Thread di Unix/Linux

Disponibili da tempo in varie librerie, una “LinuxThreads, è presente in tutte le distribuzioni.

Sfruttando la chiamata di sistema clone() che crea un nuovo processo senza cambiare spazio di indirizzamento, è stato possibile creare librerie che implementano i thread facilmente.

Quindi i thread di Linux sono processi che condividono lo spazio degli indirizzi.

Le raccomandazioni POSIX aiutano a garantire un certo grado di uniformità nella gestione dei thread nelle varie versioni di Unix Linux. Noi studieremo i ptrhead ossi i Posix Thread

Le funzioni per la gestione di un thread in linux sono:

- pthread_create (thread,attr,start_routine,arg)
- pthread_exit (status)
- pthread_join (threadid,status)
- pthread_detach (threadid)
- pthread_cancel (thread)
- pthread_attr_init (attr)
- pthread_attr_destroy (attr)
- pthread_attr_setdetachstate (attr,detachstate)
- pthread_attr_getdetachstate (attr,detachstate)

pthread_create

La funzione pthread_create() crea un thread, la sua sintassi è:

```
#include<pthread.h>
int pthread_create(
pthread_t *tid, // puntatore in cui conservare il tid del thread
pthread_attr_t *attr, // puntatore agli gli attributi del thread
void*(*start_routine)(void*),// puntatore alla funzione
void *arg // puntatore agli argomemti della funzione
);
```

La funzione pthread_create crea un nuovo thread che esegue la funzione start_routine a cui passa gli argomenti arg.

La variabile attrs specifica gli attributi del thread, se è NULL assume i valori di default, alcuni di questi possono essere inizializzare ed eliminati dal programmatore attraverso le funzioni pthread_attr_init e pthread_attr_destroy.

Altre routine che vedremo dopo si utilizzano per interrogare o impostare gli attributi, i principali riguardano:

- Stato indipendente o assemblabili
- Pianificazione del tipo di eredità
- Politica e i parametri di Scheduling
- Dimensione e indirizzi dello stack
- Dimensione dello Stack guard (overflow)

In caso di successo tid conterrà l'identificatore del thread creato e la funzione ritorna 0; in caso di fallimento viene restituito un codice di errore, i possibili errori sono:

- EAGAIN
  Il sistema non disponeva delle risorse necessarie
- EINVAL

Il valore specificato in attr non è valido

- EPERM

  Il chiamante non dispone delle autorizzazioni appropriata per impostare i parametri di programmazione richiesti o politica di scheduling.

pthread_exit()

Il nuovo thread può terminare esplicitamente attraverso la chiamata pthread_exit() o pthread_cancel(), o implicitamente quando la funzione start_routine(), o quando il processo principale termina.

La funzione pthread_exit ha la seguente sintassi

```
void pthread_exit(void *status)
```

e permette al programmatore di specificare il parametro status di ritorno consultabile da altri thread con pthread_join.

Se il programma principale termina prima dei suoi thread, tutti questi saranno terminati, ma se si ha come ultima istruzione nel mail una chiamata a pthread_exit, il main() sarà tenuto in vita per supportare la terminazione di tutti i suoi thread.

Diamo un esempio di un programma che lancia cinque thread, ogni Thread scrive un “Hello Word!” e termina.

Ricordiamo che per compilare un programma con la libreria pthread si deve utilizzare il seguente commando

gcc -lpthread nomefile

Su Ubuntu se le librerie lptrhead non sono già istallate è possibile farlo con il comando

sudo apt-get install gcc build-essential

hello.c

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5
void *PrintHello(void *threadid)
{long tid;
```

```
 tid = (long)threadid;
 printf("Hello World! It's me, thread #%ld!\n", tid);
 pthread_exit(NULL);}

int main (int argc, char *argv[])
{pthread_t threads[NUM_THREADS];
 int rc;
 long t;
 for(t=0; t<NUM_THREADS; t++)
 {printf("In main: creating thread %ld\n", t);
 rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
 if (rc)
  {printf("ERROR; return code from pthread_create() is %d\n", rc);
   exit(-1);}
 }
  /* Last thing that main() should do */
   pthread_exit(NULL);
}
```

Abbiamo visto che nella funzione pthread_create è possibile passare degli argomenti arg al trhead.

Diamo un esempio di un programma che passa un intero ai trhead.

hello_arg1.c

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS   8
char *messages[NUM_THREADS];

void *PrintHello(void *threadid)
{int *id_ptr, taskid;
 sleep(1);
 id_ptr = (int *) threadid;
 taskid = *id_ptr;
 printf("Thread %d: %s\n", taskid, messages[taskid]);
 pthread_exit(NULL);
}
int main(int argc, char *argv[])
{pthread_t threads[NUM_THREADS];
 int *taskids[NUM_THREADS];
 int rc, t;
 messages[0] = "English: Hello World!";
```

```
 messages[1] = "French: Bonjour, le monde!";
 messages[2] = "Spanish: Hola al mundo";
 messages[3] = "Klingon: Nuq neH!";
 messages[4] = "German: Guten Tag, Welt!";
 messages[5] = "Russian: Zdravstvytye, mir!";
 messages[6] = "Japan: Sekai e konnichiwa!";
 messages[7] = "Latin: Orbis, te saluto!";
 for(t=0;t<NUM_THREADS;t++)
 {taskids[t] = (int *) malloc(sizeof(int));
  *taskids[t] = t; printf("Creating thread %d\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
  if(rc)
  {printf("ERROR; return code from pthread_create() is %d\n", rc);
   exit(-1);}
 }
pthread_exit(NULL);
}
```

Diamo un altro esempio in cui si passa una struttura.

hello_arg3.c

```
void *PrintHello(void *threadarg)
{
   struct thread_data *my_data;
```

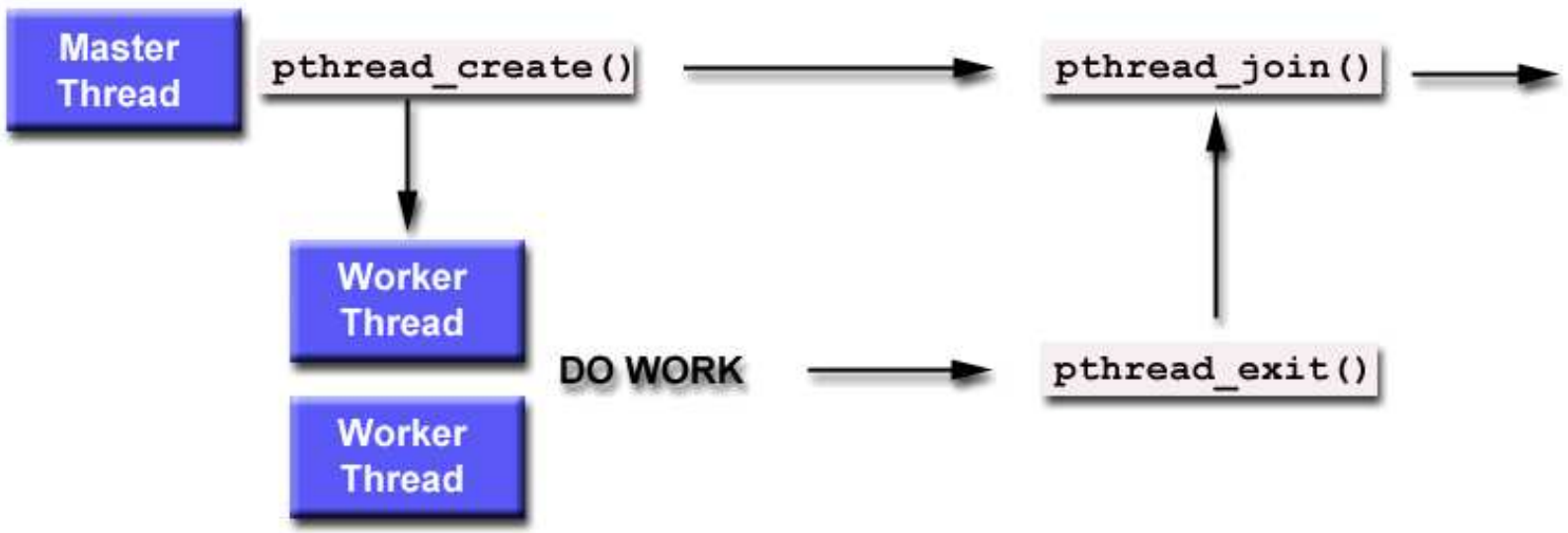
```
    ...
    my_data = (struct thread_data *) threadarg;
    taskid = my_data->thread_id;
    sum = my_data->sum;
    hello_msg = my_data->message;
    ...
}

int main (int argc, char *argv[])
{
    ...
    thread_data_array[t].thread_id = t;
    thread_data_array[t].sum = sum;
    thread_data_array[t].message = messages[t];
    rc = pthread_create(&threads[t], NULL, PrintHello,
         (void *) &thread_data_array[t]);
    ...
}
```

pthread__join()

La funzione pthread_join() sospende il thread corrente in attesa della terminazione di un altro thread, la sua sintassi è:

```
#include<pthread.h>
int pthread_join(
pthread_t th, // puntatore all thread da attendere
void **thread_return // vaolre di ritorno del thread
);
```

La funzione pthread_join() sospende l'esecuzione del thread corrente fino alla terminazione del thread th. La variabile thread_return conterrà il valore che il thread terminato ha passato a pthread_exit().



La fine di un thread può essere attesa al più da un solo altro thread.

Quando si crea un thread, uno dei suoi attributi è la possibilità che esso possa essere joinable o detached. Solo i threads creati joinable possono utilizzare la funzione pthread_join(), e solo i thread create detached, possono essere joined.

Posix standard crea thead joinable. Per creare esplicitamente un thread joinable o detached, si deve:

- Dichiarare una variabile di tipo pthread_attr_t
- Inizializzarla tramite la funzione pthread_attr_init()
- Settarla con gli attributi detached status con la funzione pthread_attr_setdetachstate()
- Alla fine liberare le risorse con la funzione pthread_attr_destroy()

Questo esempio mostra come aspettare un thread usando la join routine e creare esplicitamente nello stato joinable.

join.c

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

```c
#define NUM_THREADS 4
void *BusyWork(void *t)
{int i;
 long tid;
 double result=0.0;
 tid = (long)t;
 printf("Thread %ld starting...\n",tid);
 for (i=0; i<1000000; i++)
 {result = result + sin(i) * tan(i);}
  printf("Thread %ld done. Result = %e\n",tid, result);
  pthread_exit((void*) t);}

int main (int argc, char *argv[])
{pthread_t thread[NUM_THREADS];
 pthread_attr_t attr;
 int rc;
 long t;
 void *status;
 /* Initialize and set thread detached attribute */
 pthread_attr_init(&attr);
 pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
 for(t=0; t<NUM_THREADS; t++)
 {printf("Main: creating thread %ld\n", t);
```

```
  rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
  if (rc)
  {printf("ERROR; return code from pthread_create() is %d\n", rc);
   exit(-1);}
 }
/* Free attribute and wait for the other threads */
 pthread_attr_destroy(&attr);
 for(t=0; t<NUM_THREADS; t++)
 {rc = pthread_join(thread[t], &status);
  if(rc)
  {printf("ERROR; return code from pthread_join() is %d\n", rc);
   exit(-1);}
  printf("Main:join thread %ld status of %ld\n",t,(long)status);
  }
 printf("Main: program completed. Exiting.\n");
 pthread_exit(NULL);
}
```

pthread_detach()

La funzione pthread_ detach() marca il thread come detach, la sua sintassi è:

```
#include<pthread.h>
int pthread_detach(
pthread_t th, // puntatore all thread da attendere
);
```

Quando un thread detached termina, le sue risorse sono automaticamente rilasciate senza il bisogno di una chiamata joint da parte di un altro thread.

pthread_cancel()

Come detto il nuovo thread può terminare esplicitamente attraverso la chiamata pthread_exit() o pthread_cancel(), o implicitamente quando la funzione start_routine(), o quando il processo principale termina.

La funzione pthread_cancel() richiede che un thread sia cancellato. Lo stato di cancellabilità e il tipo di thread determinano se esso è effettivamente cancellabile.

Diamo un esempio di un programma che crea un thread e quindi lo cancella.

Il thread principale aspetta il thread cancellato per controllare se il suo status di uscita è il valore PTHREAD_CANCELED.

Join_canc.c

```c
#include <pthread.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>

#define handle_error_en(en, msg) \
do { errno = en; perror(msg); exit(EXIT_FAILURE); } while (0)

static void * thread_func(void *ignored_argument)
{
int s;
/* Disable cancellation for a while, so that we don't
  immediately react to a cancellation request */
s = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
if (s != 0) handle_error_en(s, "pthread_setcancelstate");
printf("thread_func(): started; cancellation disabled\n");
sleep(5);
printf("thread_func(): about to enable cancellation\n");
s = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
```

```c
if (s != 0)
   handle_error_en(s, "pthread_setcancelstate");
/* sleep() is a cancellation point */
sleep(1000);/* Should get canceled while we sleep */
/* Should never get here */
 printf("thread_func(): not canceled!\n");
 return NULL;
}

int
main(void)
{pthread_t thr;
 void *res;
 int s;
/* Start a thread and then send it a cancellation request */
s = pthread_create(&thr, NULL, &thread_func, NULL);
if (s != 0)
   handle_error_en(s, "pthread_create");
sleep(2);           /* Give thread a chance to get started */
printf("main(): sending cancellation request\n");
s = pthread_cancel(thr);
if (s != 0)
```

```
   handle_error_en(s, "pthread_cancel");
/* Join with thread to see what its exit status was */
s = pthread_join(thr, &res);
if (s != 0)
   handle_error_en(s, "pthread_join");
if (res == PTHREAD_CANCELED)
   printf("main(): thread was canceled\n");
else
   printf("main(): thread wasn't canceled (shouldn't
happen!)\n");
exit(EXIT_SUCCESS);
}
```

Vediamo ora brevemente le funzioni per la gestione dello stack

- pthread_attr_getstacksize (attr, stacksize)
- pthread_attr_setstacksize (attr, stacksize)
- pthread_attr_getstackaddr (attr, stackaddr)
- pthread_attr_setstackaddr (attr, stackaddr)

E un esempio di utilizzo

```
#include <pthread.h>
#include <stdio.h>
#define NTHREADS 4
#define N 1000
#define MEGEXTRA 1000000
 pthread_attr_t attr;

void *dowork(void *threadid)
{  double A[N][N];
   int i,j;
```

```
    long tid;
    size_t mystacksize;
    tid = (long)threadid;
    pthread_attr_getstacksize (&attr, &mystacksize);
    printf("Thread %ld: stack size = %li bytes \n", tid,
mystacksize);
    for (i=0; i<N; i++)
      for (j=0; j<N; j++)
       A[i][j] = ((i*j)/3.452) + (N-i);
    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t threads[NTHREADS];
    size_t stacksize;
    int rc;
    long t;

    pthread_attr_init(&attr);
    pthread_attr_getstacksize (&attr, &stacksize);
    printf("Default stack size = %li\n", stacksize);
```

```
   stacksize = sizeof(double)*N*N+MEGEXTRA;
   printf("Amount of stack needed per thread =
%li\n",stacksize);
   pthread_attr_setstacksize (&attr, stacksize);
   printf("Creating threads with stack size = %li
bytes\n",stacksize);
   for(t=0; t<NTHREADS; t++){
      rc = pthread_create(&threads[t], &attr, dowork, (void
*)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is
%d\n", rc);
         exit(-1);
      }
   }
   printf("Created %ld threads.\n", t);
   pthread_exit(NULL);

}
```

Altre funzioni sui thread sono:

- pthread_self()

  ritorna l'ID del thread chiamante .

- pthread_ equal(thread1,thread2)

  che compara due thread. Se i due ID sono differenti, ritorna zero, viceversa ritorna un valore diverso da zero