Thread in sistemi UNIX


```
NAME
     pthread_create - create a new thread

SYNOPSIS
     #include <pthread.h>

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

- int pthread_create(pthread_t *tid, pthread_attr_t *attr, void *(*funct)(void*), void *arg)

Descrizione	invoca l'attivazione di un thread
Parametri	1) *tid: buffer di informazioni sul thread 2) *attr: buffer di attributi (NULL identifica il default) 3) (*funct): puntatore a funzione per il thread 4) *arg: puntatore al buffer degli argomenti
Restituzione	un valore diverso da zero in caso di fallimento

(*funct) del terzo parametro della funzione, è un oggetto che identifica esattamente un blocco di codice, tant'è vero che abbiamo infatti un puntatore ad una funzione. Noi sappiamo che gli indirizzi di memoria delle funzioni all'interno di un address space possono essere identificati tramite il nome di una funzione. Questa funzione deve ritornare un puntatore generico void* e deve prendere un puntatore generico in input.
Bene, questa funzione è la funzione in cui deve vivere il thread che stiamo lanciando.
L'ultimo parametro è il puntatore che dovrà essere poi passato a questa funzione quando verrà fatta partire: quindi passiamo l'informazione che verrà in qualche modo "comunicata" a questa funzione che parte, come parametro.
Il primo parametro è un puntatore ad un tipo pthread_t che ci serve per ottenere dal sistema alcune informazioni sul thread appena lanciato.
Il secondo parametro è un puntatore ad un tipo pthread_attr_t, in cui passiamo informazioni in input al sistema.

pthread_t e' un **unsigned int**

"La funzione pthread_create() viene utilizzata per creare un nuovo thread, con gli attributi specificati da attr, all'interno di un processo. Se attr è NULL, vengono utilizzati gli attributi predefiniti. Se gli attributi specificati da attr vengono modificati successivamente, gli attributi del thread non ne risentono. Al completamento con successo, pthread_create() memorizzerà l'ID del thread creato nella posizione specificata da thread.

Il thread viene creato eseguendo start_routine con arg come unico argomento. Se start_routine ritorna, l'effetto è come se ci fosse una chiamata implicita a pthread_exit() utilizzando il valore restituito di start_routine come stato di uscita. Si noti che il thread in cui main() è stato originariamente invocato differisce da questo. Quando ritorna da main(), l'effetto è come se ci fosse una chiamata implicita a exit() utilizzando il valore di ritorno di main() come stato di uscita."

- "Manuale Ufficiale" - 15 marzo 2014

- void pthread_exit(void *status)

Descrizione	invoca la terminazione del thread chiamante
Parametri	*status: puntatore che definisce il codice di uscita
Restituzione	non ritorna in caso di successo

```
NAME
     pthread_exit - terminate the calling thread

SYNOPSIS
     #include <pthread.h>

     void
     pthread_exit(void *value_ptr);
```

"La funzione pthread_exit() termina il thread chiamante e rende disponibile il valore value_ptr a qualsiasi join riuscito con il thread di terminazione. Tutti i gestori di pulizia dell'annullamento che sono stati inviati e non sono ancora stati estratti vengono estratti nell'ordine inverso rispetto a quando sono stati inviati e quindi eseguiti. Dopo che tutti i gestori di annullamento sono stati eseguiti, se il thread contiene

dati specifici del thread, le funzioni di distruzione appropriate vengono chiamate in un ordine non specificato. La terminazione del thread non rilascia alcuna risorsa di processo visibile dell'applicazione, inclusi, ma non limitati a, mutex e descrittori di file, né esegue alcuna azione di pulizia a livello di processo, inclusa, ma non limitata a, chiamare le routine atexit() che possono esistere.

Viene effettuata una chiamata implicita a pthread_exit() quando un thread diverso da quello in cui main() è stato invocato per la prima volta ritorna dalla routine di avvio utilizzata per crearlo. Il valore restituito dalla funzione funge da stato di uscita del thread.
Il comportamento di pthread_exit() è indefinito se chiamato da un gestore di annullamento o da una funzione di distruzione richiamata come risultato di una chiamata implicita o esplicita a pthread_exit().
Dopo che un thread è terminato, il risultato dell'accesso alle variabili locali (auto) del thread è indefinito. Quindi, i riferimenti alle variabili locali del thread in uscita non dovrebbero essere usati per il valore del parametro pthread_exit() value_ptr.
Il processo terminerà con uno stato di uscita pari a 0 dopo che l'ultimo thread è stato terminato. Il comportamento è come se l'implementazione chiamasse exit() con un argomento zero al momento della chiusura del thread."

- "Manuale Ufficiale" - 15 marzo 2014

Il puntatore generico STATUS, ti dice dove sono posizionate le informazioni all'interno dell'address space, dove eventualmente può accedere un altro thread che stava attendendo la terminazione del thread e ovviamente può richiedere il codice di terminazione, ossia il puntatore.

- int pthread_join(pthread_t tid, void **status)

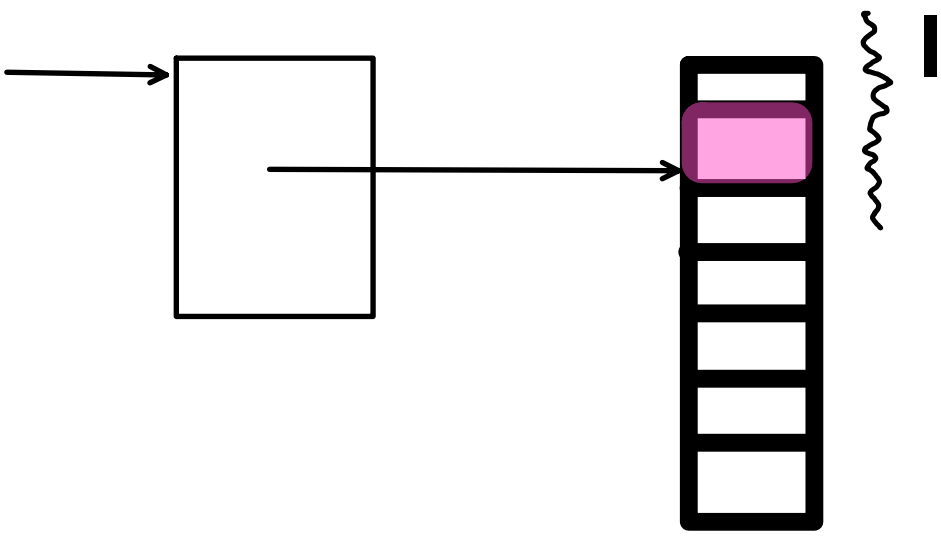
Descrizione	invoca l'attesa di terminazione di un thread
Parametri	1) tid: identificatore del thread (indicativo) 2) **status: puntatore al puntatore al buffer contenente il codice di uscita
Restituzione	-1 in caso di fallimento, altrimenti l'identificatore del thread terminato

```
NAME
     pthread_join - wait for thread termination

SYNOPSIS
     #include <pthread.h>

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

"La funzione pthread_join() sospende l'esecuzione del thread chiamante finché il thread di destinazione non termina, a meno che il thread di destinazione non sia già terminato."

- "Manuale Ufficiale" - 15 marzo 2014

Devo passare il codice numerico del thread che voglio attendere, che è il primo parametro, ma come secondo parametro passo l'indirizzo di memoria di una variabile dove posso mantenere un void*, ossia sto identificando l'indirizzo di memoria di un'area dove mi può essere consegnato un altro indirizzo, ossia se un thread ha completato la sua esecuzione con un pointer I e io lo attendo, eventualmente, mi viene consegnato il pointer I, che implica dire che mi viene indicato dove posso raggiungere la memoria per andare a lavorare sui dati che erano stati toccati dal thread stesso.



Come fa un thread a conoscere il suo codice numerico?

- pthread_t pthread_self(void)

Descrizione	chiede l'identificatore del thread chiamante
Restituzione	-1 in caso di fallimento, altrimenti l'identificatore del thread