Compte rendu TP

Système multicoeurs et multiprocesseurs

Khairi SLAMA

Les objectifs du TP:

- **Initialisation a l'environnement Ubuntu avec extension Xenomai**
- **Utilisation d'un system temps réel strict Qualifié mais non-cértifié**
- **Manipulation des parallélismes des données et de fonctions**
- **Manipulation des tâches sur un systeme**
- **Création d'une tâche**
- **Manipulation des fifos**

Table of Contents

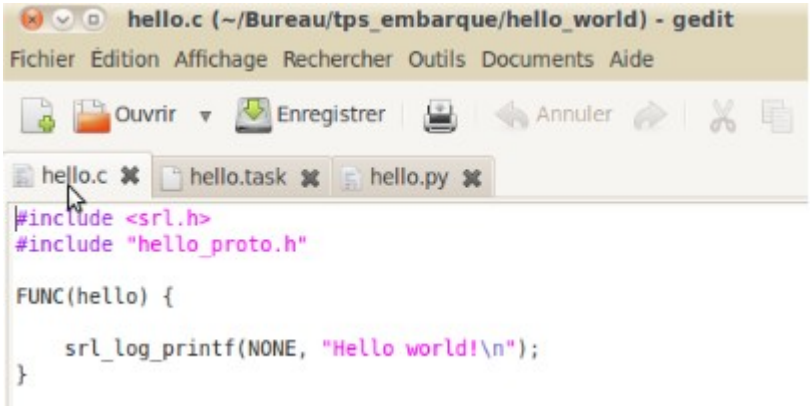
Khairi Slama 4 eme G informatique Groupe 2.2

Exercice 1 : Manipulation d'une seule tache

Dans cet exemple on va executé une tâche d'une façon synchrone qui va afficher le message Hello World !

Tout d'abord il faut implémenter le code suivant :

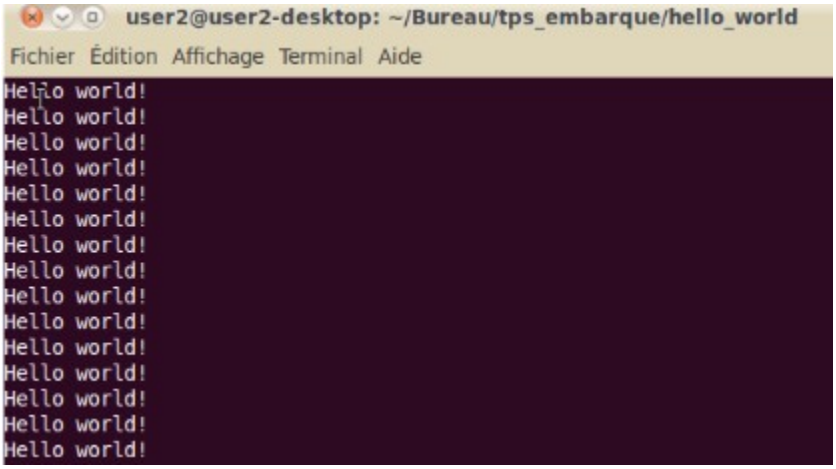
1) hello.c



2) hello.task

3) hello.py

On tape la commande ./hello.py qui va créer un ficher exécutable nommé « exe.posix » et tape après la commande ./exe.posix et on trouve l'execution qui se trouve dans la figure ci-dessous :



Khairi Slama 4 eme G informatique Groupe 2.2

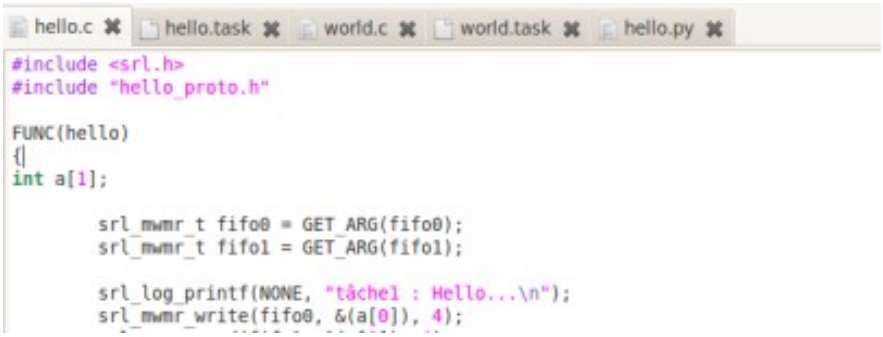
Exercice 2 : Deux taches synchrones

Dans cet exemple on va executé deux tâches d'une façon synchrone qui va afficher les messages hello et world

Lorsque la première tâche s'exécute elle envoie un signal à la deuxième tache pour s'éxecuter le signial est envoyer grâce a l'algorithme de FIFO « first in first out »

Tout d'abord il faut implémenter le code suivant :

1) hello.c

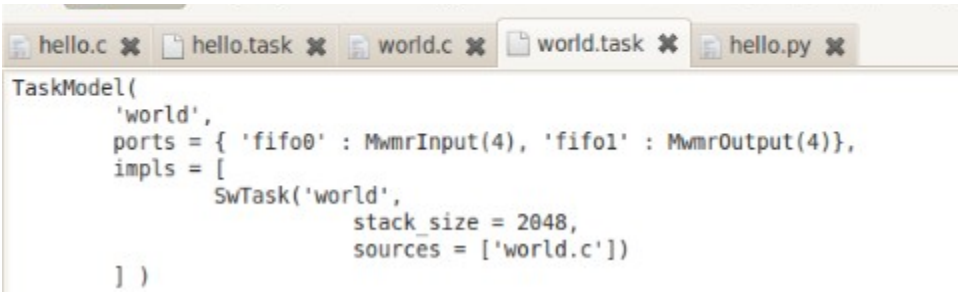


2) hello.task

Khairi Slama 4 eme G informatique Groupe 2.2

3) word.c

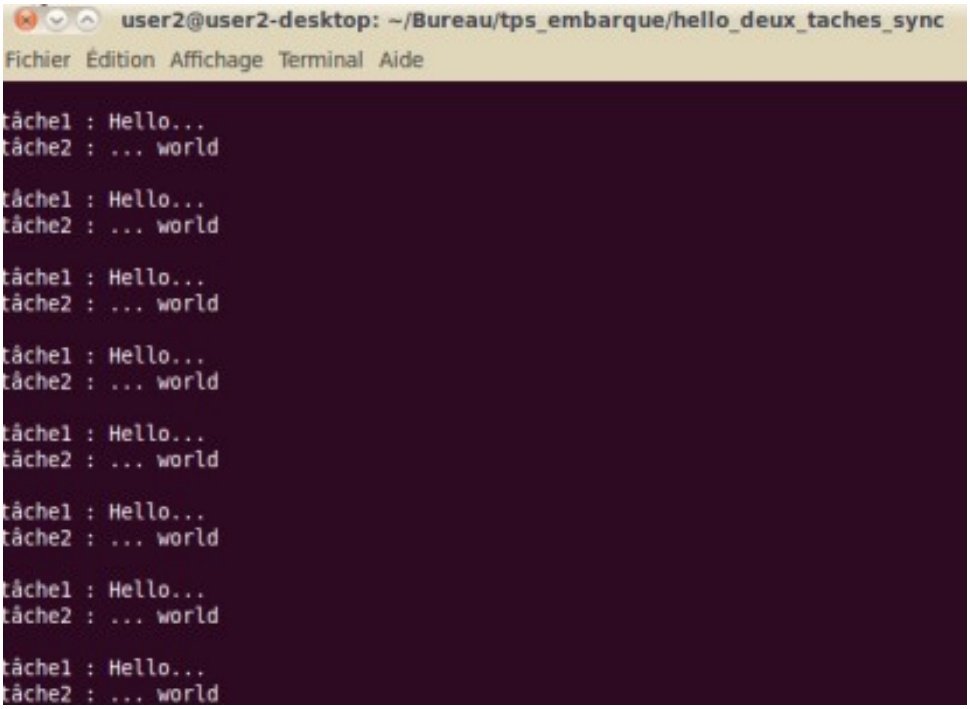
4) word.task



3) hello.py

Khairi Slama 4 eme G informatique Groupe 2.2

Après avoir implémenter le code suivant on execute le programme on suivant l'exécution dans l'exemple 1 de la seule tâche et on obtient un résultat comme suit :



Khairi Slama 4 eme G informatique Groupe 2.2

Exemple 3 : Trois taches synchrones

Dans cet exemple on va exécuter trois taches synchrones qui va afficher les trois messages hello, world et Poly donc on va implémenter le code suivant :

1) hello.c

hello.c | hello.task | poly.c | poly.task | world.c | world.task | hello.py

```
#include <srl.h>
#include "hello_proto.h"

FUNC(hello)
{
int a[1] = {1};

        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        srl_log_printf(NONE, "tâche1 : Hello.....\n");
        srl_mwmr_write(fifo0, &(a[0]), 4);
        srl_mwmr_read(fifo2, &(a[0]), 4);
        }
```

2) hello.task

hello.c | hello.task | poly.c | poly.task | world.c | world.task | hello.py

```
TaskModel(
        'hello',
        ports = { 'fifo0' : MwmrOutput(4), 'fifo2' : MwmrInput(4) },
        impls = [
                SwTask('hello',
                        stack_size = 2048,
                        sources = ['hello.c'])
        ] )
```

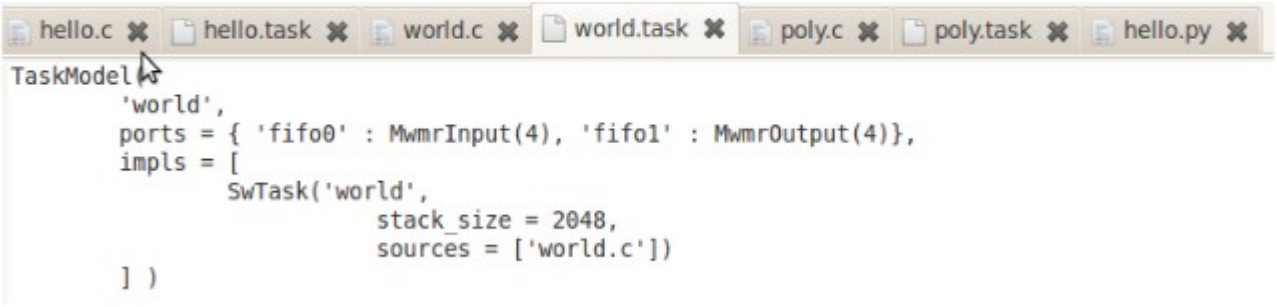
3) word.c

hello.c | hello.task | world.c | world.task | poly.c | poly.task | hello.py

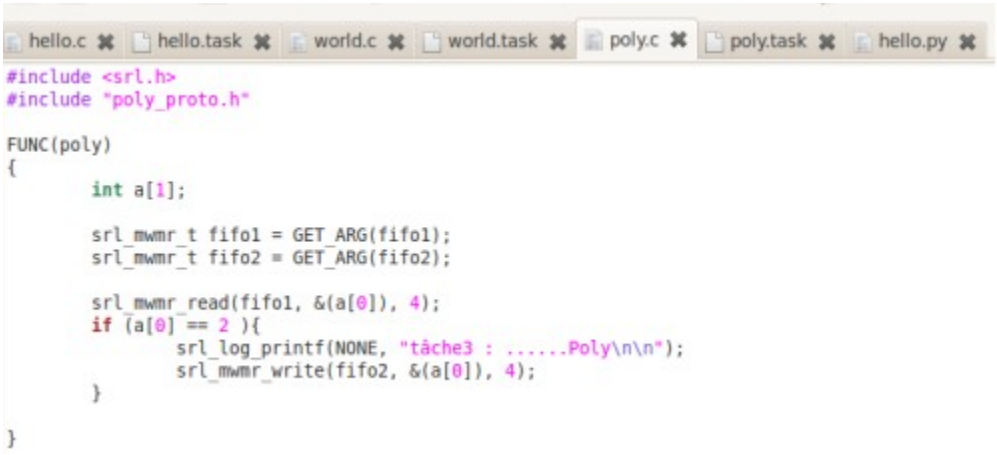
```
#include <srl.h>
#include "world_proto.h"

FUNC(world)
{
int a[1];
        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo1 = GET_ARG(fifo1);

        srl_mwmr_read(fifo0, &(a[0]), 4);
        if (a[0] == 1){
                srl_log_printf(NONE, "tâche2 : ... world...\n");
                a[0] = 2;
                srl_mwmr_write(fifo1, &(a[0]), 4);
        }
        else {
                a[0] = 2;
                srl_mwmr_write(fifo1, &(a[0]), 4);
        }
}
```

Khairi Slama 4 eme G informatique Groupe 2.2

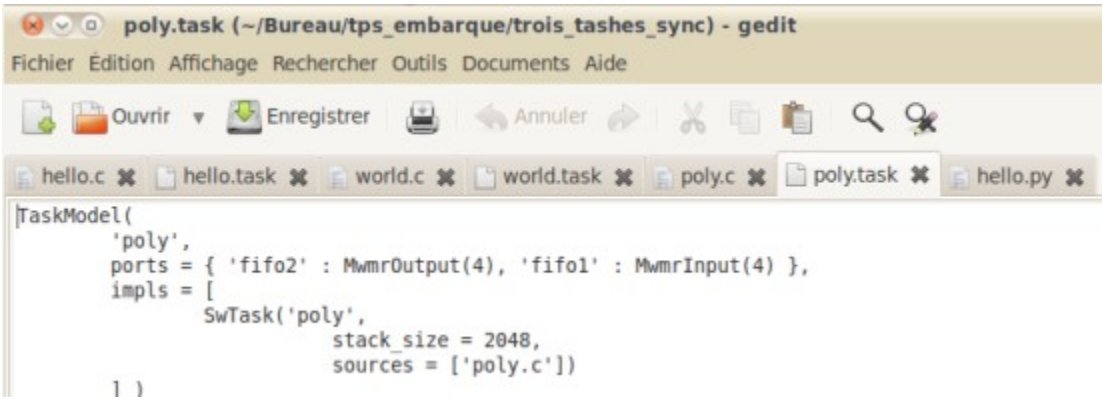
4) word.task



5) poly.c



6) poly.task

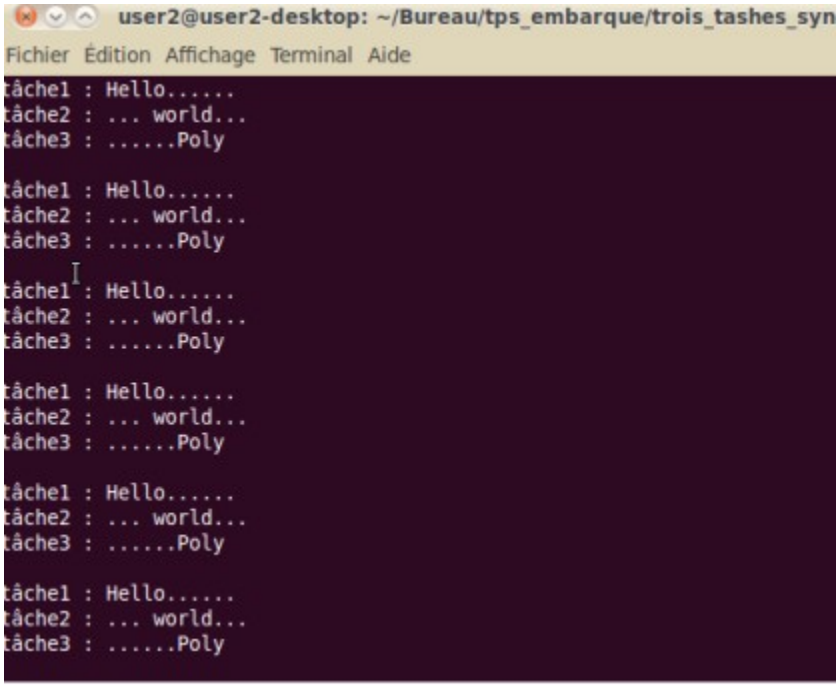


7) hello.py

hello.c | hello.task | world.c | world.task | poly.c | poly.task | hello.py

```
#!/usr/bin/env python
# coding: latin-1

import dsx
from dsx import *
from soclib import *

# Partie 1 : définition du TCG (Graphe des Tâches et des Communications)

fifo0 = dsx.Mwmr('fifo0', 4, 1)
fifo1 = dsx.Mwmr('fifo1', 4, 1)
fifo2 = dsx.Mwmr('fifo2', 4, 1)

tcg = dsx.Tcg(
        dsx.Task('hello', 'hello',
                {'fifo0':fifo0,'fifo2':fifo2} ),

        dsx.Task('world', 'world',
                {'fifo0':fifo0,'fifo1':fifo1} ),

        dsx.Task('poly', 'poly',
                {'fifo1':fifo1,'fifo2':fifo2} ),

# Partie 2 : génération du code exécutable sur station de travail POSIX

tcg.generate( dsx.Posix() )
```

8) exécution du programme

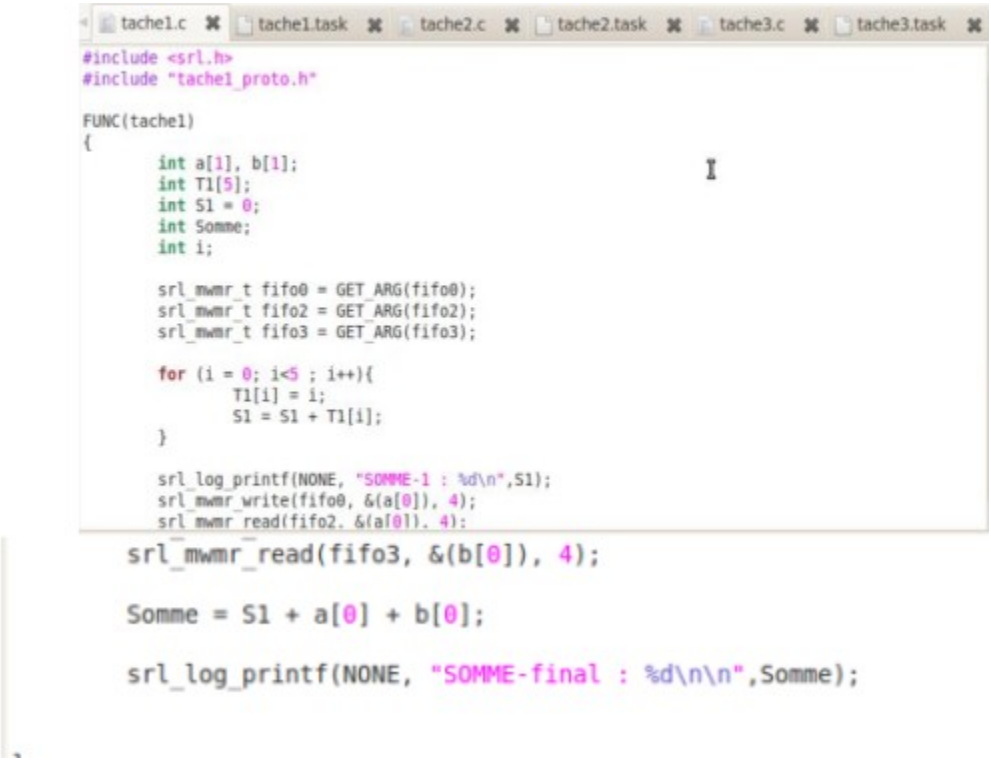


Khairi Slama 4 eme G informatique Groupe 2.2

Exemple 4 : Le calcule de la somme

Dans cet exemple on va calculer est afficher la somme de 3 entiers chaque entier est crèer par les taches exécuter voici le code suivant :

1) tache1.c



2) tache1.task

3) tache2.c

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
#include <srl.h>
#include "tache2_proto.h"

FUNC(tache2)
{
        int a[1];
        int S2[1];
        int i;
        int T2[5];

        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        S2[0] = 0;

        for (i = 0; i<5 ; i++){
                T2[i] = i;
                S2[0] += T2[i];
        }

        srl_mwmr_read(fifo0, &(a[0]), 4);

        srl_log_printf(NONE, "SOMME-2 : %d\n",S2[0]);

        srl_mwmr_write(fifo1, &(a[0]), 4);
        srl_mwmr_write(fifo2, &(S2[0]), 4);

}
```

4) tache2.task

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
TaskModel(
        'tache2',
        ports = { 'fifo0' : MwmrInput(4), 'fifo1' : MwmrOutput(4), 'fifo2' : MwmrOutput(4)},
        impls = [
                SwTask('tache2',
                        stack_size = 2048,
                        sources = ['tache2.c'])
        ] )
```

5) tache3.c

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

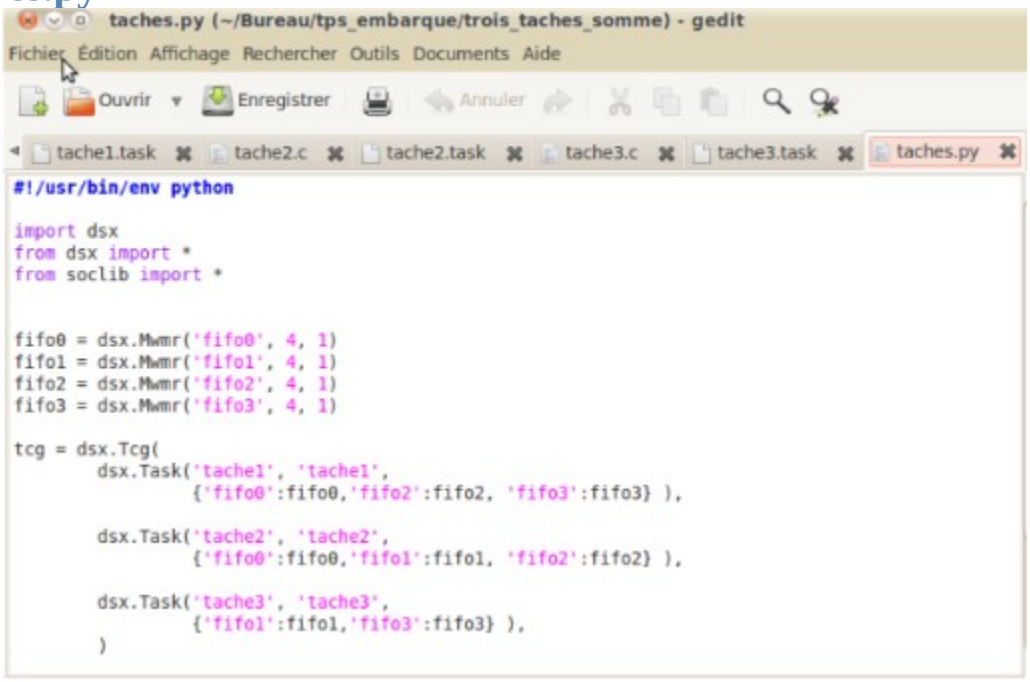
```
#include <srl.h>
#include "tache3_proto.h"

FUNC(tache3)
{
        int a[1];
        int i;
        int T3[5];
        int S3[1];

        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo3 = GET_ARG(fifo3);

        S3[0] = 0;

        for (i = 0; i<5 ; i++){
                T3[i] = i;
                S3[0] += T3[i];
        }

        srl_mwmr_read(fifo1, &(a[0]), 4);
        srl_log_printf(NONE, "SOMME-3 : %d\n",S3[0]);
        srl_mwmr_write(fifo3, &(S3[0]), 4);
```

6) tache3.task

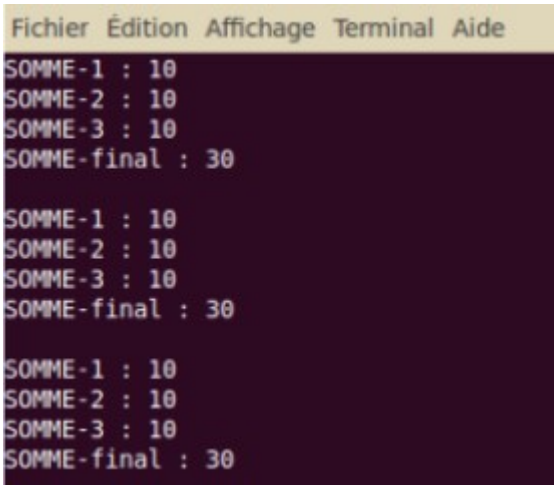
tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
TaskModel(
        'tache3',
        ports = { 'fifo3' : MwmrOutput(4), 'fifo1' : MwmrInput(4) },
        impls = [
                SwTask('tache3',
                        stack_size = 2048,
                        sources = ['tache3.c'])
        ] )
```

7) taches.py



8) exécution



Khairi Slama 4 eme G informatique Groupe 2.2

Exemple 5 : Le paraléllisme

Dans cet exemple on va exécuter les tache en mode parallèle comme le monter les figures suivante :

1) tache1.c

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
#include <srl.h>
#include "tache1_proto.h"

FUNC(tache1)
{
        int a[1];
        srl_mwmr_t fifo2 = GET_ARG(fifo2);
        srl_mwmr_t fifo0 = GET_ARG(fifo0);

        srl_log_printf(TRACE, "tache1 : Hello...\n");
        srl_mwmr_write(fifo0, &(a[0]), 4);
        srl_mwmr_read(fifo2, &(a[0]), 4);
}
```

2) tache1.task

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
TaskModel(
        'tache1',
        ports = { 'fifo0' : MwmrOutput(4), 'fifo2' : MwmrInput(4) },
        impls = [
                SwTask('tache1',
                        stack_size = 2048,
                        sources = ['tache1.c'])
        ] )
```

3) tache2.c

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
#include <srl.h>
#include "tache2_proto.h"

FUNC(tache2)
{
        int a[1];
        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo1 = GET_ARG(fifo1);

        srl_mwmr_read(fifo0, &(a[0]), 4);
        srl_log_printf(TRACE, "tache2 : ... world\n\n");
        srl_mwmr_write(fifo1, &(a[0]), 4);
}
```

4) tache2.task

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
TaskModel(
        'tache2',
        ports = { 'fifo0' : MwmrInput(4), 'fifo1' : MwmrOutput(4)},
        impls = [
                SwTask('tache2',
                        stack_size = 2048,
                        sources = ['tache2.c'])
        ] )
```

5) tache3.c

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
#include <srl.h>
#include "tache3_proto.h"

FUNC(tache3)
{
        int a[1];
        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        srl_mwmr_read(fifo1, &(a[0]), 4);
        srl_log_printf(TRACE, "tache3 : ... polytech\n\n");
        srl_mwmr_write(fifo2, &(a[0]), 4);
}
```

6) tache3.task

tache1.c | tache1.task | tache2.c | tache2.task | tache3.c | tache3.task

```
TaskModel(
        'tache3',
        ports = { 'fifo1' : MwmrInput(4), 'fifo2' : MwmrOutput(4)},
        impls = [
                SwTask('tache3',
                        stack_size = 2048,
                        sources = ['tache3.c'])
        ] )
```

7) taches.py

tache1.task | tache2.c | tache2.task | tache3.c | tache3.task | taches.py

```
#!/usr/bin/env python

import dsx
from dsx import *
from soclib import *


fifo0 = dsx.Mwmr('fifo0', 4, 1)
fifo1 = dsx.Mwmr('fifo1', 4, 1)
fifo2 = dsx.Mwmr('fifo2', 4, 1)

tcg = dsx.Tcg(
        dsx.Task('tache1', 'tache1',
                {'fifo0':fifo0,'fifo2':fifo2} ),

        dsx.Task('tache2', 'tache2',
                {'fifo0':fifo0,'fifo1':fifo1} ),

        dsx.Task('tache3', 'tache3',
                {'fifo1':fifo1,'fifo2':fifo2} ),
        )
```

8) le mapping

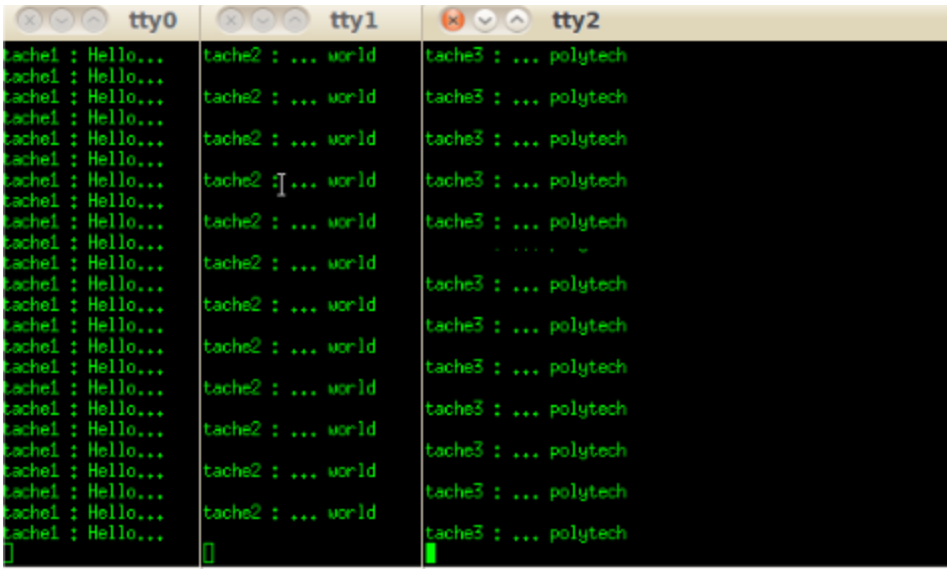
```
#########################################################
# Section B : Hardware architecture
#
# The file containing the architecture definition
# must be included, and the path to the directory
# containing this file must be defined
#########################################################

from vgmn_noirq_mono import VgmnNoirqMono

archi = VgmnNoirqMono(ntty = 3)

#########################################################
# Section C : Mapping
#
#########################################################

m = Mapper(archi, tcg)
```

```
# mapping the MWMR channel

m.map( "fifo0",  buffer  = "cram0",  status  = "cram0",  desc    = "cram0")
m.map( "fifo1",  buffer  = "cram0",  status  = "cram0",  desc    = "cram0")
m.map( "fifo2",  buffer  = "cram0",  status  = "cram0",  desc    = "cram0")

m.map("tache1",
      desc = 'cram0',
      run = 'cpu0',
      stack = 'cram0',
      tty = 'tty',
      tty_no = 0)

m.map("tache2",
      desc = 'cram0',
      run = 'cpu1',
      stack = 'cram0',
      tty = 'tty',
      tty_no = 1)

m.map("tache3",
      desc = 'cram0',
      run = 'cpu2',
      stack = 'cram0',
      tty = 'tty',
      tty_no = 2)
```

```
# mapping the software objects associated to a processor

m.map( 'cpu0',
  private    = "cram0",
  shared   = "cram0")

m.map( 'cpu1',
  private    = "cram0",
  shared   = "cram0")

m.map( 'cpu2',
  private    = "cram0",
  shared   = "cram0")

# mapping the software objects used by the embedded OS
```

```
m.map(tcg,
  private  = "cram0",
 shared  = "uram0",
  code    = "cram0",
  tty = "tty",
  tty_no = 0)
```

9) exécution



dans cette exemple le code va ouvrir 3 fenêtre nommée tty0, tty1, tty2 chaque fenêtre va afficher un message envoyer par une task.