## Exercice 1 :

On redirige d’abord la sortie vers un fichier, puis on exécute la commande en paramètre.

        int fdout = open(argv[1],O\_WRONLY | O\_CREAT, 0600);

        dup2(fdout, STDOUT\_FILENO);

        close(fdout);

        execvp(args[0], args);

## Exercice 2 :

Le fils redirige ses sorties standard et d’erreur vers /dev/null et renvoie le statut de la commande entrée en paramètre.

int execute\_fils(char \*\*args)

{

    int fdout = open("/dev/null",O\_WRONLY);

    // Redirige la sortie standard vers /dev/null

    dup2(fdout, STDOUT\_FILENO);

    // Redirige également la sortie d'erreur - "Le  programme ne  devra  rien afficherd’autre que «OK»ou «ERREUR». "

    dup2(fdout, STDERR\_FILENO);

    close(fdout);

    return execvp(args[0],args);

}

Le père attend la fin de l’exécution du fils pour afficher « OK » ou « ERREUR » en fonction de la valeur que le fils a renvoyée.

waitpid(pidt,&status, 0);

    if ( WIFEXITED(status) ) {

        int es = WEXITSTATUS(status);

        if(es!=0){

            error++;

        }

    }

    if(error==0){

        printf("OK\n");

        return EXIT\_SUCCESS;

    }else{

        printf("ERREUR\n");

        return error;

    }

## Exercice 3 :

On crée une pipe pour chaque processus puis on fork.

        if (pipe(pipefds + i\*2) == -1) {

            perror("pipe");

            exit(EXIT\_FAILURE);

        }

        fflush(stdout);

        if ((pid[i] = fork()) == -1){

            perror("fork");

            exit(EXIT\_FAILURE);

        }

Chaque fils poursuit son exécution dans sa fonction associée

if(pid[i] == 0){ //fils

            if(i==0){

                addition(pipefds + i\*2);

            }else if(i==1){

                multiplication(pipefds + i\*2);

            }else{

                soustraction(pipefds + i\*2);

            }

            exit(EXIT\_FAILURE);

        }

Le père attend en entrée des commandes et les écrit dans des messages destinés aux processus concernés

while (1){

        scanf("%s %s %s", &operation, &a, &b);

        memcpy(ab[0], a, sizeof(ab[0]));

        memcpy(ab[1], b, sizeof(ab[0]));

        if(strcmp(operation,"addition")==0){

            write(pipefds[1],ab,sizeof(ab));

        }else if(strcmp(operation,"multiplication")==0){

            write((pipefds+2)[1],ab,sizeof(ab));

        }else if (strcmp(operation,"soustraction")==0){

            write((pipefds+4)[1],ab,sizeof(ab));

        }else{

            for (i=0; i<3; i++){

                close((pipefds + i\*2)[1]);

                kill(pid[i], SIGKILL);

            }

            exit(EXIT\_SUCCESS);

            break;

        }

    }

Alors que chaque co-processus lit le message de la pipe et affiche le résultat du calcul des paramètres d’entrée.

    close(pipe[1]);

    char ab [2][100];

    while(1){

        //lit l'entrée standard en boucle

        read(pipe[0],ab,sizeof(ab));

        printf("%s + %s = ",ab[0],ab[1]);

        int sum = atoi(ab[0])+atoi(ab[1]);

        //écrit sur la sortie standard

        printf("%d\n> ",sum);

    }

On a alors dans son fonctionnement

kali@RYZEN:/mnt/d/Documents/Polytech/Système/TP2/ex3$ ./dispatch

Entrez un calcul (addition / multiplication / soustraction)

ex: "addition 2 4"

> soustraction 9 6

9 - 6 = 3

> multiplication 3 9

3 \* 9 = 27

> addition 2 4

2 + 4 = 6

## Exercice 4 :

On crée d’abord une structure pour envoyer en argument des threads leur durée de parcours hors du tunnel et leur nom. On crée aussi un mutex.

#define MAX\_TIME 30

#define MIN\_TIME 10

#define TUNNEL\_TIME 4

sem\_t mutex;

struct arg\_struct {

    int random\_time;

    char\* name;

};

On renseigne les informations des trains dans des structures associées

    sem\_init(&mutex, 0, 1);

    pthread\_t RER\_A,RER\_B,RER\_C,TrainDeBanlieue;

    srand(time(NULL));

    struct arg\_struct argsA = {rand() % (MAX\_TIME + 1 - MIN\_TIME) + MIN\_TIME, "RER A"};

    struct arg\_struct argsB = {rand() % (MAX\_TIME + 1 - MIN\_TIME) + MIN\_TIME, "RER B"};

    struct arg\_struct argsC = {rand() % (MAX\_TIME + 1 - MIN\_TIME) + MIN\_TIME, "RER C"};

    struct arg\_struct argsT = {rand() % (MAX\_TIME + 1 - MIN\_TIME) + MIN\_TIME, "Train de banlieue"};

Puis on démarre les threads

    // Création des threads

    pthread\_create(&RER\_A,NULL,train,(void \*)&argsA);

    pthread\_create(&RER\_B,NULL,train,(void \*)&argsB);

    pthread\_create(&RER\_C,NULL,train,(void \*)&argsC);

    pthread\_create(&TrainDeBanlieue,NULL,train,(void \*)&argsT);

Le thread affiche son nom et sa durée de parcours. Il affiche lorsqu’il arrive devant le tunnel et qu’il y entre. Le sémaphore utilisé encadre la section critique (tunnel) de sem\_wait(&mutex); et sem\_post(&mutex);

void\* train(void\* arguments)

{

    struct arg\_struct \*args = arguments;

    while (1){

        printf("\nLe %s fait un tour de %d secondes..\n", args -> name, args -> random\_time);

        //Parcourt une distance pendant une durée aléatoire

        sleep(args -> random\_time);

        printf("\nLe %s est devant le tunnel\n", args -> name);

        sem\_wait(&mutex);

        printf("\n\tLe %s entre dans le tunnel..\n", args -> name);

        //Section critique

        sleep(TUNNEL\_TIME);

        //signal

        sem\_post(&mutex);

    }

}

Ainsi, on a par exemple en sortie :

kali@RYZEN:/mnt/d/Documents/Polytech/Système/TP2/ex3$ ./dispatch

Le RER A fait un tour de 6 secondes..

Le RER B fait un tour de 19 secondes..

Le RER C fait un tour de 13 secondes..

Le Train de banlieue fait un tour de 15 secondes..

Le RER A est devant le tunnel

Le RER A entre dans le tunnel..

Le RER A fait un tour de 6 secondes..

Le RER C est devant le tunnel

Le RER C entre dans le tunnel..

Le Train de banlieue est devant le tunnel

Le RER A est devant le tunnel

Le RER C fait un tour de 13 secondes..

Le Train de banlieue entre dans le tunnel..

Le RER B est devant le tunnel

Le Train de banlieue fait un tour de 15 secondes..

Le RER A entre dans le tunnel..

Le RER A fait un tour de 6 secondes..

Le RER B entre dans le tunnel..

Le RER B fait un tour de 19 secondes..

Le RER C est devant le tunnel

Le RER C entre dans le tunnel..

Le RER A est devant le tunnel