Compléments sur les PThreads

Jean-Louis PAZAT
IRISA / INSA

Exemple de threads utilisateur: Threads Posix

- Interface portable de programmation (API)
- Fonctionnalités
  - création/destruction de Threads
  - Synchronisation
    - attente de terminaison
    - sémaphores d 'exclusion mutuelle
    - conditions
  - variables « thread locales »

création/destruction de Threads

```
int pthread_create(
  pthread_t *new_thread_ID,
  const pthread_attr_t *attr,
  void * (*start_func)(void *),
  void *arg);
```

- crée un thread qui exécute la fonction start_func
- prend arg en paramètre et rend le tid du thread créé

`pthread_exit(void *status)`

- termine le thread courant

`pthread_cancel(tid)`

- demande au thread tid de se terminer

ordonnancement

- spécifié par les attributs du thread

`contentionscope` PTHREAD_SCOPE_PROCESS

Ordonnancement par la bibliothèque (local)
ou par le système (global)

`priority` NULL

Définit la priorité (dépend de l 'ordonnanceur)

`policy` SCHED_OTHER

Définit le type d 'ordonnancement

`inheritsched` PTHREAD_EXPLICIT_SCHED

Détermine si les paramètres d 'ordonnancement sont hérités ou explicitement définis

Autres attributs (attr)

detachstate PTHREAD_CREATE_JOINABLE

Permet à d 'autres threads d 'attendre la terminaison du thread courant avant libération de la mémoire

stackaddr **NULL**

Permet d 'accéder à la pile du thread

stacksize **NULL**

Fixe la taille de la pile du thread

```
#include <stdio.h>
#include <pthread.h>
void* do_loop(void* data){
 int i;
 int me = *((int*)data);     /* numéro du thread */
 for (i=0; i<10; i++)
   printf("%d - etape %d\n", me, i);
 pthread_exit(NULL);
}
int main(int argc, char* argv[]){
 int t_id;         /* thread ID */
 pthread_t  p_thread;   /* thread structure  */
 int a = 1;             /* thread 1 id       */
 int b = 2;             /* thread 2 id       */
 t_id = pthread_create(&p_thread, NULL, do_loop,(void*)&a);
 do_loop((void*)&b);
 return 0;
}
```

Threads et signaux unix

- Danger …
  - Très dépendant des implémentations
- Spécificités:
  - chaque thread a son masque
    - pthread_sigmask()
  - signaux entre threads d 'un même processus
    - pthread_kill(pthread_t thread, int sig)

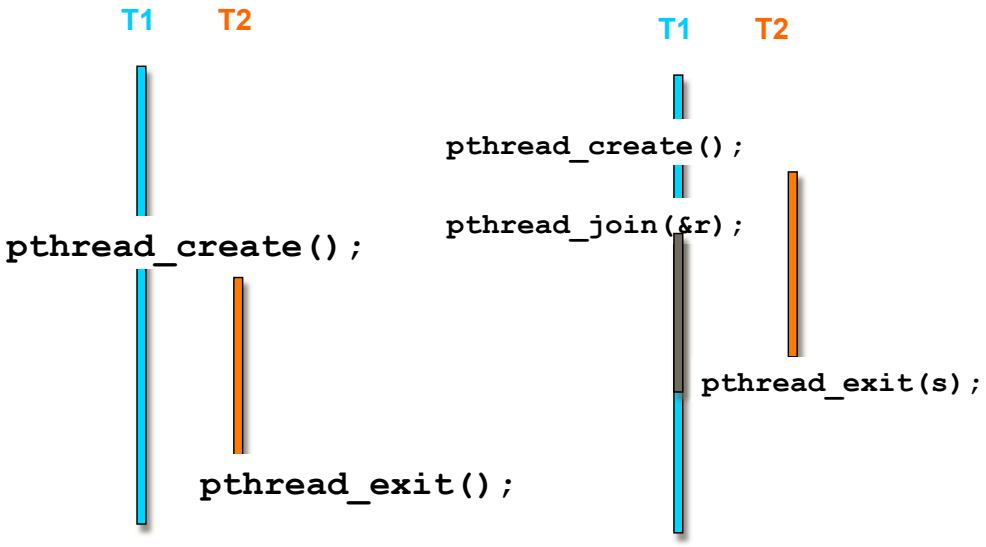
Synchronisation : join

- Permet d 'attendre la terminaison d 'un thread

```
int pthread_join(pthread_t t, void **status);
```

  - permet de récupérer une valeur retour
    - Création thread +join
      - Peut remplacer un appel de fonction asynchrone
    - le résultat sera placé dans `**status`
  - Le thread t sera détruit lors du premier pthread_join
    - ne plus accéder à la pile du thread t !!!!
    - un seul thread peut faire un join
    - t doit être « `joinable` »

join



Demande de terminaison (cancel)

- Pour arrêter un thread à partir d 'un autre
- 3 modes
  - disable
    - les demandes restent « pendantes » jqa chgt de mode
  - deferred
    - les demandes sont traitées dès que possible
      - cancellation points: wait, join, test_cancel, i/o, ...
  - asynchronous
    - les demandes seront traitées n 'importe quand

Sémaphores d 'exclusion mutuelle

- 2 types de sémaphores
  - rapides `PTHREAD_MUTEX_INITIALIZER;`
  - réentrants `PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP`
- création/initialisation

```
pthread_mutex_t sema = PTHREAD_MUTEX_INITIALIZER_NP
```

- utilisation

```
pthread_mutex_lock(&sema);
< section critique >
pthread_mutex_unlock(&sema);
```

- destruction

```
rc = pthread_mutex_destroy(&sema);
```

Sémaphores d 'exclusion mutuelle

- Plus précisément:

```
int rc = pthread_mutex_lock(&sema);
if (rc) { /* erreur d 'init/signal/... */
    perror("pthread_mutex_lock");
    pthread_exit(NULL);
```

Conditions (files d 'attente)

- création/initialisation

```
pthread_cond_t pour_faire = PTHREAD_COND_INITIALIZER;
```

- blocage

```
rc = pthread_cond_wait(&pour_faire, &mutex);
ou
pthread_cond_timedwait(&pour_faire, &mutex, &timeout);
```

- réveil (pas besoin du sémaphore d 'exclusion mutuelle !)

```
rc = pthread_cond_signal(&pour_faire)
ou
rc = pthread_cond_broadcast(&pour_faire)
```

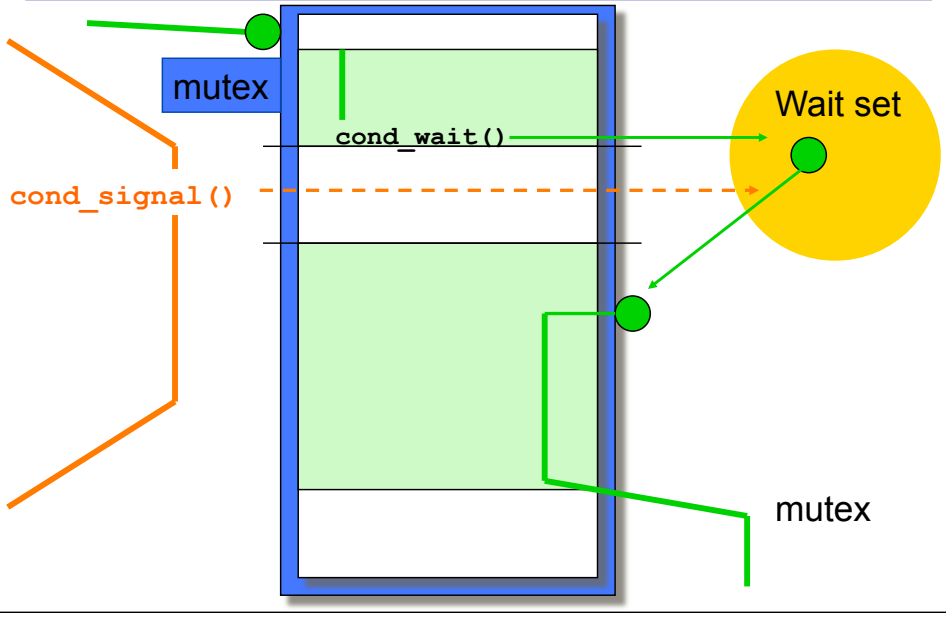
- destruction

```
rc = pthread_cond_destroy(&pour_faire);
```

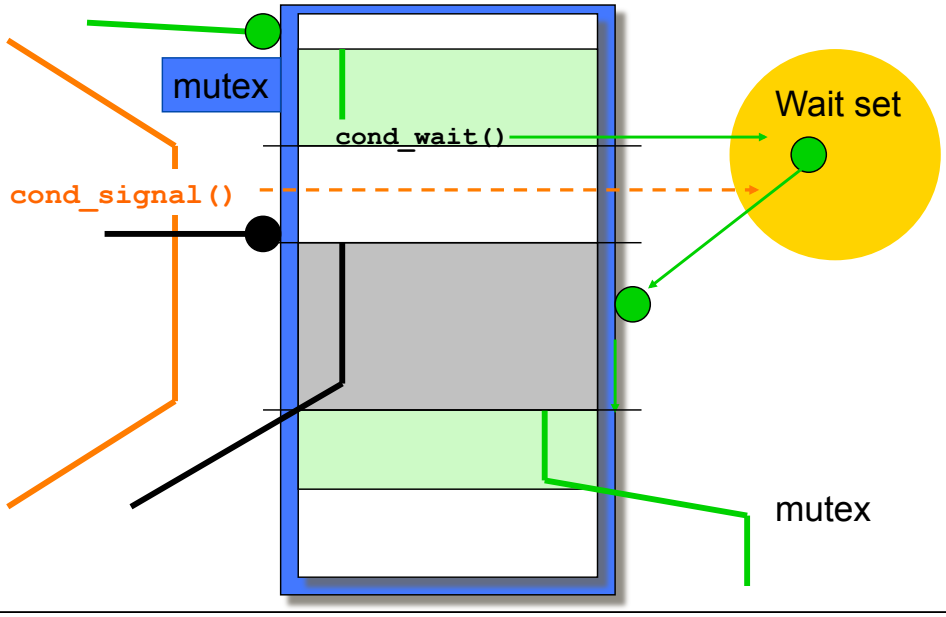
Conditions (utilisation)

```
pthread_mutex_lock(&mutex);

  while (peux_pas_faire())
    pthread_cond_wait(&pour_faire, &mutex);

   <faire>

pthread_mutex_unlock(&mutex);
...
if (peux_faire())
   pthread_cond_signal(&pour_faire)
```

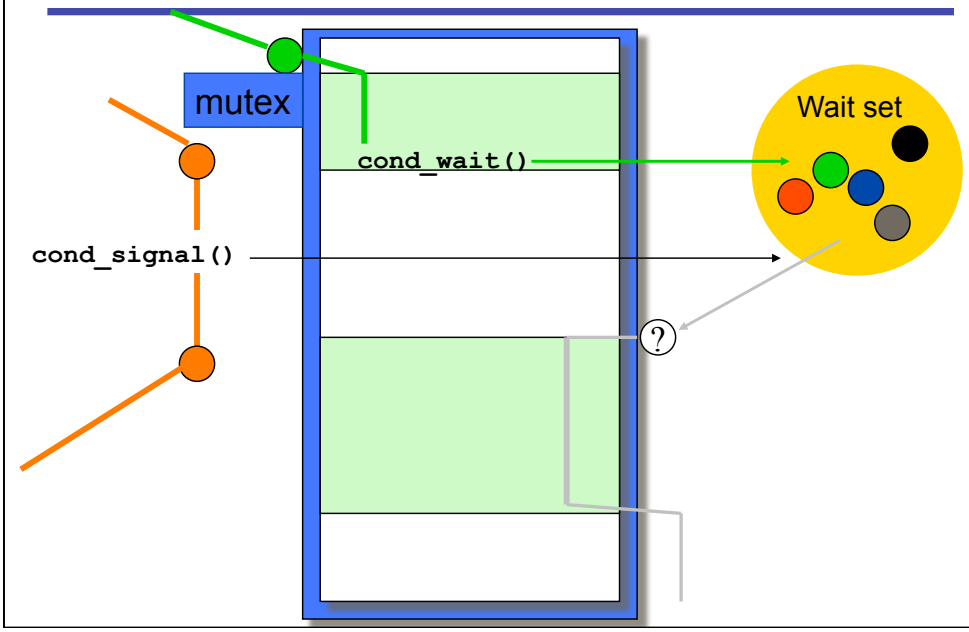
Conditions



Conditions



Synchronisations par blocage/réveil



Synchronisations par blocage/réveil