Programmation Temps Réel
Threads POSIX

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Septembre 2017

Bibliothèque Pthread

- Bibliothèque normalisée POSIX (ISO/IEC 9945-1:1996)
- Existe sous Linux et ses dérivés, et sous Windows (installation depuis http://sourceware.org/pthreads-win32/)
- Fichier d'en-tête :
  `pthread.h`
- Compilation et édition des liens :
  `gcc app.c -lpthread`
- ou
  `gcc -std=c99 app.c -lpthread`
- Déclaration d'un thread
  `pthread_t thread;`

Création d'un thread

```
int pthread_create(
        pthread_t *thread,
        const pthread_attr_t *attr,
        void *(*start_routine)(void *),
        void *arg);
```

- `thread` est un pointeur sur une variable de type `pthread_t`
  - argument de sortie!
- `attr` permet de définir les attributs du thread (`NULL` = attributs par défaut)
- `start_routine` correspond à la fonction qui est exécutée par le thread créé. La fonction exécutée par le thread créé devra avoir le prototype suivant:

  ```
  void *fonction(void *data);
  ```
- `arg` est un pointeur passé à la fonction

Création d'un thread

Exemple

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

void *func(void *arg) {
    printf("Hello from our first thread!\n");
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t thread;
    if (pthread_create(&thread, NULL, func, NULL) != 0) {
        perror("thread creation error");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Création de thread : passage d'argument

Exemple

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

void *func(void *arg) {
    char *msg = (char *) arg;
    printf("Message = %s\n", msg);
    return NULL;
   }

int main(int argc, char *argv[]) {
    pthread_t t;
    char *msg = "Threads are awesome!";
    if (pthread_create(&t, NULL, func, msg) != 0) {
        perror("thread creation error");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```


Jointure

- La jointure permet à un thread d'attendre qu'un autre se termine
- La jointure se fait via une fonction bloquante
  - Attente jusqu'à ce que le thread "joint" se termine
- Le thread terminé peut retourner une valeur au thread ayant effectué la jointure
  - Forme rudimentaire de communication inter-threads

Jointure

```
int pthread_join(
        pthread_t thread,
        void **value_ptr);
```

- Attend que la tâche en paramètre se termine
- Le thread appelant est bloqué jusqu'à la terminaison du thread spéficié
- `value_ptr` contient la valeur de retour de la tâche

Jointure : Exemple (1)

```
typedef struct {
    int a;
    int b;
} struct_t;

void *Tache1(void *arg) {
    struct_t *var;
    var = (struct_t *)arg;
    printf("Tache1: a=%d, b=%d\n",
    var->a,var->b);
    return NULL;
}

void *Tache2(void *arg) {
    int i = (int)arg;
    printf("Tache2: i=%d\n",i);
    return NULL;
}
```

```
int main(int argc,char *argv[])
{
    struct_t v;
    v.a = 1; v.b = 2;
    pthread_t thread;
    pthread_create(&thread,
                   NULL,
                   Tache1,
                   &v);
    pthread_join(thread,NULL);
    pthread_create(&thread,
                   NULL,
                   Tache2,
                   (void *)2);
    pthread_join(thread,NULL);
    return EXIT_SUCCESS;
}
```

Jointure : Exemple (2)

```c
typedef struct {
    int a;
    int b;
} struct_t;

void *Tache1(void *arg) {
    struct_t *var;
    var = (struct_t *)arg;
    printf("Tache1: a=%d, b=%d\n",
    var->a,var->b);
    return NULL;
}

void *Tache2(void *arg) {
    int i = (int)arg;
    printf("Tache2: i=%d\n",i);
    return NULL;
}
```

```c
int main(int argc,char *argv[])
{
    struct_t v;
    v.a = 1; v.b = 2;
    pthread_t thread1,thread2;
    pthread_create(&thread1,
                   NULL,
                   Tache1,
                   &v);
    pthread_create(&thread2,
                   NULL,
                   Tache2,
                   (void *)2);
    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);
    return EXIT_SUCCESS;
}
```


Terminaison: options

- La terminaison d'un thread peut être exécutée depuis:
  - Le thread lui-même:
    - **return**
    - `pthread_exit()`
  - Un autre thread:
    - `pthread_cancel()`
- ⚠ Mal terminer un thread peut laisser le système dans un état incohérent!!
  - Plus spécifiquement depuis un autre thread
- Pour terminer l'application (destruction de tous les threads):
  - `exit()`

Auto-terminaison

- `return value;`
  - ⚠ Attention avec le thread principal: Terminaison du programme!!
- `void pthread_exit(void *value)`
  - La fonction met fin au thread, et retourne `value` au thread attendant grâce à une jointure (`pthread_join()`).

Auto-terminaison: exemple (1)

```
void *tache1(void *arg) {
    printf("Tâche 1\n");
    return 3;
}
void *tache2(void *arg) {
    printf("Tâche 2\n");
    pthread_exit(4);
}
int main(int argc,char *argv[]) {
    pthread_t thread1;
    pthread_t thread2;
    pthread_create(&thread1,NULL,tache1,NULL);
    pthread_create(&thread2,NULL,tache2,NULL);
    void *statut1;
    void *statut2;
    pthread_join(thread1,&statut1);
    pthread_join(thread2,&statut2);
    printf("Statut1: %d, Status2: %d\n",
                    (int)(intptr_t)statut1,(int)(intptr_t)statut2);
    return EXIT_SUCCESS;
}
```

Dans quel etat se trouve le thread ensuite?

Attention

Auto-terminaison: exemple (2)

```
void *tache1(void *arg) {
    printf("Tâche 1\n");
    return 3;
}
void *tache2(void *arg) {
    printf("Tâche 2\n");
    pthread_exit(4);
}
int main(int argc,char *argv[]) {
    pthread_t thread1;
    pthread_t thread2;
    pthread_create(&thread1,NULL,tache1,NULL);
    pthread_create(&thread2,NULL,tache2,NULL);
    void *statut1;
    void *statut2;
    pthread_join(thread1,&statut1);
    pthread_join(thread2,&statut2);
    printf("Statut1: %d, Status2: %d\n",
                    statut1,statut2);
    pthread_exit(NULL);
}
```

Attention

Annulation par un autre thread

```
int pthread_cancel(pthread_t thread);
```

- `thread` est un pointeur sur une variable de type `pthread_t`
- Cette fonction permet de faire se terminer un thread depuis un autre
- La terminaison s'effectue sur un *point d'annulation*

Politique d'annulation (1)

- Le thread annulé peut définir sa politique d'annulation

```
int pthread_setcancelstate(
        int state,
        int *oldstate);
```

`state` peut prendre les valeurs:

- `PTHREAD_CANCEL_ENABLE`: Autorise l'annulation
- `PTHREAD_CANCEL_DISABLE`: Interdit l'annulation

`oldstate` contient ensuite l'ancienne valeur d'enable

Politique d'annulation (2)

- Si le thread autorise l'annulation, il est possible de définir son type d'annulation

```
int pthread_setcanceltype(
        int type,
        int *oldtype);
```

`type` peut prendre les valeurs:

- `PTHREAD_CANCEL_DEFERRED`: Autorise l'annulation en des point précis
- `PTHREAD_CANCEL_ASYNCHRONOUS`: Autorise l'annulation n'importe quand

`oldtype` contient ensuite l'ancien type d'annulation

Point d'annulation

- Un thread peut placer ses points d'annulations

```
void pthread_testcancel(void);
```

- L'annulation est donc permise en ces points
- ⚠ Attention, certains appels système sont des points d'annulation
  Exemple: `write()`, utilisé par `printf()`

Exemple (1)

```
int counter1 = 1;

void *tache1(void *arg) {
    int ancien_etat, ancien_type;
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,&ancien_etat);
    pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED,&ancien_type);
    while (true) {
        counter1++;
        if (counter1 % 100 == 0)
            pthread_testcancel();
    }
}

int main(int argc,char *argv[]) {
    pthread_t thread1;
    pthread_create(&thread1,NULL,tache1,NULL);
    void *statut1;
    usleep(50);
    pthread_cancel(thread1);
    pthread_join(thread1,&statut1);
    printf("Counter1: %d\n",counter1);
    return EXIT_SUCCESS;
```

Exemple (2)

```
int counter1 = 1;

void *tache1(void *arg) {
    int ancien_etat, ancien_type;
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,&ancien_etat);
    pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED,&ancien_type);
    while (true) {
        counter1++;
    }
}

int main(int argc,char *argv[]) {
    pthread_t thread1;
    pthread_create(&thread1,NULL,tache1,NULL);
    void *statut1;
    usleep(50);
    pthread_cancel(thread1);
    pthread_join(thread1,&statut1);
    printf("Counter1: %d\n",counter1);
    return EXIT_SUCCESS;
}
```


Exemple (3)

```
int counter1 = 1;

void *tache1(void *arg) {
    int ancien_etat, ancien_type;
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,&ancien_etat);
    pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED,&ancien_type);
    while (true) {
        counter1++;
        printf("counter1: %d\n");
    }
}

int main(int argc,char *argv[]) {
    pthread_t thread1;
    pthread_create(&thread1,NULL,tache1,NULL);
    void *statut1;
    usleep(50);
    pthread_cancel(thread1);
    pthread_join(thread1,&statut1);
    printf("Counter1: %d\n",counter1);
    return EXIT_SUCCESS;
}
```

Exemple (4)

```
int counter1 = 1;

void *tache1(void *arg) {
    int ancien_etat, ancien_type;
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,&ancien_etat);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,&ancien_type);
    while (true) {
        counter1++;
    }
}

int main(int argc,char *argv[]) {
    pthread_t thread1;
    pthread_create(&thread1,NULL,tache1,NULL);
    void *statut1;
    usleep(50);
    pthread_cancel(thread1);
    pthread_join(thread1,&statut1);
    printf("Counter1: %d\n",counter1);
    return EXIT_SUCCESS;
}
```


Exemple (5)

- Soit le thread `tache1` pouvant être annulé par un autre

```
void *tache1(void *arg) {
    int ancien_etat, ancien_type;
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,&ancien_etat);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,&ancien_type);
    // traitement divers

    FILE f=fopen(...);
    fprintf(f, ...);
    fprintf(f, ...);
    fclose(f);

    // traitement divers
}
```

- Que peut-il se passer?
- Que faire pour y remédier?

Auto-identification

- Un thread peut obtenir son identifiant avec la fonction:

```
pthread_t pthread_self(void);
```

- Il s'agit donc de la même valeur d'identifiant que celle retournée en premier argument de la fonction `pthread_create`
- Chaque thread dispose d'un numéro d'identifiant unique
- Sur linux x64, l'identifiant est un entier long non signé (unsigned long int)
- ⚠ il n'est pas garanti que l'identifiant d'un thread soit un entier ! Le type est propre à l'implémentation (voir frame suivante pour comparer des identifiants de threads).

Comparaison d'identifiants

- Comparer deux identifiants de threads doit se faire avec la fonction:

```
int pthread_equal(pthread_t t1, pthread_t t2);
```

- Si les threads t1 et t2 sont égaux, alors la fonction retourne une valeur différente de zéro
- Il est nécessaire d'utiliser cette fonction pour comparer les identifiants, car rien ne garanti que les types `pthread_t` soient des valeurs comparables avec l'opérateur d'égalité =

Rendre le processeur

- Il est possible de forcer le thread appelant à relâcher le processeur avec la fonction:

```
#include <sched.h>

int sched_yield();
```

- Après l'appel à cette fonction, le thread est placé à la fin de la file d'attente des threads en attente du processeur
- Le thread suivant en attente du processeur est alors activé
- Renvoie zéro en cas de succès