# Travaux Pratiques Systèmes d'Exploitation
# longjmp/setjmp

## 1 Premier exemple de longjmp et setjmp

```
#include <setjmp.h>
#include <stdio.h>
/*
 * Testez le programme avec:
 * > nomexecutable
 * > nomexecutable x
 * > nomexecutable x x
 */
static jmp_buf env;
static void f2(void)
{
   longjmp(env,2);
}
static void f1(int val)
{
if(val == 1){
   longjmp(env,1);
}
f2();
}

int main(int argc, char** argv)
{
switch(setjmp(env))
{

case 0:
/* Nous sommes ici car on vient de mettre en place la cible*/
printf("cible en place !\n");
f1(argc);
printf("Ce texte ne sera jamais affiche !");
break;
```

```
case 1:
/*Nous sommes ici car on vient d'un jump avec comme valeur 1*/
printf("saut depuis f1 ! \n");
break;

case 2:
/*Nous sommes ici car on vient d'un jump avec comme valeur 2*/
printf("saut depuis f2 ! \n");
break;
}
}
```

- Lisez attentivement l'exemple introLongJmp.c.
- Compilez et exécutez le programme en essyant de passer un nombre différent de paramètres
- Illustrez à l'aide un schéma l'évolution de la pile dans les cas suivants:
  - on passe 0 argument au programme
  - on passe 1 argument au programme
  - on passe 2 arguments au programme

# 2 Droit au but !

```
#include <stdlib.h>     /* atol */
#include <stdio.h> /*fprintf*/

static int recEstPremier(long valeur, long diviseur)
{
  if(diviseur==1)
    {
      return 1;
    }

  if(valeur%diviseur == 0)
    {
      return 0;
    }
    else
    {
      int reponse = recEstPremier(valeur, diviseur-1);
      return reponse;
    }
}
```

```
static int estPremier(long valeur)
{
  if(valeur == 1)
    {
      return 1;
    }
  else
    {
      return recEstPremier(valeur, valeur-1);
    }
}

int main(int argc, char** argv)
{
  if(argc!=2)
    {
      printf("Usage %s <entier>",argv[0]);
      exit(-1);
    }

  long valeur = atol(argv[1]);
  if(valeur==0)
    {
      printf("Usage %s <entier !=0>",argv[0]);
    }
  valeur = (valeur<0)?-valeur:valeur;
  fprintf(stdout,"estPremier ?: %d\n",estPremier(valeur));
}
```

Le programme estPremier.c propose un algorithme naif qui vérifie si un nombre passé en paramètre est premier. Nous testons simplement que tous les nombres inférieurs jusqu'à 1 ne divisent pas ce nombre. En utilisant uniquement longjmp/setjmp pouvez-vous améliorer les performances de ce programme? [1] (Indication: une fois la réponse est trouvée, il est inutile de dépiler toutes les frames des appels récursifs)

# 3 Les co-routines

Les co-routines sont les ancêtres des threads. Il s'agit de fonctions qui vont mutuellement et volontairement se passer le contrôle durant leurs exécutions. Nous proposons de réaliser des co-routines simples en utilisant longjmp/setjmp.

```
#include  <stdio.h>
#include  <setjmp.h>
```

[1] 169151 est un nombre premier, vous pouvez l'utiliser pour tester les performances. Pour avoir le temps d'exécution d'un processus on peut utiliser la commande à partir du shell 'time'

```
int       max_tour;
int       compteurPing;
//??

jmp_buf   ancreMain;
jmp_buf   ancrePing;
//??

void      Ping(void);
void      Pong(void);

void  main(int  argc, char* argv[])
{
     if (argc != 2) {
          printf("Usage %s <nombre-iteraction>\n", argv[0]);
          exit(1);
     }
     max_tour = abs(atoi(argv[1]));
     compteurPing = 1;

     printf("\t((Main 0))\n");

     if(setjmp(ancreMain) == 0)
       {
    //create de l'ancre
    Ping();
       }
     printf("\t((Main 1))\n");
     longjmp(ancrePing, 1);

}
void  Ping(void)
{
  printf("Initialisation de Ping\n");
  if (setjmp(ancrePing) == 0)
    {//creation de l'ancre
      longjmp(ancreMain, 1);// retour au main
    }
  // ici nous venons d un jump !
  while (1)
    {
      printf("%3d:\t<<Ping>\n", compteurPing);
      compteurPing++;
      if (compteurPing > max_tour) {exit(0);}
      if (setjmp(ancrePing) == 0)
```

```
   { // creation de l'ancre
     longjmp(ancreMain, 1);   //on cede le controle a la
    fonction main
   }
    }
}

void  Pong(void)
{
  //??
  printf(">>Pong<<\n");   //
  //??
}
```

Le programme coroutine-student.c propose un premier exemple de coroutine. La fonction main et la fonction ping se passent mutuellement le contrôle.
Voici une trace d'exécution:

```
>./coroutine-student 3
((Main 0))
Initialisation de Ping
((Main 1))
  1: <<Ping>
((Main 1))
  2: <<Ping>
((Main 1))
  3: <<Ping>
```

Nous voyons bien qu'alternativement la fonction main affiche ((Main 1)) et la fonction ping affiche Ping.

## 3.1 Etape 1

Relisez attentivement le code du programme pour comprendre les mécanismes mis en place.

## 3.2 Etape 2

Nous ne souhaitons plus avoir une alternance entre la fonction main et la fonction ping mais entre la fonction ping et la fonction pong (la fonction main va service simplement à initialiser le système)

Voici à titre d'exemple la trace souhaitée:

```
>./coroutine-pingpong 3
((Main 0))
Initialisation de Ping
Initialisation de Pong
((Main 1))
```

```
1: <<Ping>>
1: >>Pong<<
2: <<Ping>>
2: >>Pong<<
3: <<Ping>>
```

Modifiez le programme pour avoir ce résultat

### 3.3 Etape 3

Dans le programme les coroutines ping et pong utilisent des variables globales compteurPing et compteurPong. Que se passera-t-il si ces variables n'étaient plus globales mais des variables automatiques des fonctions ping et pong respectivement ?