Introduction à la programmation d'applications temps réel sous Linux : un exemple en mesure et contrôle numérique

Peter Wurmsdobler
Centre de Transfert des Microtechniques
peterw@ctm-france.com

Plan de l'exposé

- Circonstances et motivation
- Systèmes d'exploitation temps réel
- Réalisations temps réel sous Linux
- RTLinux, une version temps réel sous Linux
- Application RTLinux en mesure et contrôle numérique
- Résultats et visions

Circonstances

- Développement d'un banc de mesure, Micro-couple-mètre dans le cadre du projet Européen HAFAM
- Matériel existant et ses performances:
  - une partie mécanique avec capteur de couple
  - une partie électronique avec moteur d'entraînement
  - ordinateur 486/33/4MB-RAM sous DOS/Win3.11
  - carte de mesure : PCL818, cadence de mesure : 100Hz
- EXIGENCES:
  Multiplier la vitesse par 10, mais en plus sophistiqué

Premières démarches

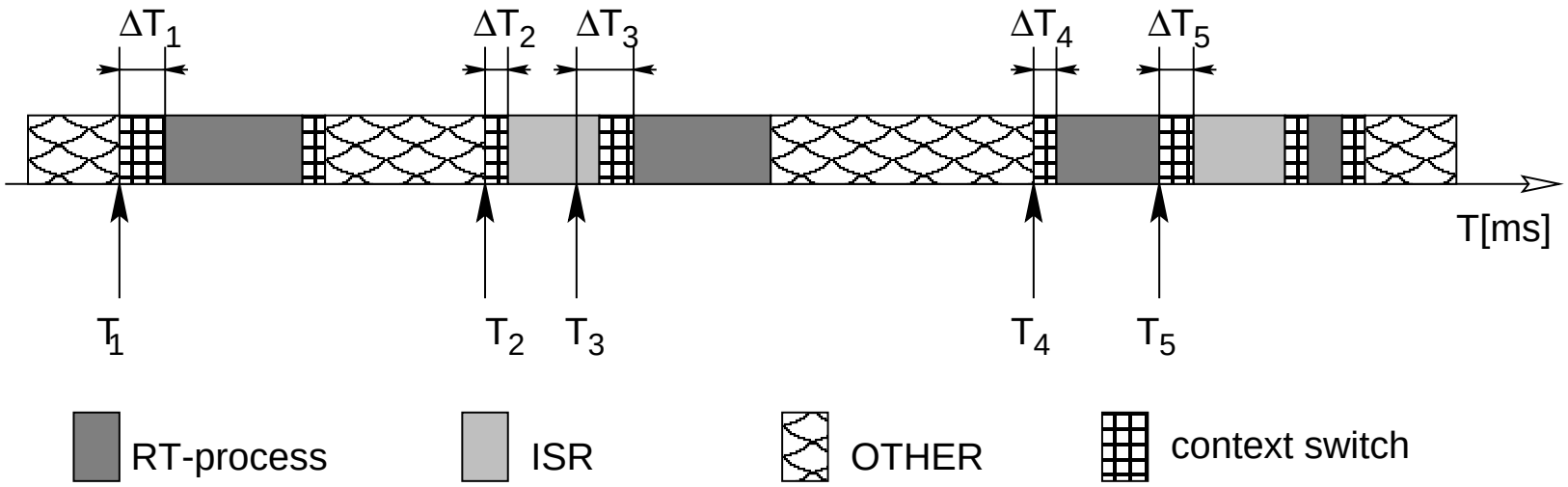
- Le logiciel propriétaire existant n'est pas satisfaisant
- La recherche d'un autre logiciel, évidemment sous Windows
- Ajouter 16MB-RAM et une carte de réseau pour préparer la mise à jour
- En cours d'installation de Windows 95 s'affichait de manière répétitive le "bluescreen"
- Windows 98 ne peut pas être installé sur un 486/33
- Une solution propriétaire n'est pas possible

Conséquences

- Rechercher une meilleure solution sur internet
- Se joindre aux forums linuxiens et discuter des solutions
- Convaincre les responsables d'utiliser une solution basée sur Linux et les sources libres : RTLinux
- Installer Linux et RTLinux, apprendre le C et GTK+
- Programmer le logiciel pour le micro-couple-mètre

Quelques notions de temps réel

Chaque évènement est toujours traité avec un certain retard :



Système d'exploitation temps réel prend en compte les délais !

Plusieurs types de temps réel

Temps réel dur (hard real time) :

Tous les délais sont absolument respectés et tous les retards sont inférieurs à une tolérance prédéfinie

Temps réel mou (soft real time) :

La moyenne de retards est inférieure à une tolérance prédéfinie sachant que quelques délais ne sont pas respectés

Temps réel ferme (firm real time) :

Dans un sens statistique tous les délais sont respectés avec une variance de retard prédéfinie.

Est-ce que Linux est un systeme temps réel ?

NON, le noyau de Linux n'est pas prévu pour le temps réel, car :

- "no kernel pre-emption"
- "scheduling time" augmente avec le nombre de processus
- "interrupt disable" est présent pour de longues périodes
- "bottom half" est quasiment un processus "haute priorité"
- "thread" n'est pas supporté par le noyau

Les techniques pour transformer Linux en temps réel

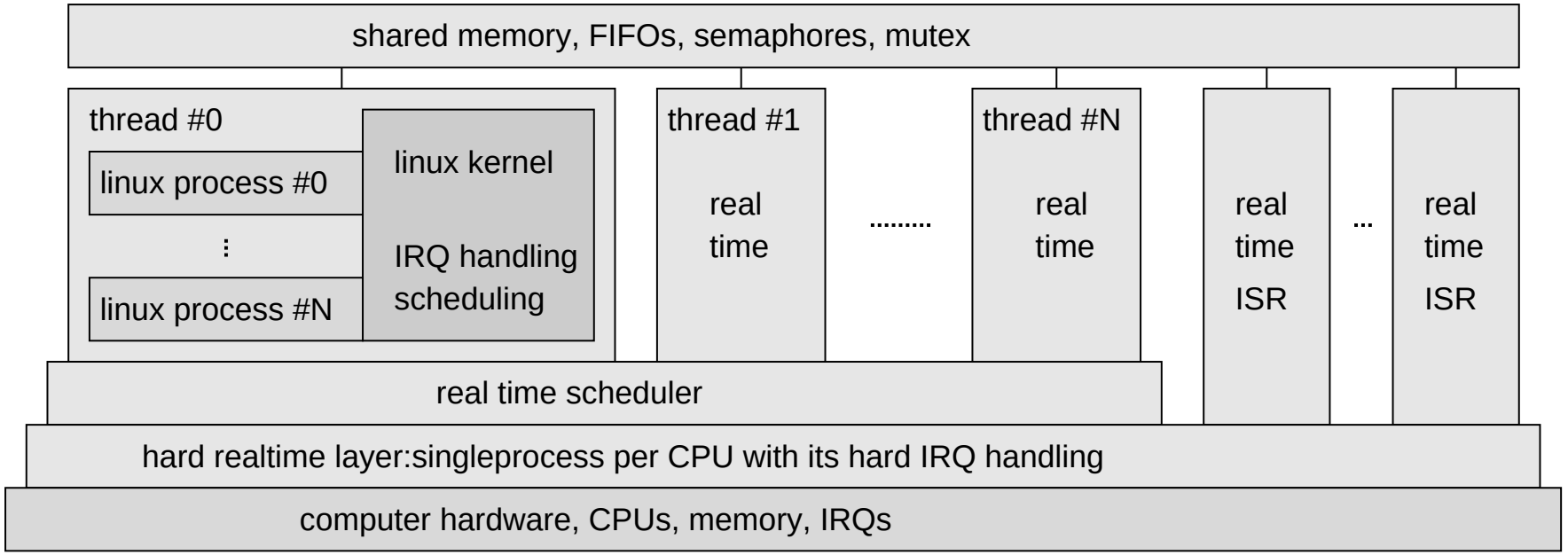
1. Le remplacement du noyau :
   LynxOS, OnCore, etc. offrent le temps réel dur ou mou
2. La modification du noyau :
   Linux/RK, KURT, etc. offrent le temps réel mou ou ferme
3. La coexistence avec un micro-noyau temps réel :
   RTLinux, RTAI, etc. offrent le temps réel dur

Les principes, astuces de RTLinux

1. Le système d'exploitation Linux est un "idle thread" parmi d'autres "threads" pour un système exécutif en temps réel
2. L'interception et gestion de toutes les interruptions et le traitement des interruptions prévues en temps réel
3. Le système d'exploitation temps réel passe les autres interruptions à Linux qui opère sur des interruptions molles.

Architecture de RTLinux

Linux comme "idle thread" parmi d'autres "threads" et parmi des procédures de service d'interruptions :



Interface de programmation RTLinux, I

```
int rtl_request_global_irq(unsigned int irq,
  unsigned int (*handler)(unsigned int, struct pt_regs *));
void rtl_hard_enable_irq(unsigned int ix);
void rtl_hard_disable_irq(unsigned int ix);
int rtl_free_global_irq(unsigned int irq);

int rtf_create(unsigned int fifo, int size);
int rtf_create_handler(unsigned int fifo,
  int (*handler)(unsigned int fifo) ;
int rtf_put(unsigned int fifo, void * buf, int count ):
int rtf_get(unsigned int fifo, void * buf, int count );
int rtf_destroy(unsigned int fifo);

int shm_allocate(const char *name, unsigned int size, void **shm);
int shm_deallocate(void * shm);
```

Interface de programmation RTLinux, II

```
typedef long long hrtime_t;
hrtime_t gethrtime(void);

typedef RTL_THREAD_STRUCT *pthread_t;
int pthread_create (pthread_t *thread, pthread_attr_t *attr,
  void *(*start_routine)(void *), void *arg);
int pthread_make_periodic_np (pthread_t p,
  hrtime_t start_time, hrtime_t period);
int pthread_suspend_np (pthread_t thread);
int pthread_wakeup_np (pthread_t thread);
int pthread_wait_np(void);
void pthread_exit(void *retval);
int pthread_delete_np (pthread_t thread);
```

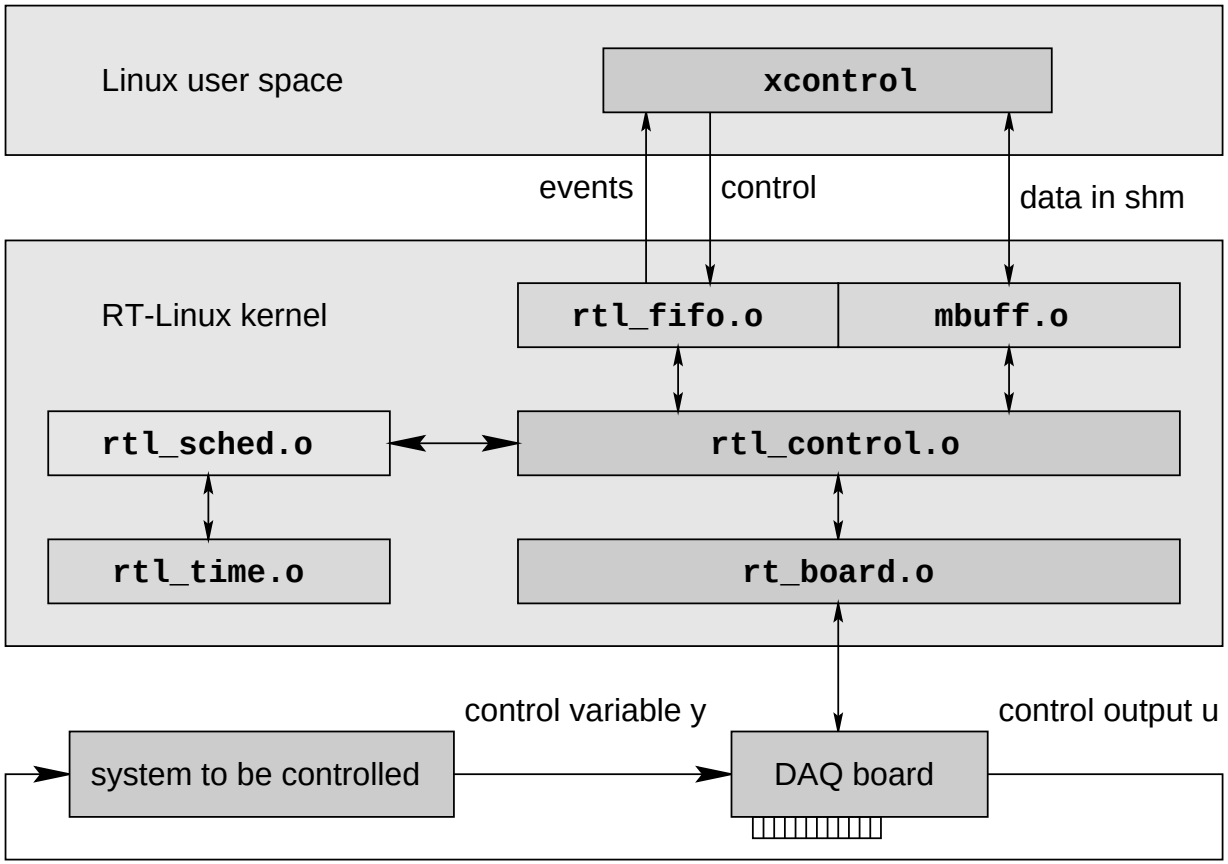
L'application de contrôle numérique

... est coupée en trois morceaux :

- `xcontrol`, une interface graphique sous X-Windows, basée sur GTK+ et GtkFigure
- `rtl_control.o`, le module qui contient le thread temps réel sous la version rtl-2.0 et le noyau 2.2.14
- `rtl_board.o`, une compilation de fonctions pour accéder à une carte de mesure

avec son architecture ...

Architecture en plusieurs niveaux



Mémoire partagée (shared memory)

```
#define SHM_DEV_FILE  ( "/dev/mbuff" )
#define SHM_NAME    ( "control" )
#define SHM_SIZE    ( sizeof(shm_t) )
typedef struct
  {
  unsigned int N;         /* length of measured vectors */
  unsigned short int w;       /* {0,65535}, setpoint */
  unsigned short int u[SAMPLES]; /* {0,65535}, control current */
  unsigned short int y[SAMPLES]; /* {0,65535}, measured position */
  int a[LENGTH];          /* control denominator of ORDER+1 */
  int b[LENGTH];          /* control numerator of ORDER+1 */
  }
  shm_t;
```

Tampons (FIFOs) et leurs messages

```
#define FIFO_SIZE        (5000)   /* byte size for fifo buffer      */
#define CONTROL_FIFO     (0)      /* passing messages to rt_control */
#define CONTROL_FILE     "/dev/rtf0"
#define EVENT_FIFO       (1)      /* triggering events in Xcontrol  */
#define EVENT_FILE       "/dev/rtf1"
#define START_CONTROL    ('a')    /* xcontrol -> rtl_control        */
#define STOP_CONTROL     ('b')    /* xcontrol -> rtl_control        */
#define TRIGGER_MEASURE  ('c')    /* xcontrol -> rtl_control        */
#define MEASURE_READY    ('a')    /* rtl_control -> xcontrol        */
#define INVALID_MESSAGE  ('b')    /* rtl_control -> xcontrol        */
```

"thread" de contrôle et ses variables

```
pthread_t control;            /* THE control thread, or task */
static int measure;           /* flag if measurment is on    */
static shm_t *shm;            /* shared memory (sic!)        */
static unsigned int index;    /* index in shared memory      */
static int e[LENGTH];         /* control deviation buffer    */
static int u[LENGTH];         /* control signal buffer       */
static unsigned int k;        /* index k of actual value     */
static unsigned short int Y; /* THE measured value, input   */
static unsigned short int U; /* THE control current, output */
static unsigned short int W; /* THE setpoint                */
```

La fonction de contrôle, I

Cette partie est appelée lors de la création du "thread" :

```
void *rt_control_function( void *t )
  {
  unsigned char message = MEASURE_READY;
  unsigned int l,h;

  while(1) /* this is the periodic part */
    {
    pthread_wait_np(); /* yield to scheduler */

    /* AND HERE STARTS THE CONTROL ALGORITHM */
```

La fonction de contrôle, II

```
Y = rt_board_aget();

e[k] = (int) ( W - Y ); /* this is the control deviation ! */

u[k] = (shm->b[0]) * e[k];
for ( l=1; l<LENGTH; l++ )
  {
  h = ( LENGTH + k - l ) % LENGTH;
  u[k] += ( (shm->b[l]) * e[h] - (shm->a[l]) * u[h] );
  }
u[k] /= (shm->a[0]); k = (k+1) % LENGTH;

U = (unsigned short int)( u[k] + 32768 );

rt_board_aset( OUTPUT_CHANNEL, U );
```

La fonction de contrôle, III

Si la trace d'une réponse du système est souhaitée :

```
    if (measure == 0)
      {
      shm->y[index] = Y;
      shm->u[index] = U;
      index++;
      if (index == (shm->N) )
        {
        rtf_put( EVENT_FIFO, &message, 1 );
        measure = -1;
        }
      }
    }
}
```

La réaction aux messages

```
static int rt_control_message_handler(unsigned int fifo) {
  unsigned char message;
  while( rtf_get( fifo, &message, 1 ) > 0) {
    switch(message) {
      case START_CONTROL:
        pthread_make_periodic_np( control, gethrtime(), (TS*1000) ); break;
      case STOP_CONTROL:
        pthread_make_periodic_np( control, HRTIME_INFINITY, 0 ); break;
      case TRIGGER_MEASURE:
        index = 0; measure = 0; break;
      default:
        rtf_put( EVENT_FIFO, &message, 1 ); }
     }
  return 0;
  }
```

Initialisation du module temps réel

Dans `init_module` et `rtl_control.o`

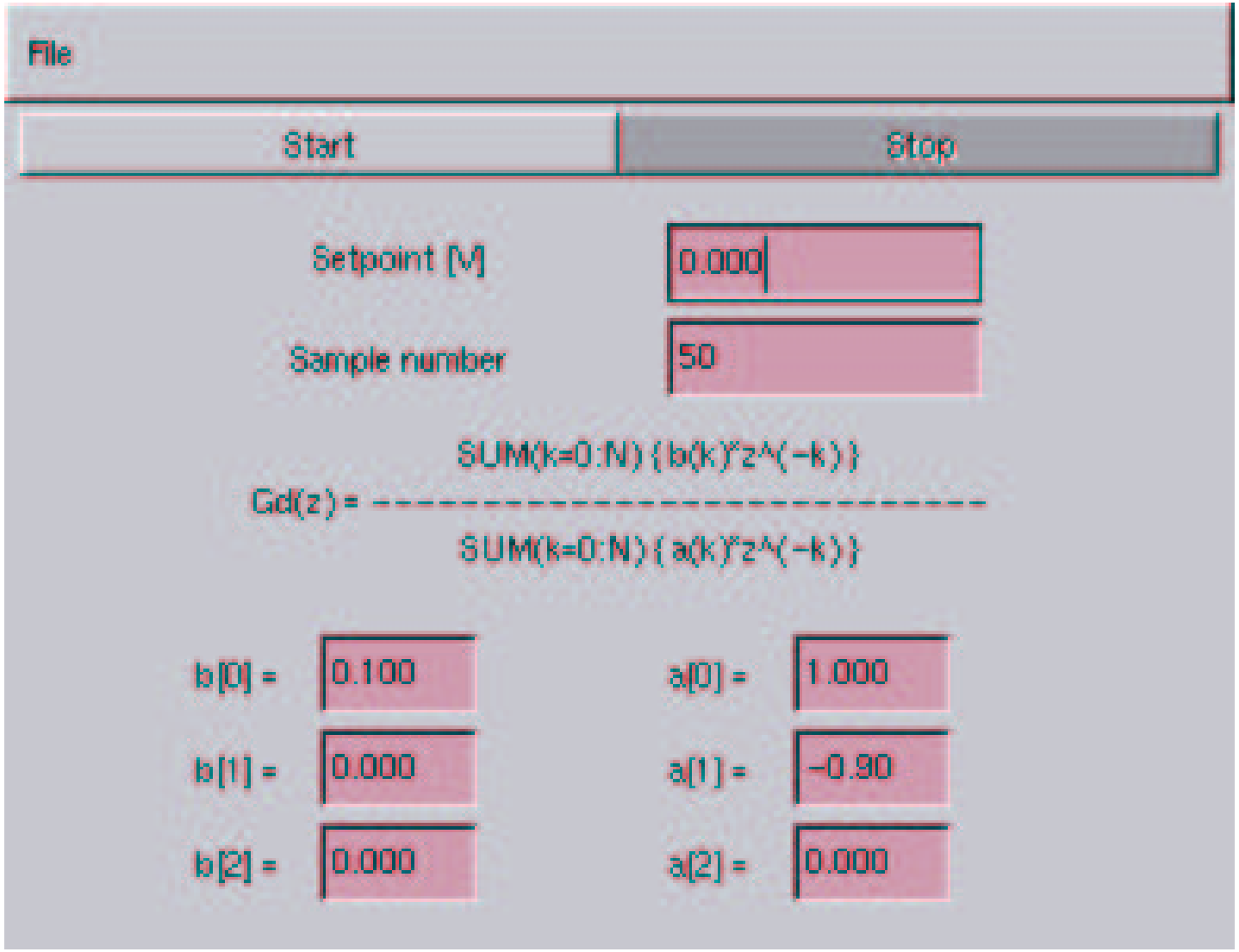
- créer FIFOs "control" et "events" pour les messages
- initialiser "message handlers" pour "control FIFO"
- initialiser "shared memory" avec `mbuff`
- créer le "thread" de contrôle

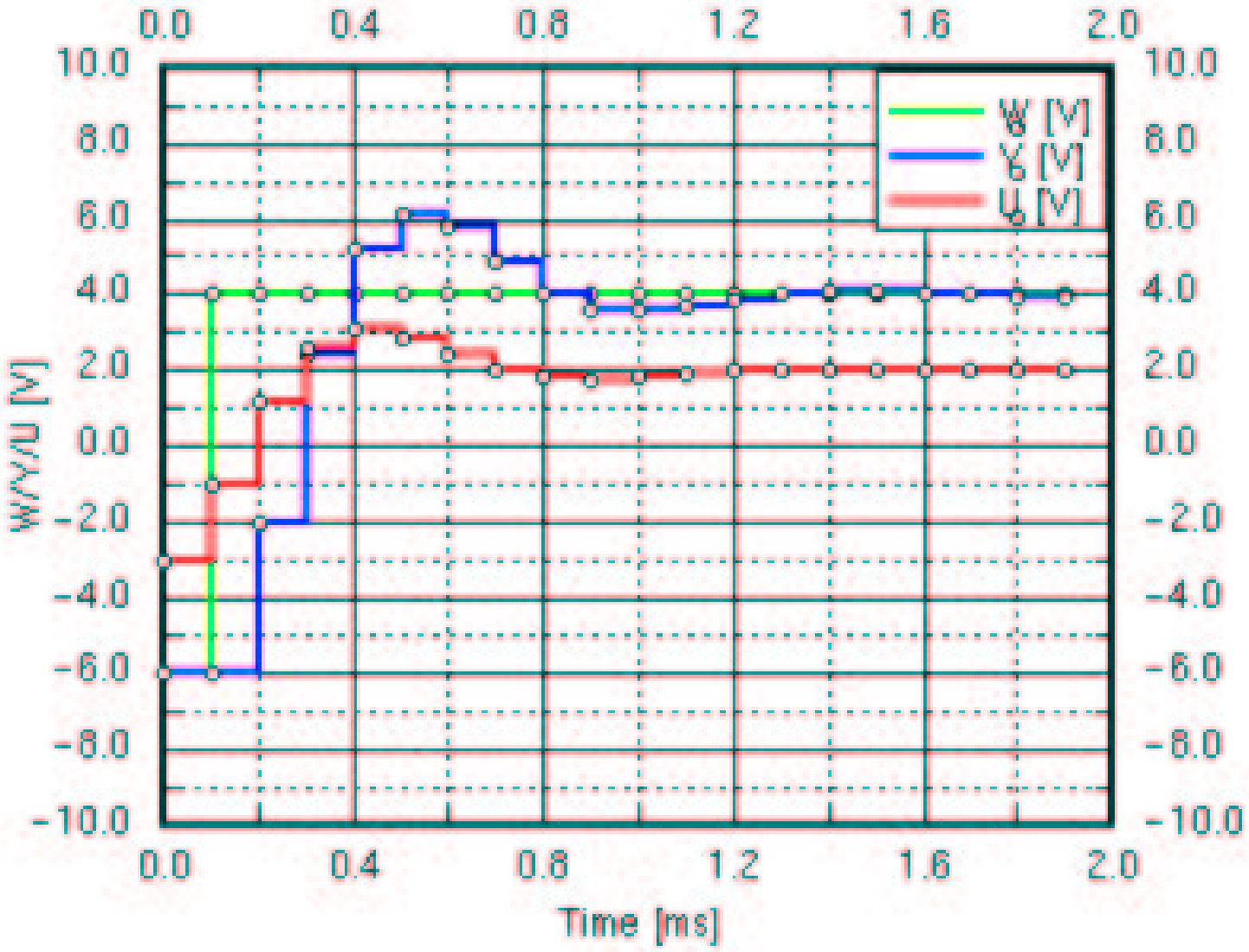
Après son insertion, le module attend le message pour lancer le "thread" par `rt_control_start()`

Application pour l'utilisateur

L'application et l'interface graphique sous GTK+ :

- ouvrir les FIFOs pour lire (events) et écrire (control)
- "maps shared memory onto its memory space"
- initialiser les objets GTK+, fenêtre, figures, etc.
- offrir la possibilité d'entrer des paramètres
- offrir la possibilité de lancer le "thread" de contrôle.
- offrir la possibilité de tracer une réponse du système


File
Start
Stop
Setpoint [V]
0.000
Sample number
50
SUM(k=0:N){b(k)*z^(-k)}
Gd(z) = ----------------------------
SUM(k=0:N){a(k)*z^(-k)}
b[0] =
0.100
b[1] =
0.000
b[2] =
0.000
a[0] =
1.000
a[1] =
-0.90
a[2] =
0.000


0.0
0.4
0.8
1.2
1.6
2.0
10.0
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
-10.0
W/V/U [V]
Time [ms]
W [V]
V [V]
U [V]

Performance avec un processeur Intel P233

- La cadence de "thread" à $100\,\mu$s, cad 10 kHz n'influence pas le système Linux
- La limite avec une carte de mesure BMC1000 : $22\,\mu$s, due à un temps de conversion de $10\,\mu$s
- L'incertitudes de l'ordonnancement (jitter) dans l'ordre de quelques μs, "worst case" $10\,\mu$s

Une réduction de "jitter" est possible avec une technique dite "sliding".

La vision du temps réels sous Linux

- Le système d'exploitation temps réel dur est disponible pour : i386, PPC, MIPS et AlphaAXP
- Une adaptation à MK68, ARM7, StrongARM, etc. en cours
- Un support gratuit et rapide par internet `www.rtlinux.org`
- Un livre : Real Time Linux de Philip N. Daly
- L'ENTREE pour Linux temps réels:

`http://www.realtimelinuxfoundation.org`