Le temps réel dans la conception et le développement logiciel

Mise en œuvre de systèmes temporellement contraints

— Travaux pratiques —

Loïc PLASSART, PhD
loic.plassart@altran.com

Altran Ouest
Technopôle Brest-Iroise - Site du Vernis - Rue Pierre Rivoallon
29238 Brest Cedex

28 janvier 2013

Accès au compilateur croisé

- Déclaration et initialisation de la variable d'environnement

```
PATH=<chemin_chaine_compilation_armadeus>/usr/bin:$PATH
export PATH
```

Utilisation

- Visualisation de la version installée
  `arm-linux-gcc -v`
- Compilation d'un programme
  `arm-linux-gcc -Wall -o <binaire> <source>.c`

Vérification

- Analyse du binaire produit
  `file <binaire>`

Différents moyens

- Commande `scp` depuis la machine hôte
  `scp <fichier_a_transferer> root@<ip_cible>:/root`
- Commande `tftp` depuis la cible
  `tftp -g -r <fichier_a_transferer> <ip_serveur>`

2. Codage et compilation d'un thread périodique
   - Expérimentation à partir d'un code fourni
   - Travail demandé

Objectifs visés

- Mise en œuvre d'une chaîne de compilation croisée
- Chargement d'un applicatif sur une cible
- Exploitation des appels système POSIX
- Analyse de l'exécution du thread
- Évaluation de la performance globale du programme

Code (1/4)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <sys/time.h>

#define SIGNAL SIGALRM
#define TIMER ITIMER_REAL
#define PERIODE 10000
#define PRIORITE 50
#define DISCIPLINE SCHED_RR
#define DUREE 10

int latence_max = 0;
int avance_max = 0;
```


Code (2/4)

```
/* Gestion du signal d'interruption */
void interruption(int cause, siginfo_t *HowCome, void *ptr)
{
  unsigned long long valeur_precedente, valeur_courante;
  int delai;
  static int flag = 0;
  static struct timeval date;

  /* Initialisation de l'horadatage */
  if (flag == 0)
  {
    gettimeofday(&date, NULL);
    flag = 1;
  }
  else
  {
    valeur_precedente = date.tv_sec * 1000000 + date.tv_usec;
    gettimeofday(&date, NULL);
    valeur_courante = date.tv_sec * 1000000 + date.tv_usec;
    delai = (int)(valeur_courante - valeur_precedente);
    printf("%d microsec.\n", delai);

    if (delai - PERIODE > latence_max)
      latence_max =  delai - PERIODE;
```

Code (3/4)

```
      if (PERIODE - delai > avance_max)
        avance_max = PERIODE - delai;
    }
  }


  /* Fonction du thread */
  void *thread(void *nom)
  {
    struct sched_param param;
    struct itimerval tempo;
    struct sigaction action;

    /* Déclaration du signal */
    action.sa_sigaction = interruption;
    sigemptyset(&action.sa_mask);
    action.sa_flags = SA_SIGINFO;

    if (sigaction(SIGNAL, &action, 0))
    {
      perror("sigaction");
      exit(1);
    }
```


Code (3/4)

```
    /* Parametrage de la temporisation */
    tempo.it_interval.tv_sec = 0;
    tempo.it_interval.tv_usec = PERIODE;
    tempo.it_value.tv_sec = 0;
    tempo.it_value.tv_usec = PERIODE;
    setitimer(TIMER, &tempo, NULL);

    /* Paramétrage d'ordonnancement du thread */
    param.sched_priority = PRIORITE;

    if (pthread_setschedparam(pthread_self(), DISCIPLINE, &param))
      perror("pthread_setschedparam");

    /* Durée d'exécution */
    sleep(DUREE);

    return NULL;
  }
```

Code (4/4)

```
/* Routine principale */
int main(void)
{
  pthread_t id_thread;
  pthread_attr_t attributs;

  pthread_attr_init(&attributs);
  pthread_create(&id_thread, &attributs, thread, NULL);
  pthread_join(id_thread, NULL);

  /* Résultats de latence */
  printf("Latence maximale : %d\n", latence_max);
  printf("Avance maximale : %d\n", avance_max);

  return EXIT_SUCCESS;
}
```


Tâches à réaliser

- Codage en langage C et compilation d'une tâche basée sur le code fourni
- Lancement de l'éxécution simultanée de plusieurs processus
- Modification de la priorité des processus
- Modification de la discipline des processus
- Analyse et explication des résultats obtenus

Compilation

- Ligne de commande pour la compilation

```
arm-linux-gcc -Wall thread_periodique.c \
-o thread_periodique -lpthread
```


3 Implémentation de threads périodiques concurrents

- Premier travail demandé
- Travail complémentaire

Tâches à réaliser

- Programmation en langage C d'un processus décomposé en deux tâches périodiques en se basant sur le code du thread périodique
- Utilisation des appels système POSIX
- Compilation croisée du code et exécution sur la cible
- Considération d'une tâche de haute priorité et d'une tâche de basse priorité
- Prise en compte d'une période commune aux deux tâches
- Évaluation de l'éxécution avec les disciplines `SCHED_FIFO` et `SCHED_RR`

Compilation

- Ligne de commande pour la compilation

```
arm-linux-gcc -Wall threads_concurrents.c \
-o threads_concurrents -lpthread
```

Tâches à réaliser

- Extension du code à n tâches périodiques
- Prise en compte de priorités et de disciplines différentes
- Considération d'une période propre à chacune des tâches