TP Communication entre Processus : Les Signaux
Environnement Linux, Raspbian

Paramétrer un signal USR (signal laissé libre pour les programmeurs)

Voici un exemple de processus que l'on va programmer pour réagir au SIGUSR1 (les parties concernant les signaux sont en gras) :

Programme *principal.c* :

```
#include <stdio.h>
#include <unistd.h> // pour sleep()
#include <stdlib.h> // pour system()
#include <signal.h>

int cpt=0;

////////////////////////////////////////////////////////////
// Fonction pour ecrire a l'ecran à une position X,Y precise
#define ESC 27;
void ecrire_xy(int x, int y, char *chaine)
{
      char esc=ESC;
      printf("%c[%d;%dH%s",esc,y,x,chaine);fflush(stdout);
}

////////////////////////////////////////////////////////////
// La fonction executee à reception du signal
void moncode(int s)
{
      cpt++;
}


////////////////////////////////////////////////////////////////////
//         P R O G R A M M E     P R I N C I P A L                //
////////////////////////////////////////////////////////////////////
int main(int argc, char*argv[])
{
      char chaine[20];
      int x=12;
      int y=10;
      struct sigaction alrm;

      // Mise en place du comportement a reception du SIGUSR1
      sigaction(SIGUSR1, NULL, &alrm);
      alrm.sa_handler=moncode;
      sigaction(SIGUSR1, &alrm, NULL);


      system("clear");
      while(cpt < 100) {
            sprintf(chaine,"(%d,%d)[%d]%d ", x,y,getpid(), cpt);
            ecrire_xy(x,y, chaine);
            sleep(1);
      }
      ecrire_xy(x,y,"                                                    ");
      return 0;
}
```

Le **signal** reçu se comporte comme une « interruption » : Le programme est certainement dans sa boucle *while.* Lorsque le signal arrive, il s'interrompt, exécute la routine *moncode* puis retourne à l'exécution de la boucle. Il reprend à l'instruction qui suit celle qu'il exécutait lorsqu'il a été interrompu.

Travail : Créez le programme ci-dessus et testez le :
Procédure de test : dans 1 fenêtre ssh le programme s'exécute
Dans une autre fenêtre ssh, envoyer le signal SGUSR1 au processus avec la commande kill
Ex : Si le PID de votre processus est 18520,
Tapez la commande : `kill –SIGUSR1 18520`
Observez le résultat sur le programme.

Modifier le Code Existant :

Ajouter la gestion du SIGUSR2 pour qu'il se passe une remise à zéro du compteur sur réception de SIGUSR2

Créer un client pour le kill

Vous venez de tester la commande ***kill*** qui envoi des signaux aux processus (Start, stop, interrupt, kill, …). En programmation C il est aussi possible d'envoyer des signaux par la commande ***kill*** de la librairie *signal.h.*

Ce dispositif existe sur tous les Linux et s'appelle *signaux Posix*.

Exemple : Le programme *test_signal.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>


int main(int argc, char **argv)
{

	int pid;

	// Le pid est saisi par la ligne de commande
	if ( argc < 2 )
	{
		printf("Entrez le pid\n"); return 1;
	}

	pid = atoi( argv[1] );
	if ( pid == 0 )
	{
		printf("Entrez un pid valide\n"); return 1;
	}

	kill(pid, SIGUSR1); //Envoi du signal SIGUSR1
}
```

Dans cet exemple, on envoie le SIGUSR1 au processus dont le PID a été saisi en argument de la commande :
Exemple : `./test_signal 6988`
Il faut bien sûr que le numéro de PID corresponde à un processus, et que ce processus sache gérer le SIGUSR1.

Testez ce programme et adaptez-le pour qu'on ajoute le choix de la remise à zéro du compteur.