
```

Puisque le calcul doit être séquentiel, alors il n'y aura pas d'utilisation de techniques IPC, il suffit d'utiliser des switch, dans notre code il y a trois switch, chacune fait un calcul ou bien une tâche spécifique, switch 1 fait la tâche 1, switch 2 fait la tâche 2, et switch 3 fait l'opération entre les tâches 1 et 2.

Et pour calculer le temps d'exécution on a utilisé la fonction clock() de la bibliothèque time.h, le résultat à afficher est stocké dans un fichier txt. Cette fonction est utilisée dans les deux manières de calcul séquentiel et parallèle.



```

t = clock() - t;

double temps_seq = ((double)t)/CLOCKS_PER_SEC; // in seconds
printf("Le programme séquentiel prend %f secondes en exécution\n", v);
// Création du fichier temps_sequentiel_seq.txt
FILE *seq_ptr;
seq_ptr = fopen("temps_execution_seq.txt", "w+");
fprintf(seq_ptr, "%f", temps_seq);
printf("After creation of temps_seq file");

```

### Fichier file\_message.c

```

#include "ipcsem.h"
#include "stdlib.h"
#include "string.h"
#include <stdio.h>
#include <sys/msg.h>
#include <unistd.h>
#include <time.h> // Pour calculer le temps d'exécution

int Num;
// prototype des fonctions
void P1(int a, int b, char* op1); // processuse 1
void P2(int c, int d, char* op2); // processuse 2
void P3(char* op3); // processuse 3

/* Structure pour envoyer ou recevoir les msgs */
typedef struct{
    long mtype;
    int mcontent;
}msg;
msg msg1, msg2, msg3;
#define LGMES sizeof(msg1.mcontent)

```

Dans ce code, on a utilisé les files de messages pour faire passer les données entre processus et l'utilisation des sémaphores unix pour la synchronisation entre les 3 processus P1,P2,P3 .

La file de messages créée est de clé externe 10, mtype du message 1 (tâche 1) = 100, et le mtype du message 2 (tâche 2) = 200.

Voici le code de ces processus :

```

void P1(int a, int b, char* op1){
    if(fork()==0){
        int resultat_a_b;
        int numfms=msgget(10,IPC_CREAT|0666);
        printf("id interne de P1 : %d\n", numfms);
        if(numfms==-1){
            perror("case 1 erreur msgget");
            exit(1);
        }
        // Calculer l'opération entre a et b
        switch(*op1){
            case ('*'):
                resultat_a_b = a*b;
                break;
            case ('+'):
                resultat_a_b = a+b;
                break;
            case ('-'):
                resultat_a_b = a-b;
                break;
            case ('/'):
                resultat_a_b = a/(float)b;
                break;
        }
        msg1.mtype = 100;
        msg1.mcontent = resultat_a_b;
        if(msgsnd(numfms,&msg1,LGMES,IPC_NOWAIT)==-1){
            perror("case 4 erreur msgsnd");
        }
    }
}

```

```

void P2(int c, int d, char* op2){
    if(fork()==0){
        int numfms = msgget(10,IPC_CREAT|0666);
        printf("id interne de P2 : %d\n", numfms);
        if(numfms==-1){
            perror("case 1 erreur msgget");
            exit(1);
        }
        int resultat_c_d;
        switch(*op2){
            case ('*'):
                resultat_c_d = c*d;
                break;
            case ('+'):
                resultat_c_d = c+d;
                break;
            case ('-'):
                resultat_c_d = c-d;
                break;
            case ('/'):
                resultat_c_d = c/(float)d;
                break;
        }
        msg2.mtype = 200;
        msg2.mcontent = resultat_c_d;
        if(msgsnd(numfms,&msg2,LGMES,IPC_NOWAIT)==-1){
            perror("case 4 erreur msgsnd");
        }
    }
}

```

Chaque processus fait une tâche, et le résultat est calculé et stocké dans un fichier Myfile.txt par le processus P3.

```

void P3(char* op3){
    if(fork()==0){
        //attent processus P1 et P2 en utilise semaphore unix pour
        P(0);
        P(1);
        //int a,b;
        int resultat;
        int numfms = msgget(10,IPC_CREAT|0666);
        printf("id interne de P3 : %d\n", numfms);
        int n1=msgrcv(numfms,&msg1,LGMES,100,IPC_NOWAIT); /* type
        int n2=msgrcv(numfms,&msg2,LGMES,200,IPC_NOWAIT); /* type
        if((n1 < 0) || (n2 < 0)){
            perror("case 5 erreur msgrcv");
            exit(4);
        }

        //read(tub1[0],&a,4);
        //read(tub2[0],&b,4);

        if(strcmp(op3, "+") == 0)
            resultat = msg1.mcontent + msg2.mcontent;
        else if(strcmp(op3, "-") == 0)
            resultat = msg1.mcontent - msg2.mcontent;
        else if(strcmp(op3, "*") == 0)
            resultat = msg1.mcontent * msg2.mcontent;
        else if(strcmp(op3, "/") == 0)
            resultat = msg1.mcontent / (float)msg2.mcontent;
        //write(tub3[1], &r, 4);

        resultat = msg1.mcontent / (float)msg2.mcontent;
        //write(tub3[1], &r, 4);
        printf("\n-----Resultat total : %d-----\n", resultat);
        /* Ecrire les resultats dans un fichier, pour les lres depuis java */
        FILE *fpt;
        fpt = fopen("MyFile.txt", "w+");
        fprintf(fpt, "%d", resultat);
        fclose(fpt);
        exit(0);
    }
}

```

### Classe Taches\_calcul.java (suite) :


```
if(evt.getSource()==jButton1) {
    String resultat      = "";
    String currentNumber = "";
    File file = new File("MyFile.txt");
    BufferedReader br = null;
    try{
        br = new BufferedReader(new FileReader(file));
        while((currentNumber = br.readLine()) != null){
            resultat = resultat + currentNumber;
        }
    }catch(Exception ex){
        ex.printStackTrace();
    }
    System.out.println("Ce fichier Contient ce nombre : " + resultat);
    // Lire le temps d'exécution depuis le fichier temps_execution_ipc.txt
    String temps_seq      = "";
    String currentNumber2 = "";
    File file2 = new File("temps_execution_seq.txt");
    BufferedReader br2 = null;
    try{
        br2 = new BufferedReader(new FileReader(file2));
        while((currentNumber2 = br2.readLine()) != null){
            temps_seq = temps_seq + currentNumber2;
        }
    }catch(Exception ex3){
        ex3.printStackTrace();
    }
}
```

Cette partie de code se contente de lire le résultat total et les temps d'exécution à partir des fichiers Myfile.txt, temps\_execution\_seq.txt, temps\_execution\_ipc.txt.

La méthode suivante : my\_update envoie les données à afficher vers l'interface Resultat.

```
Resultat r = new Resultat();
r.my_update(resultat, ipc_temps_execution, temps_seq);
r.setVisible(true); // Open the Second.java window
this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
this.dispose();
```

Classe Resultat.java :



**Résultat**

Résultat

Temps HPC

Temps sequentiel

Menu

Code java :

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event
    Calculette cl = new Calculette();
    cl.my_update4(jTextField1.getText(), jTextField2.getText(), jTextField3.getText());
    //cl.setSize(270, 160);
    cl.setVisible(true);
    this.setVisible(false);
    this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
    this.dispose();
} //GEN-LAST:event_jButton1ActionPerformed

public void my_update(String r, String hpc, String seq) {
    jTextField1.setText(r);
    jTextField2.setText(hpc);
    jTextField3.setText(seq);
}

public void my_update2(String r, String hpc, String seq) {
    jTextField1.setText(r);
    jTextField2.setText(hpc);
}
```

C'est une interface graphique d'affichage des résultats, le bouton menu permet de retourner vers le menu principale, les 2 méthodes my\_update et my\_update2 pour mettre à jour les cases (textFields) et passer les données.

## 5. Démonstration et exécution :

Dans cette partie nous allons suivre un scénario d'exécution pour s'assurer du bon fonctionnement de notre programme.

### La compilation des programmes C et Java.

```
soufyan@ubuntu:~/Documents$ cc sequentiel.c -o sequentiel  
soufyan@ubuntu:~/Documents$ cc file_message.c -o file_message  
soufyan@ubuntu:~/Documents$ javac calculette2/Calculette.java  
soufyan@ubuntu:~/Documents$
```

Compilation sans erreurs.

### L'exécution de l'exécutable principale Calculette :

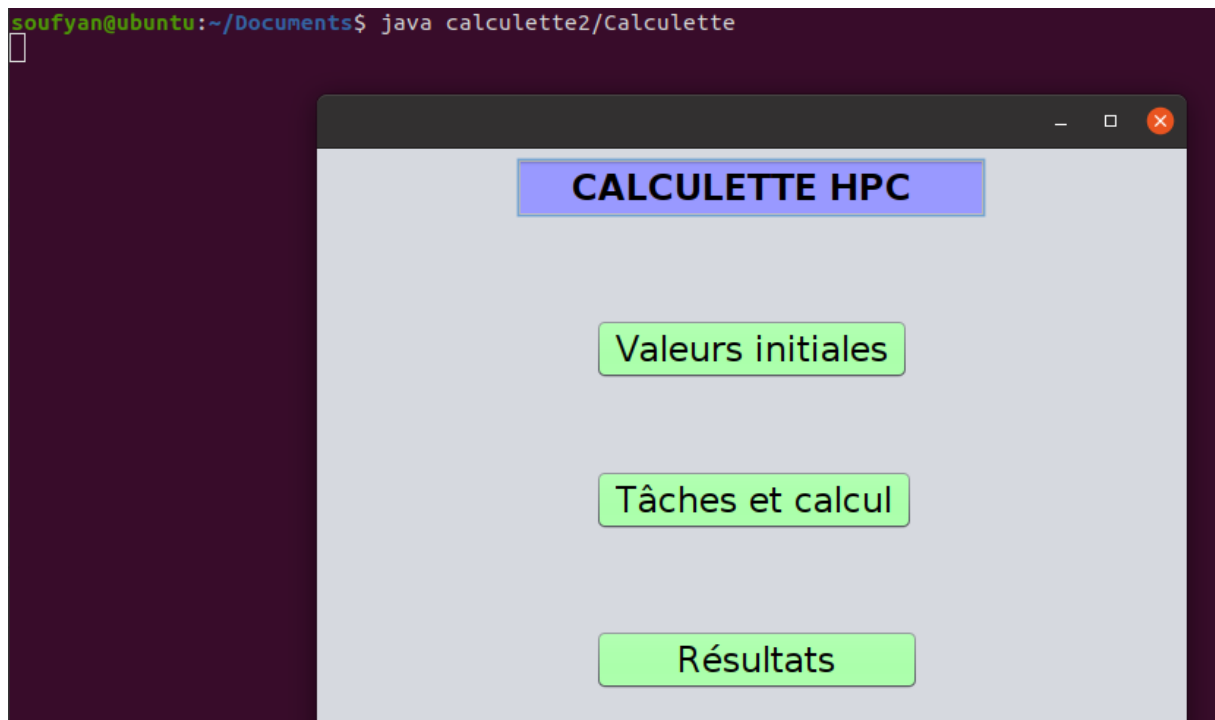
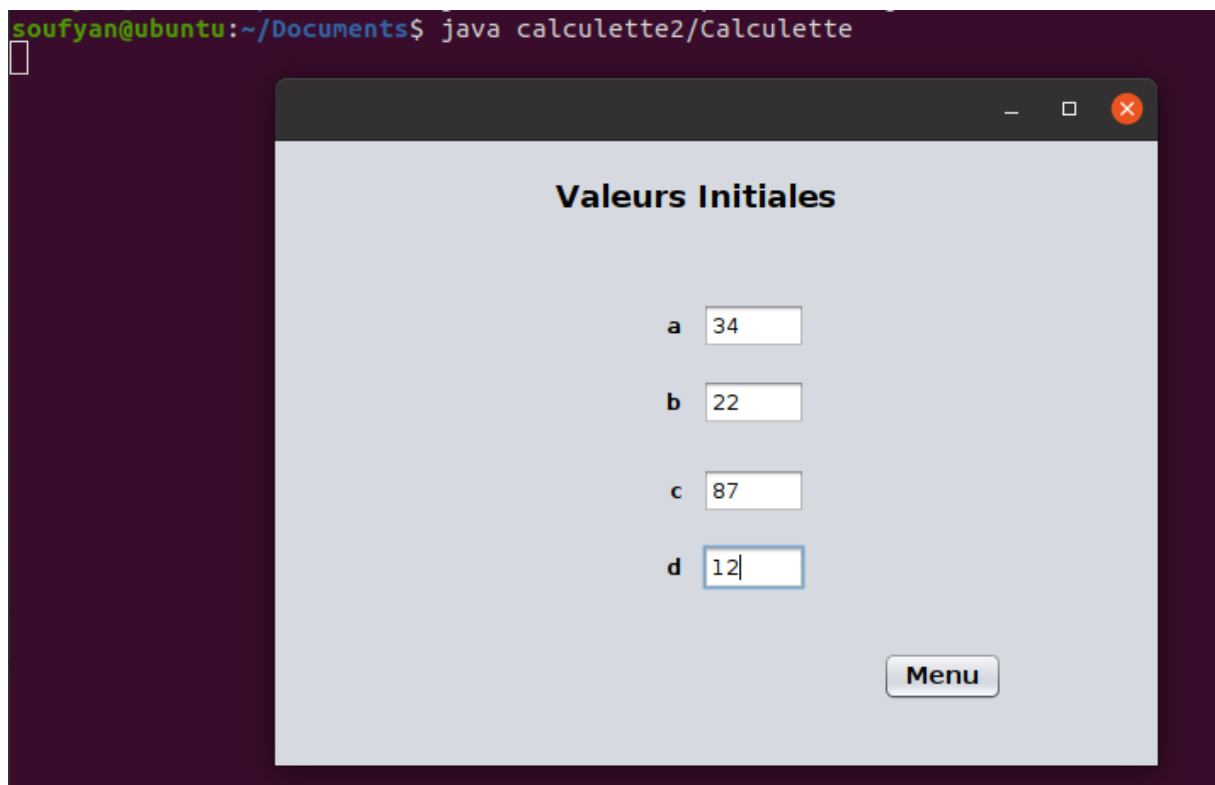
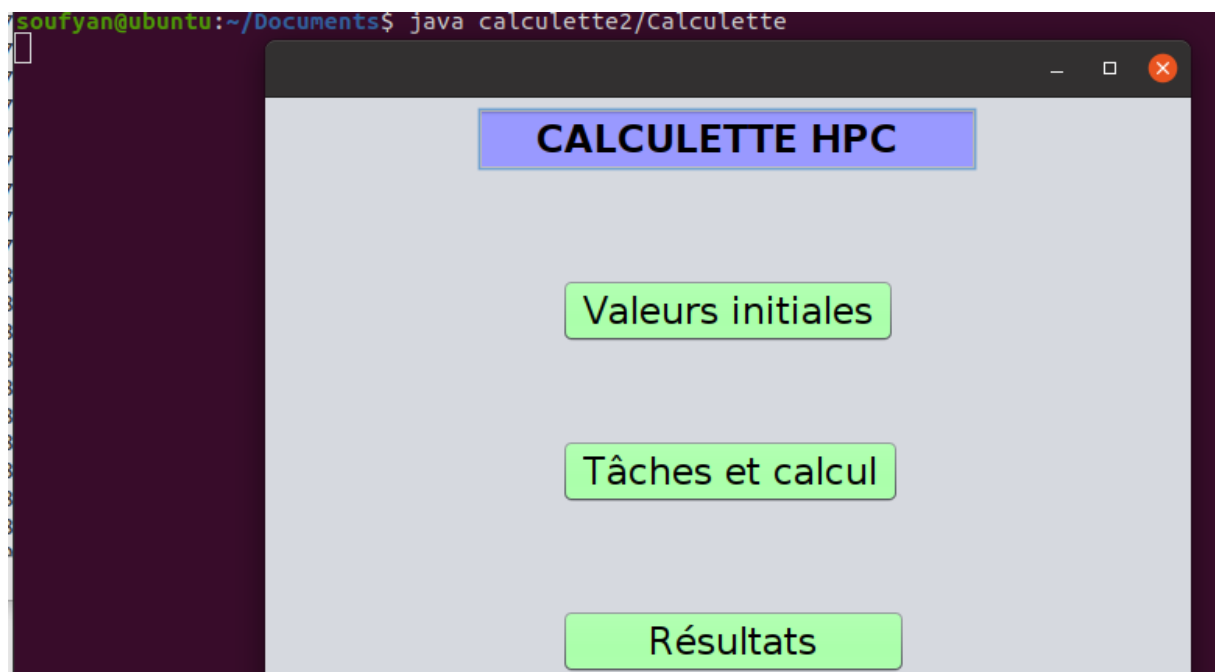


Figure 1 : Menu principale

Saisir des valeurs a,b,c et d et cliquer sur menu pour revenir au menu principale :

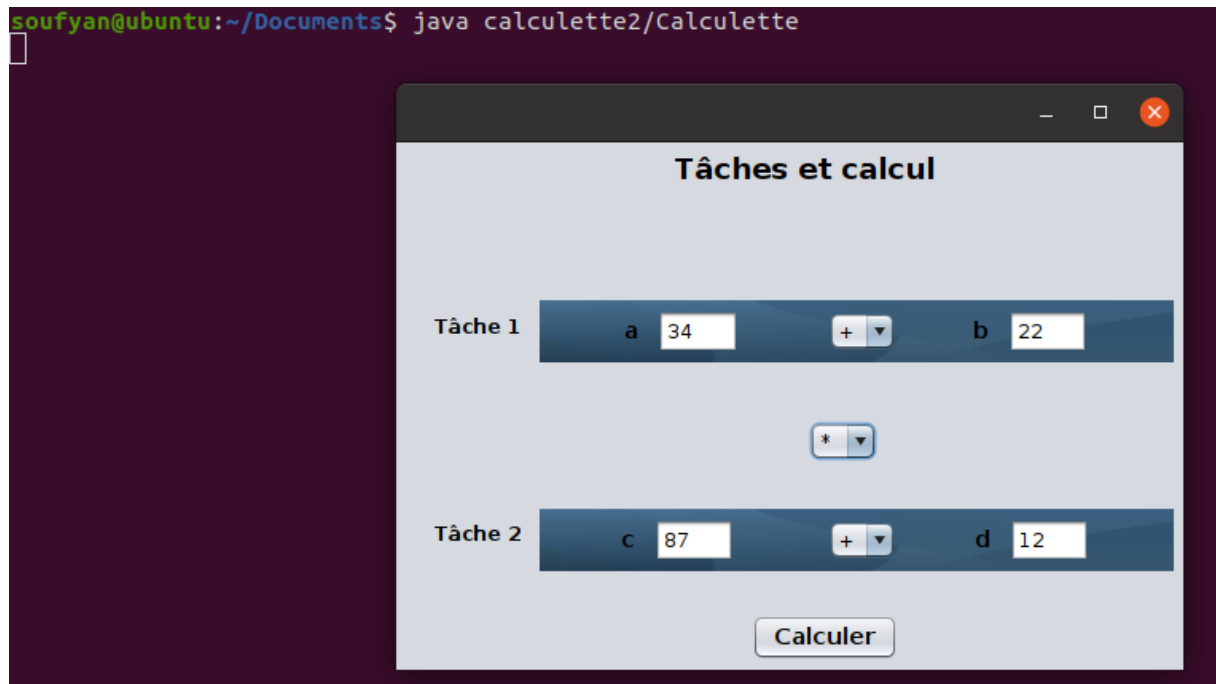


Revenir au menu principale et choisir Tâches et calcul :



Une fois cliquer sur Tâches et calcul les valeurs s'affichent automatiquement sans les resaisir une autre fois , l'utilisateur fait juste le choix des opérations. Dans cet exemple on veut réaliser l'opération suivante :  $(34 + 22) * (87 + 12)$

Après cliquer sur le bouton calculer la fenêtre Résultat s'affichent. Elle affiche le résultat totale et les temps d'exécution.



**Remarque** : le temps d'exécution en sequentiel est inférieur à celui en parallèle. Parce que le calcul en parallèle (concurrent) passe par plusieurs étapes, parmi eux la création d'une file de message, utilisation des sémaphores, par contre en sequentiel il y a enchaînement de switch statements qui fait le tout sans besoin d'intermédiaire.

Les fichiers de sauvegarde après exécution des codes :

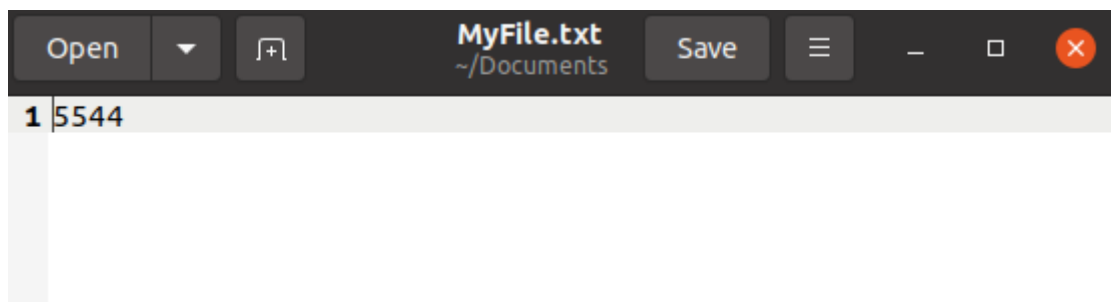


Figure 2 : Fichier MyFile.txt

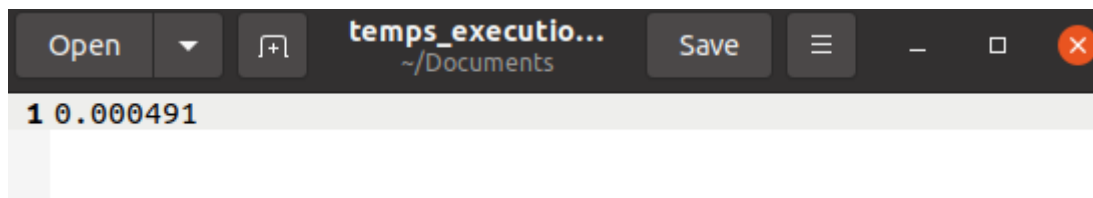


Figure 3 : Fichier temps\_execution\_ipc

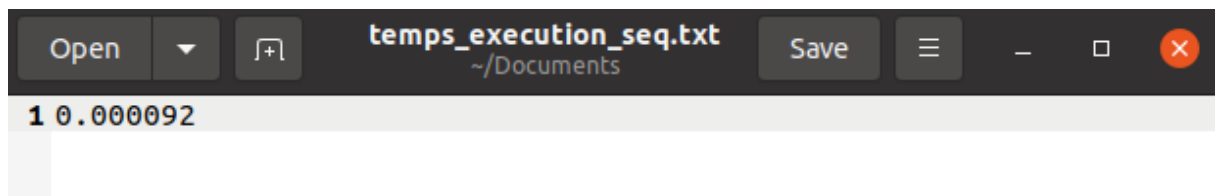


Figure 4 : Fichier temps\_execution\_seq