

Programmazione concorrente tramite threads

Matteo Dessalvi

Gruppo Utenti Linux Cagliari



Gruppo Utenti Linux Cagliari h...?

Sommario

- Cosa è la programmazione concorrente?
- Definire i processi
- Programmazione multiprocesso
- Vantaggi e svantaggi
- Cosa sono i threads?
- Programmazione multithreading
- Vantaggi e svantaggi

Gruppo Utenti Linux Cagliari h...?

Configurazione della Linux box

- Distribuzione Mandrake 9.1
- Kernel 2.4.21 (va comunque bene uno qualunque dei kernel stabili della serie 2.4.x)
- Gcc 3.2.2
- Glibc 2.3.1
- Gdb 5.3 (con supporto al debug di applicazioni multi-threading)

Gruppo Utenti Linux Cagliari h...?

Cosa è la programmazione concorrente?

Con questo termine si indica quell'insieme di tecniche e di strumenti necessari a poter supportare più attività simultanee in una applicazione software.

Questa è una caratteristica dei cosiddetti sistemi multiprogrammati.

Gruppo Utenti Linux Cagliari h...?

Caratteristiche della multiprogrammazione

- Consentire a più utenti di accedere contemporaneamente ad un sistema informatico
- Consentire ad un solo utente l'esecuzione di più programmi simultaneamente
- Consentire ad un singolo programma di scomporre la propria attività in più attività concorrenti

Gruppo Utenti Linux Cagliari h...?

Processi

La programmazione concorrente è basata sul concetto di processo.

Un processo è considerato l'unità di esecuzione di base, su un s.o. Unix.

Ogni processo ha il suo spazio di indirizzamento privato, il suo stack ed il suo heap. Due processi non condividono mai il proprio spazio di indirizzamento.

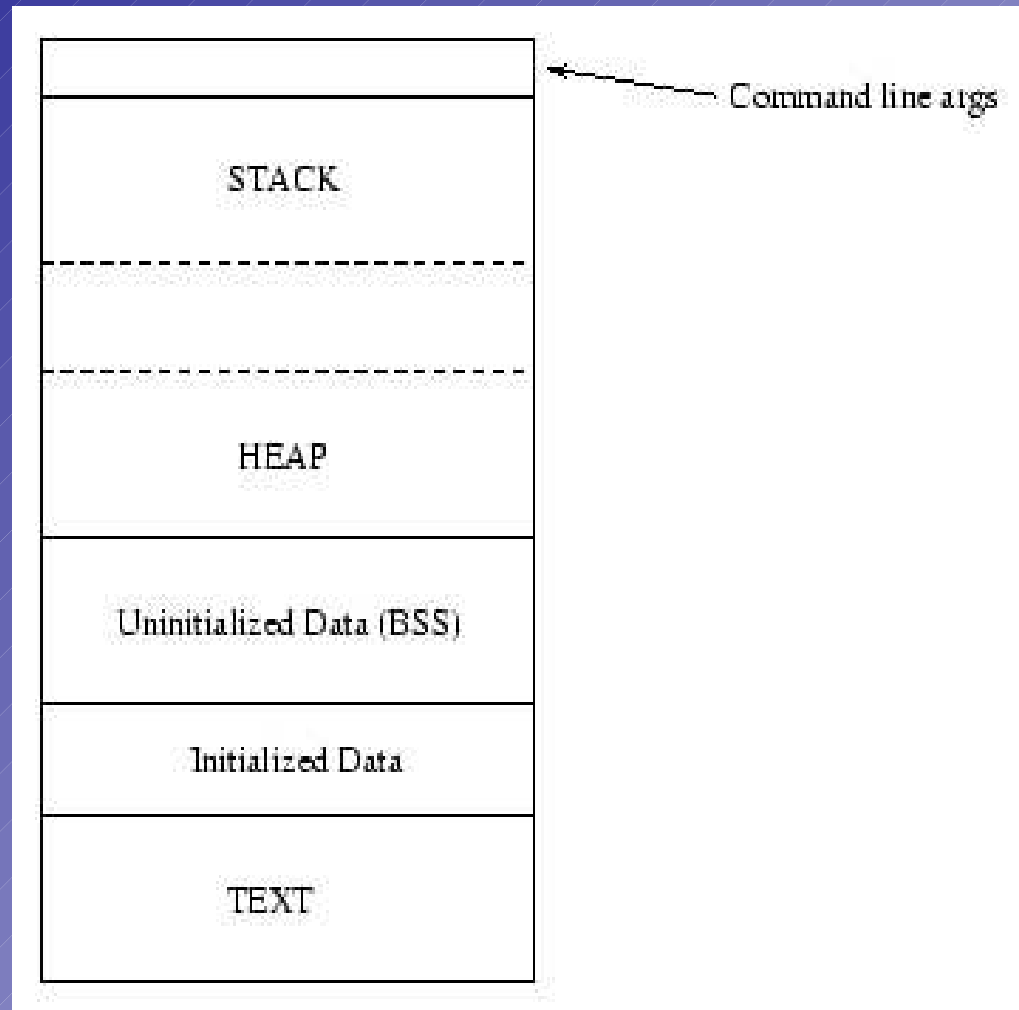
Gruppo Utenti Linux Cagliari h...?

Caratteristiche di un processo

- Un program counter (PC)
- Un corpo (ossia il codice che viene eseguito)
- Uno spazio di indirizzamento privato (diviso in zona dati, stack e heap)
- Una tabella di descrittori di files
- Un descrittore di processo (PID)
- Una tabella dei segnali

Gruppo Utenti Linux Cagliari h...?

Struttura di un processo



Gruppo Utenti Linux Cagliari h...?

Multiprocesso

Nei sistemi Unix il multitasking viene implementato tramite la system call `fork()`. Questo significa che il processo padre (che ha chiamato la `fork`) verrà “diviso” in due.

Il processo figlio eredita ogni caratteristica del processo padre:

i suoi dati, i file descriptors, il codice.

Da questo punto in poi i processi intraprendono strade differenti.

Gruppo Utenti Linux Cagliari h...?

Vantaggi e svantaggi

- La programmazione multiprocesso permette l'esistenza dei sistemi multitasking.
- Ma il meccanismo di creazione di nuovi processi porta a 2 problemi principali:
- Per un numero elevato di processi in esecuzione (ad esempio un web server) questo può causare un elevato sovraccarico nella macchina.
- La condivisione dei dati è intrinsecamente complessa.

Gruppo Utenti Linux Cagliari h...?

Cosa sono i threads?

Possiamo definire un thread come un flusso di esecuzione, non necessariamente legato ad uno spazio di indirizzamento privato.

A differenza dei processi, quando un thread viene creato, condivide il suo spazio di memoria con tutti gli altri threads che fanno parte del processo.

Gruppo Utenti Linux Cagliari h...?

All'interno

Un thread è composto essenzialmente da:

- 1) un program counter privato
- 2) uno stack
- 3) una tabella dei segnali

Quindi, la creazione di un thread è molto meno dispendiosa di quella di un processo.

Gruppo Utenti Linux Cagliari h...?

Threads POSIX sotto Linux

L'accesso alle funzioni per creare, distruggere e manipolare i threads avviene attraverso la libreria pthread (libpthread), che ormai da tempo è parte delle glibc.

Implementa lo standard POSIX 1003.1c API (Application Programming Interface).

L'unica parte non conforme allo standard è la gestione dei segnali.

Gruppo Utenti Linux Cagliari h...?

Creazione di un thread

Faremo uso delle funzioni definite nell'header *pthread.h* della libreria Pthread. La chiamata per creare un thread è definita da questo prototipo:

```
pthread_create(pthread_t ThID,  
               pthread_attribute ATTR,  
               void* StartRoutine,  
               void ARG)
```

Gruppo Utenti Linux Cagliari h...?

Argomenti di pthread_create

- **ThID** : è l'id del thread creato e costituisce un identificativo univoco mentre esso è in uso. In particolare pthread_t è un tipo di dato opaco.
- **ATTR**: è un attributo del thread appena creato. Può indicare le caratteristiche del thread riguardo alle operazioni di join e detach, oppure allo scheduling.
- **StartRoutine**: è la routine di partenza del thread, a cui viene passato come argomento ARG.

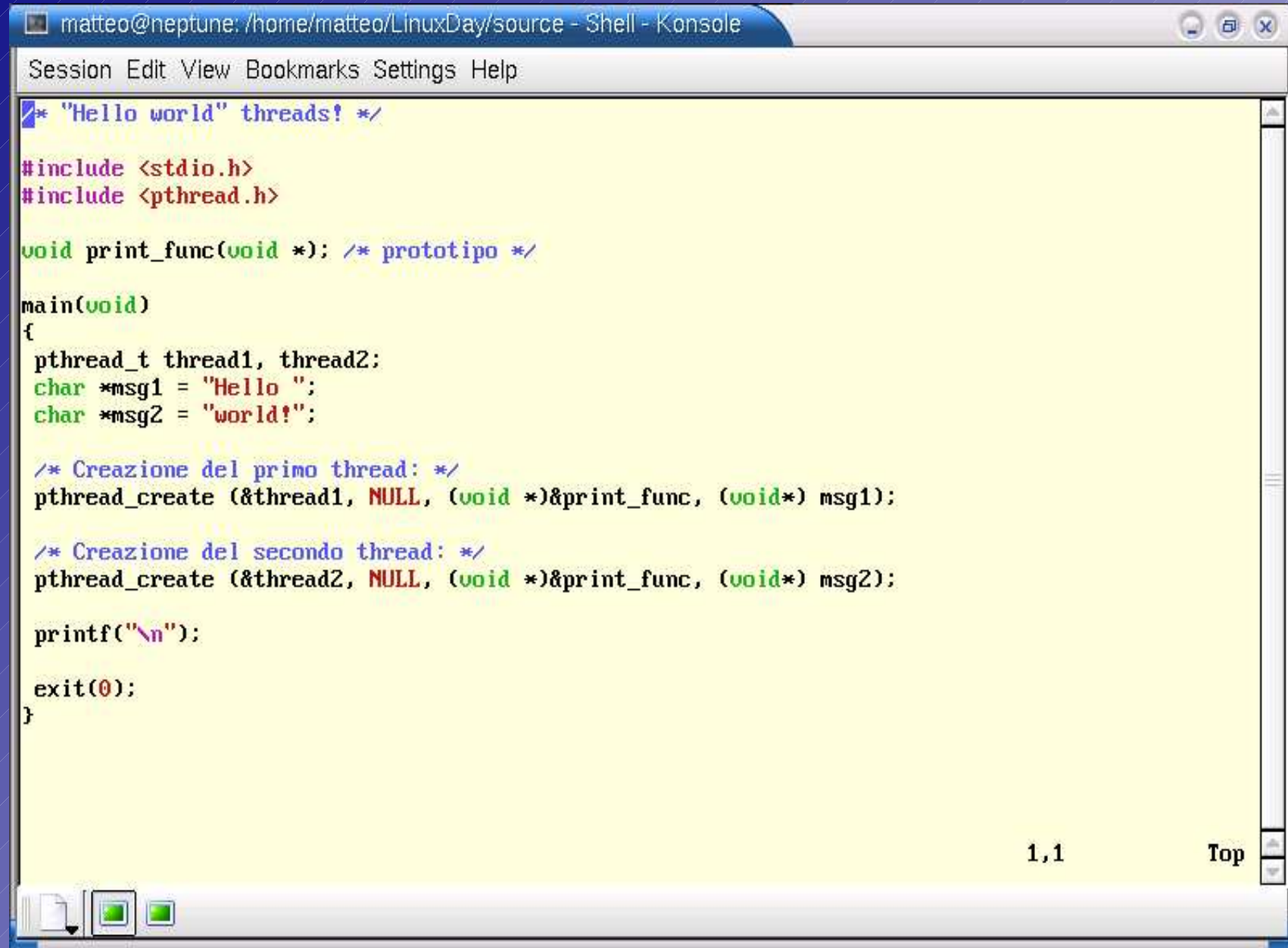
Gruppo Utenti Linux Cagliari h...?

Il primo programma

Il programma che vediamo esegue una stampa sullo schermo di un stringa, creando due thread.

I threads ricevono come argomento una parola della stringa, la stampano con la funzione `printf()`, ed all'uscita della funzione terminano il loro compito.

Gruppo Utenti Linux Cagliari h...?



The image shows a screenshot of a Konsole window titled "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The window contains a C program that demonstrates thread creation using pthreads. The code is as follows:

```
/* "Hello world" threads! */

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

void print_func(void *); /* prototipo */

main(void)
{
    pthread_t thread1, thread2;
    char *msg1 = "Hello ";
    char *msg2 = "world!";

    /* Creazione del primo thread: */
    pthread_create (&thread1, NULL, (void *)&print_func, (void*) msg1);

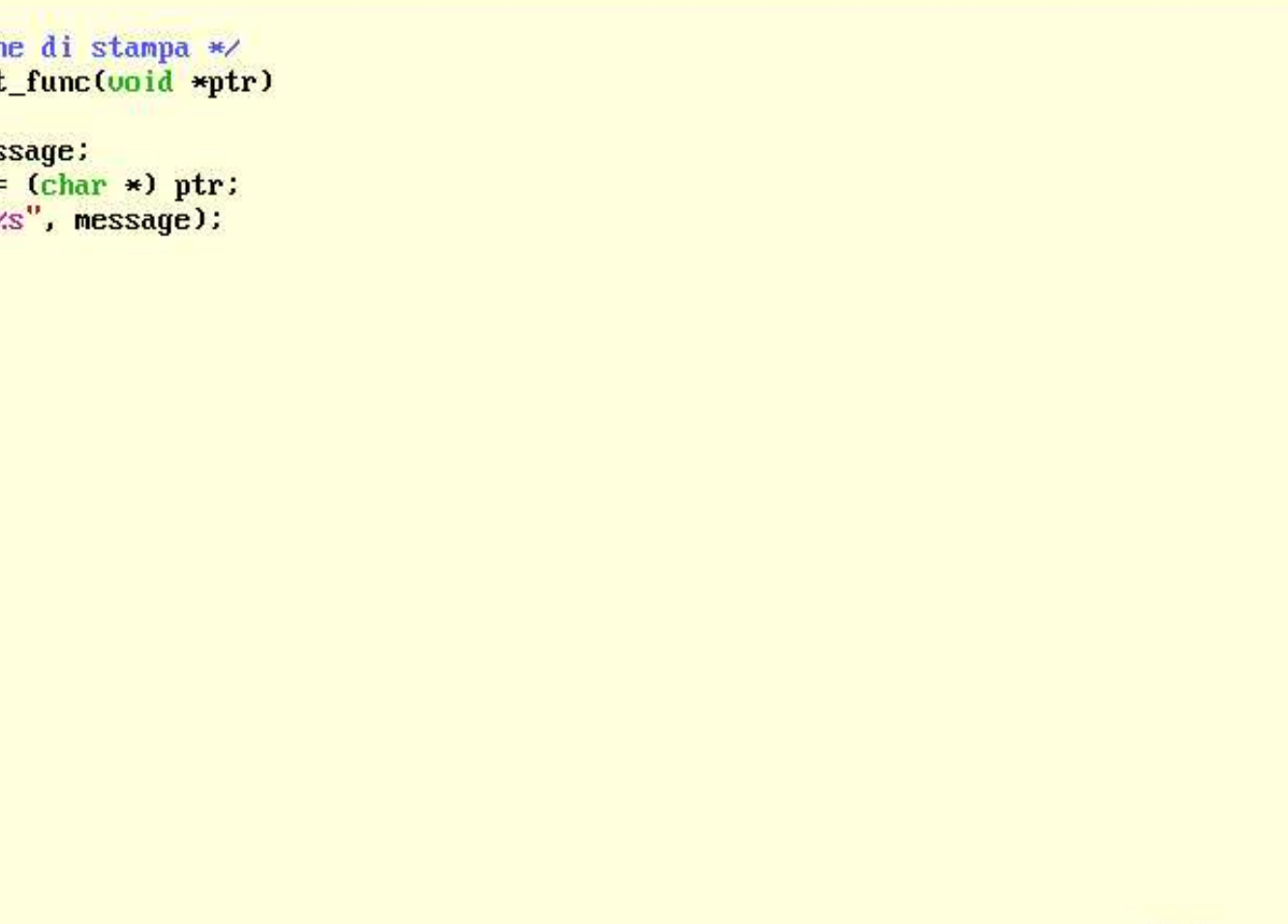
    /* Creazione del secondo thread: */
    pthread_create (&thread2, NULL, (void *)&print_func, (void*) msg2);

    printf("\n");

    exit(0);
}
```

The code is color-coded: comments are in blue, preprocessor directives in purple, function declarations in green, and variable/constant declarations in red. The main function is in black. The window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The status bar at the bottom shows "1,1" and "Top".

Gruppo Utenti Linux Cagliari h...?



```
matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole
Session Edit View Bookmarks Settings Help

/* Funzione di stampa */
void print_func(void *ptr)
{
    char *message;
    message = (char *) ptr;
    printf("%s", message);
}

36,1 Bot
```

Gruppo Utenti Linux Cagliari h...?

Compilazione

Da linea di comando lanciamo il compilatore:

```
gcc hellopth.c -o hello -lpthread
```

Il flag -l è una opzione per il linker che permette di collegare il programma con la libreria pthread.

Gruppo Utenti Linux Cagliari h...?

In esecuzione

Se da cmd line lanciamo il programma:

`./hello`

otteniamo l'output:

Hello world!

Notiamo che la funzione di partenza di entrambi i thread all'atto della creazione è la `print_func()`. Un thread giunge al termine quando lascia la sua funzione iniziale.

Gruppo Utenti Linux Cagliari h...?

Debolezze del codice

- I threads vengono eseguiti in modo concorrente. Nel caso precedente non esiste alcuna garanzia che il thread1 venga eseguito prima del thread2! Di conseguenza l'output potrebbe anche essere:
world! Hello
- Alla fine della funzione main() viene chiamata la funzione exit(). Se il 'main thread' esegue questa funzione prima dei 'child thread', non verrà stampato alcun output.

Gruppo Utenti Linux Cagliari h...?

Possibili rimedi

- Per eliminare il problema della precedenza nell'esecuzione, è possibile usare la funzione `sleep()`, ma essa si riferisce ai processi e dunque non funzionerebbe in modo corretto con i threads.
- Per evitare i problemi con la funzione `exit()`, è possibile utilizzare la `pthread_exit()`, che termina esplicitamente un thread, ma non l'intero processo.

Gruppo Utenti Linux Cagliari h...?

Thread joining

Nel programma precedente il problema della sincronizzazione non trova una soluzione soddisfacente. Ci serviremo allora dell'operazione di joining di un thread.

Questa operazione consiste nel bloccare il thread chiamante fino a che il thread (ThID) specificato non termina.

Gruppo Utenti Linux Cagliari h...?

Uso di pthread_join()

Rifacendoci al programma precedente possiamo aggiungere in coda alla main():

.....

```
pthread_join(thread1, NULL);  
pthread_join(thread2, NULL);
```

Il secondo argomento della funzione può, eventualmente, contenere il valore di ritorno del thread.

Gruppo Utenti Linux Cagliari h...?

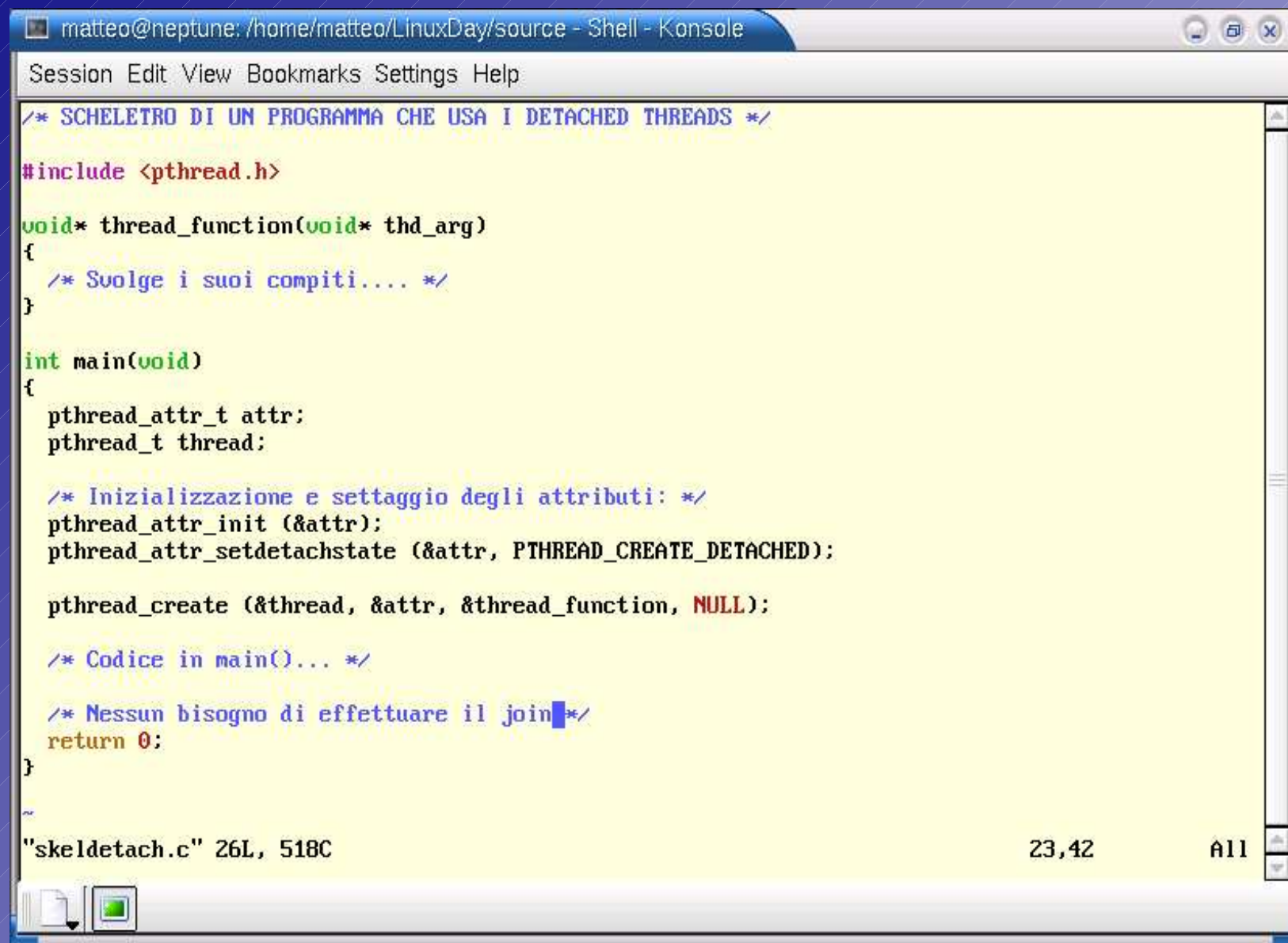
Attributi di un thread

Nella maggior parte delle applicazioni GNU/Linux, i threads che vengono creati sono di due tipi:

Joinable threads: non vengono rimossi quando il loro compito termina, ma aspettano che un altro thread chiami la `pthread_join()`.

Detached threads: vengono rimossi istantaneamente quando terminano. Non è possibile effettuare una operazione di join con un thread di questo genere.

Gruppo Utenti Linux Cagliari h...?



The image shows a screenshot of a Konsole window titled "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The window contains a C program named "skeldetach.c". The code is as follows:

```
/* SCHELETRO DI UN PROGRAMMA CHE USA I DETACHED THREADS */  
  
#include <pthread.h>  
  
void* thread_function(void* thd_arg)  
{  
    /* Svolge i suoi compiti.... */  
}  
  
int main(void)  
{  
    pthread_attr_t attr;  
    pthread_t thread;  
  
    /* Inizializzazione e settaggio degli attributi: */  
    pthread_attr_init (&attr);  
    pthread_attr_setdetachstate (&attr, PTHREAD_CREATE_DETACHED);  
  
    pthread_create (&thread, &attr, &thread_function, NULL);  
  
    /* Codice in main()... */  
  
    /* Nessun bisogno di effettuare il join */  
    return 0;  
}  
~  
"skeldetach.c" 26L, 518C
```

The status bar at the bottom of the window shows "23,42" and "A11".

Gruppo Utenti Linux Cagliari h...?

Stati di un thread

- **READY**: il thread è pronto per essere eseguito, ma si trova in attesa di un processore, oppure può essere appena stato sbloccato o essere stato “interrotto” da un altro thread.
- **RUNNING**: il thread è in esecuzione.
- **BLOCKED**: il thread non è in esecuzione perchè sta aspettando il completamento di qualche operazione o la disponibilità di una risorsa.

Gruppo Utenti Linux Cagliari h...?

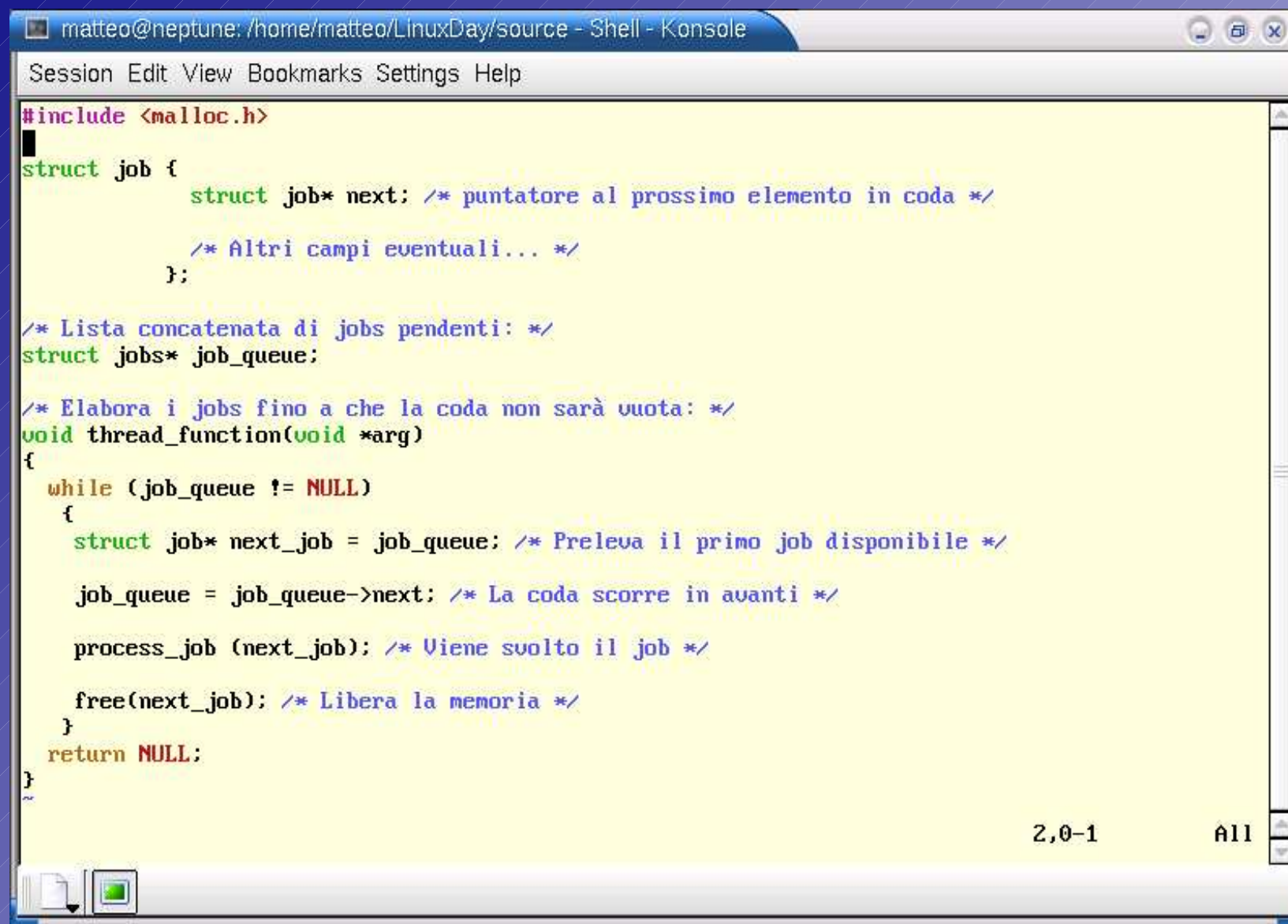
Race conditions

E' impossibile conoscere con esattezza quando un determinato thread verrà schedulato.

La possibilità di condividere i dati facilmente è anche una delle maggiori cause di errore.

Programmi che funzionano solo se un thread viene eseguito prima o più spesso di altri si dice che contengano delle race conditions.

Gruppo Utenti Linux Cagliari h...?



The image shows a terminal window titled "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The window contains C code for a linked list. The code defines a `struct job` with a `next` pointer, a `job_queue` pointer, and a `thread_function` that processes jobs from the queue. The code is color-coded: keywords in green, comments in blue, and other identifiers in black. The terminal window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The status bar at the bottom shows "2,0-1" and "All".

```
matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole
Session Edit View Bookmarks Settings Help

#include <malloc.h>

struct job {
    struct job* next; /* puntatore al prossimo elemento in coda */

    /* Altri campi eventuali... */
};

/* Lista concatenata di jobs pendenti: */
struct jobs* job_queue;

/* Elabora i jobs fino a che la coda non sarà vuota: */
void thread_function(void *arg)
{
    while (job_queue != NULL)
    {
        struct job* next_job = job_queue; /* Preleva il primo job disponibile */

        job_queue = job_queue->next; /* La coda scorre in avanti */

        process_job (next_job); /* Viene svolto il job */

        free(next_job); /* Libera la memoria */
    }
    return NULL;
}
```

Gruppo Utenti Linux Cagliari h...?

Supponiamo che...

Se due thread finiscono il proprio lavoro, più o meno nello stesso momento, e rimane un solo job da elaborare, può accadere che:

- 1) Il thread 1 verifica se la coda è vuota. Non lo è, quindi inizia il lavoro, ma viene bloccato dal kernel.
- 2) Il thread 2 inizia il proprio lavoro e, trovando la coda non vuota, elabora anch'esso il job.

Ci sono due thread che elaborano lo stesso job!

Gruppo Utenti Linux Cagliari h...?

Uno per volta

Ciò di cui abbiamo bisogno è che le operazioni di controllo della consistenza della coda siano rese *atomiche*.

Ovvero: è impossibile metterle in pausa oppure interromperle. Per far questo ci avvarremo del supporto fornito dal sistema operativo.

Gruppo Utenti Linux Cagliari h...?

MUTual Exclusion locks

Pensiamo ai *mutex* come a delle serrature: il primo thread che ha accesso alla coda dei lavori lascia fuori gli altri thread, fino a che non ha portato a termine il suo compito.

I threads piazzano un mutex nelle sezioni di codice nelle quali vengono condivisi i dati.

Gruppo Utenti Linux Cagliari h...?

pthread_mutex

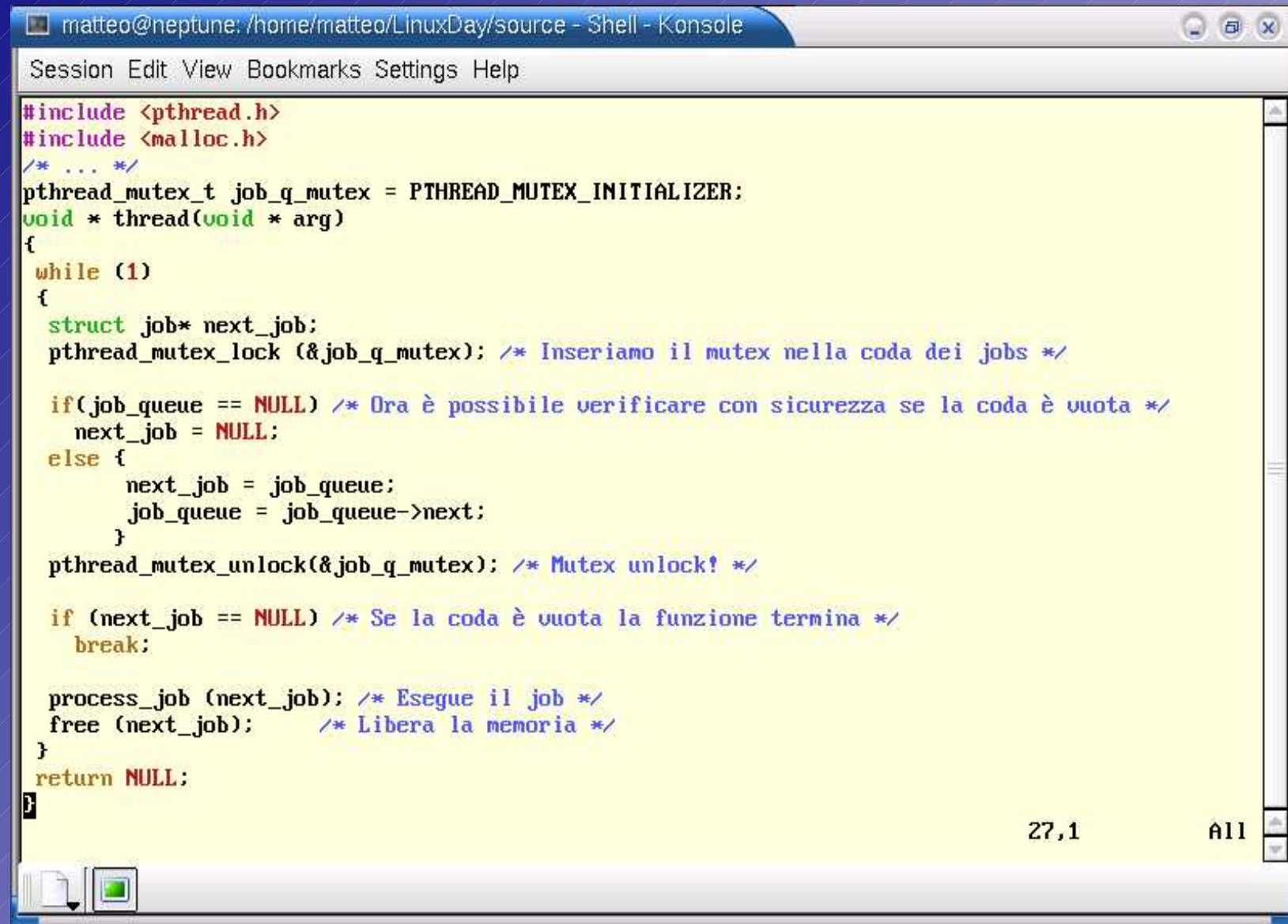
Per creare un mutex è necessario usare una variabile di tipo `pthread_mutex_t` e passarla come puntatore a `pthread_mutex_init`:

```
pthread_mutex_t mutex;  
pthread_mutex_init (&mutex, NULL);
```

Oppure con:

```
pthread_mutex_t mutex =  
    PTHREAD_MUTEX_INITIALIZER
```

Gruppo Utenti Linux Cagliari h...?



The image shows a screenshot of a Konsole terminal window. The title bar reads "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The menu bar includes "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The main text area contains C code for a thread function. The code includes headers for pthread.h and malloc.h, initializes a mutex, and enters a loop where it processes jobs from a queue. Comments in Italian explain the steps: inserting the mutex into the job queue, checking if the queue is empty, dequeuing a job, executing it, and freeing its memory. The code ends with a return statement. The status bar at the bottom right shows "27,1" and "All".

```
matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole
Session Edit View Bookmarks Settings Help

#include <pthread.h>
#include <malloc.h>
/* ... */
pthread_mutex_t job_q_mutex = PTHREAD_MUTEX_INITIALIZER;
void * thread(void * arg)
{
    while (1)
    {
        struct job* next_job;
        pthread_mutex_lock (&job_q_mutex); /* Inseriamo il mutex nella coda dei jobs */

        if(job_queue == NULL) /* Ora è possibile verificare con sicurezza se la coda è vuota */
            next_job = NULL;
        else {
            next_job = job_queue;
            job_queue = job_queue->next;
        }
        pthread_mutex_unlock(&job_q_mutex); /* Mutex unlock! */

        if (next_job == NULL) /* Se la coda è vuota la funzione termina */
            break;

        process_job (next_job); /* Esegue il job */
        free (next_job);        /* Libera la memoria */
    }
    return NULL;
}
```

27,1 All

Gruppo Utenti Linux Cagliari h...?

Mutex deadlocks

Una situazione di deadlock si verifica quando uno o più thread sono bloccati aspettando un evento che non si verificherà mai.

Il fast mutex, creato di default sui sistemi GNU/Linux, è soggetto a questo tipo di problema.

Gruppo Utenti Linux Cagliari h...?

Tipi di mutex

Fast mutex: questo tipo di mutex può causare un deadlock quando un thread tenta di impostare lo stesso mutex due volte di seguito.

Recursive mutex: non causa deadlock. Il mutex ricorda quanti lock ha applicato e si sbloccherà solo dopo lo stesso numero di unlock.

Error check mutex: il secondo lock consecutivo di un mutex genera un errore (solo su Linux).

Gruppo Utenti Linux Cagliari h...?

Settare il tipo di mutex

Vediamo come creare un *mutex error checking*:

```
int type = PTHREAD_MUTEX_ERRORCHECK_NP
```

```
pthread_mutexattr_t attr;
```

```
pthread_mutex_t mutex;
```

```
pthread_mutexattr_init (&attr)
```

```
pthread_mutexattr_settype (&attr, type);
```

```
pthread_mutex_init (&mutex, &attr);
```

```
pthread_mutexattr_destroy(&attr);
```

Gruppo Utenti Linux Cagliari h...?

Semafori

Se i threads lavorano molto velocemente potrebbero svuotare la coda e terminare le loro operazioni prima che essa sia di nuova piena.

Ciò di cui abbiamo bisogno è un meccanismo che blocchi i threads fino a che nella coda non vengano immessi nuovi jobs. Useremo i *semafori*.

Gruppo Utenti Linux Cagliari h...?

Cosa sono i semafori?

I semafori sono contatori usati per sincronizzare le operazioni di thread multipli.

GNU/Linux garantisce, anche per queste primitive, l'atomicità. Quindi, ogni modifica o check del valore di un semaforo può essere effettuata senza sollevare race conditions.

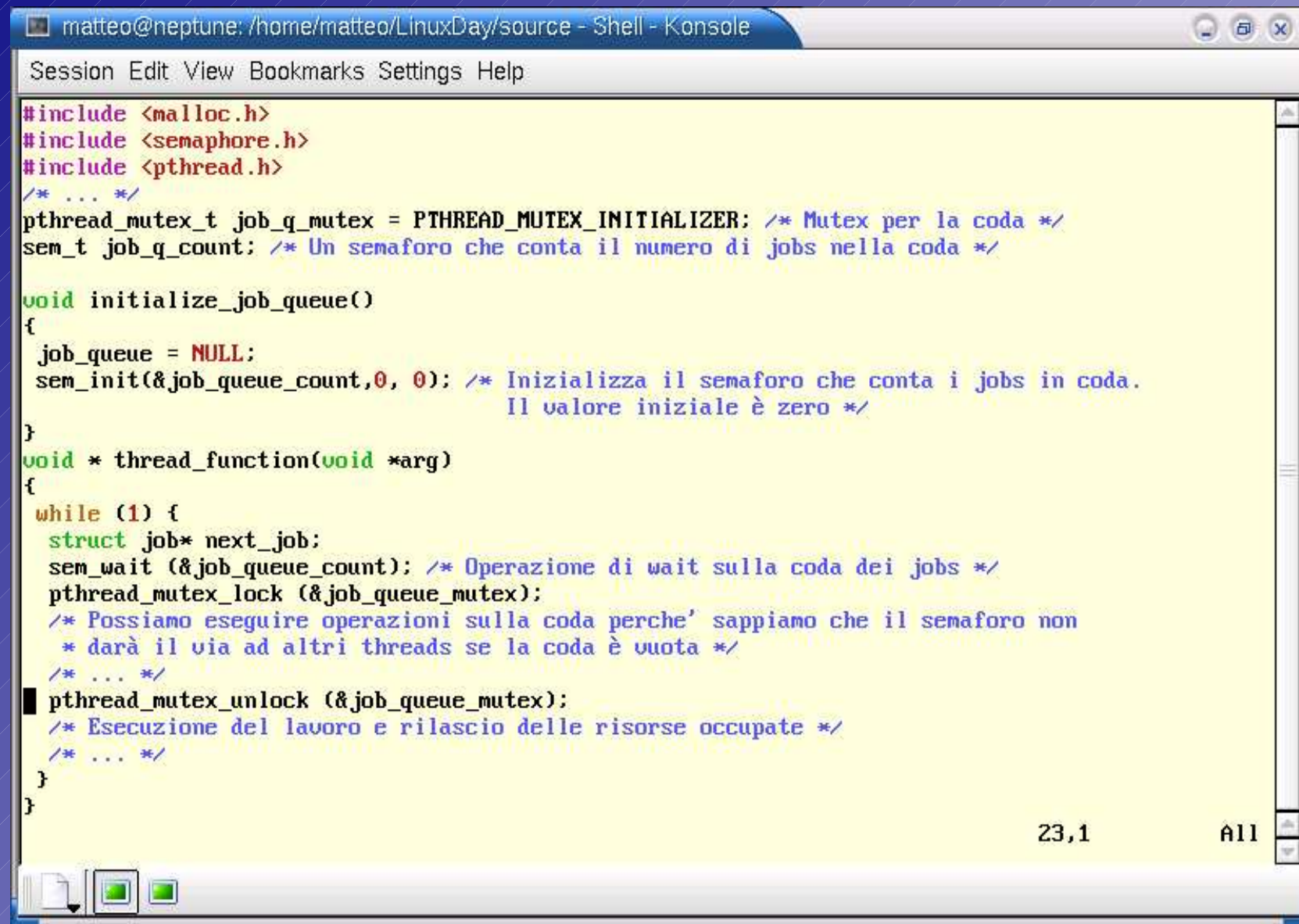
Gruppo Utenti Linux Cagliari h...?

Operazioni di base

Wait: decrementa di 1 il valore del semaforo. Se il valore è zero, l'operazione viene bloccata fino a che il valore non ridiviene positivo.

Post: incrementa il valore di 1 del semaforo. Se il valore precedente era zero, e vi sono threads in attesa, uno di questi viene sbloccato e la sua operazione di wait è completata.

Gruppo Utenti Linux Cagliari h...?

A screenshot of a Linux terminal window titled "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The main area displays C code for a job queue. The code includes headers for malloc, semaphore, and pthread. It defines a mutex and a semaphore for a job queue. A function "initialize_job_queue()" sets the queue to NULL and initializes the semaphore to 0. A "thread_function()" is defined as a while loop that waits on the semaphore, locks the mutex, processes a job, and then unlocks the mutex. The bottom right of the window shows "23,1" and "All".

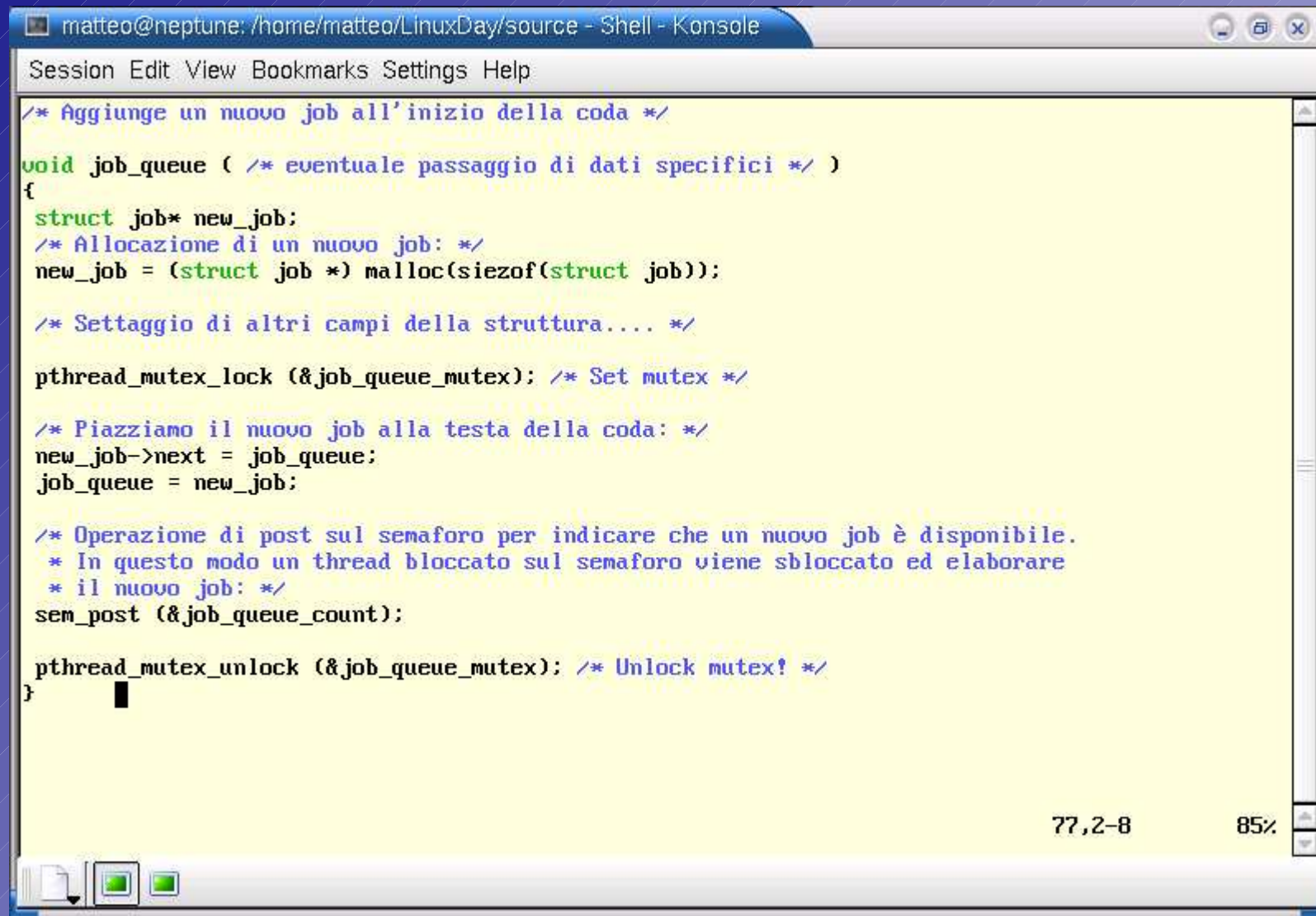
```
matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole
Session Edit View Bookmarks Settings Help

#include <malloc.h>
#include <semaphore.h>
#include <pthread.h>
/* ... */
pthread_mutex_t job_q_mutex = PTHREAD_MUTEX_INITIALIZER; /* Mutex per la coda */
sem_t job_q_count; /* Un semaforo che conta il numero di jobs nella coda */

void initialize_job_queue()
{
    job_queue = NULL;
    sem_init(&job_queue_count, 0, 0); /* Inizializza il semaforo che conta i jobs in coda.
                                     Il valore iniziale è zero */
}

void * thread_function(void *arg)
{
    while (1) {
        struct job* next_job;
        sem_wait (&job_queue_count); /* Operazione di wait sulla coda dei jobs */
        pthread_mutex_lock (&job_queue_mutex);
        /* Possiamo eseguire operazioni sulla coda perche' sappiamo che il semaforo non
         * darà il via ad altri threads se la coda è vuota */
        /* ... */
        pthread_mutex_unlock (&job_queue_mutex);
        /* Esecuzione del lavoro e rilascio delle risorse occupate */
        /* ... */
    }
}
```

Gruppo Utenti Linux Cagliari h...?

A screenshot of a terminal window titled "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The main area displays C code for a job queue. The code includes comments in Italian and uses standard C and pthread functions. The code is as follows:

```
/* Aggiunge un nuovo job all'inizio della coda */  
  
void job_queue ( /* eventuale passaggio di dati specifici */ )  
{  
    struct job* new_job;  
    /* Allocazione di un nuovo job: */  
    new_job = (struct job *) malloc(sizeof(struct job));  
  
    /* Settaggio di altri campi della struttura.... */  
  
    pthread_mutex_lock (&job_queue_mutex); /* Set mutex */  
  
    /* Piazziamo il nuovo job alla testa della coda: */  
    new_job->next = job_queue;  
    job_queue = new_job;  
  
    /* Operazione di post sul semaforo per indicare che un nuovo job è disponibile.  
     * In questo modo un thread bloccato sul semaforo viene sbloccato ed elaborare  
     * il nuovo job: */  
    sem_post (&job_queue_count);  
  
    pthread_mutex_unlock (&job_queue_mutex); /* Unlock mutex! */  
}
```

The terminal window has a status bar at the bottom showing "77,2-8" and "85%". There are also some icons in the bottom left corner.

Gruppo Utenti Linux Cagliari h...?

Condition variables

Questa è la terza primitiva di sincronizzazione che riguarda i thread POSIX.

Attraverso essa è possibile implementare condizioni più complesse che i thread devono soddisfare per essere eseguiti. Linux garantisce che i threads bloccati su una condizione verranno sbloccati quando essa cambierà.

Gruppo Utenti Linux Cagliari h...?

Condition variables e mutex

Una variabile condizione non fornisce la mutua esclusione. C'è bisogno di un mutex per poter sincronizzare l'accesso ai dati.

Pthreads garantisce che le operazioni di unlock e di wait (o signal) vengano rese atomiche, in modo da evitare che si verifichino race conditions tra le due operazioni durante lo scheduling di due thread.

Gruppo Utenti Linux Cagliari h...?

Funzioni per le condition variables

Una variabile condizione è rappresentata da una istanza del tipo `pthread_cond_t`.

Inizializzazione:

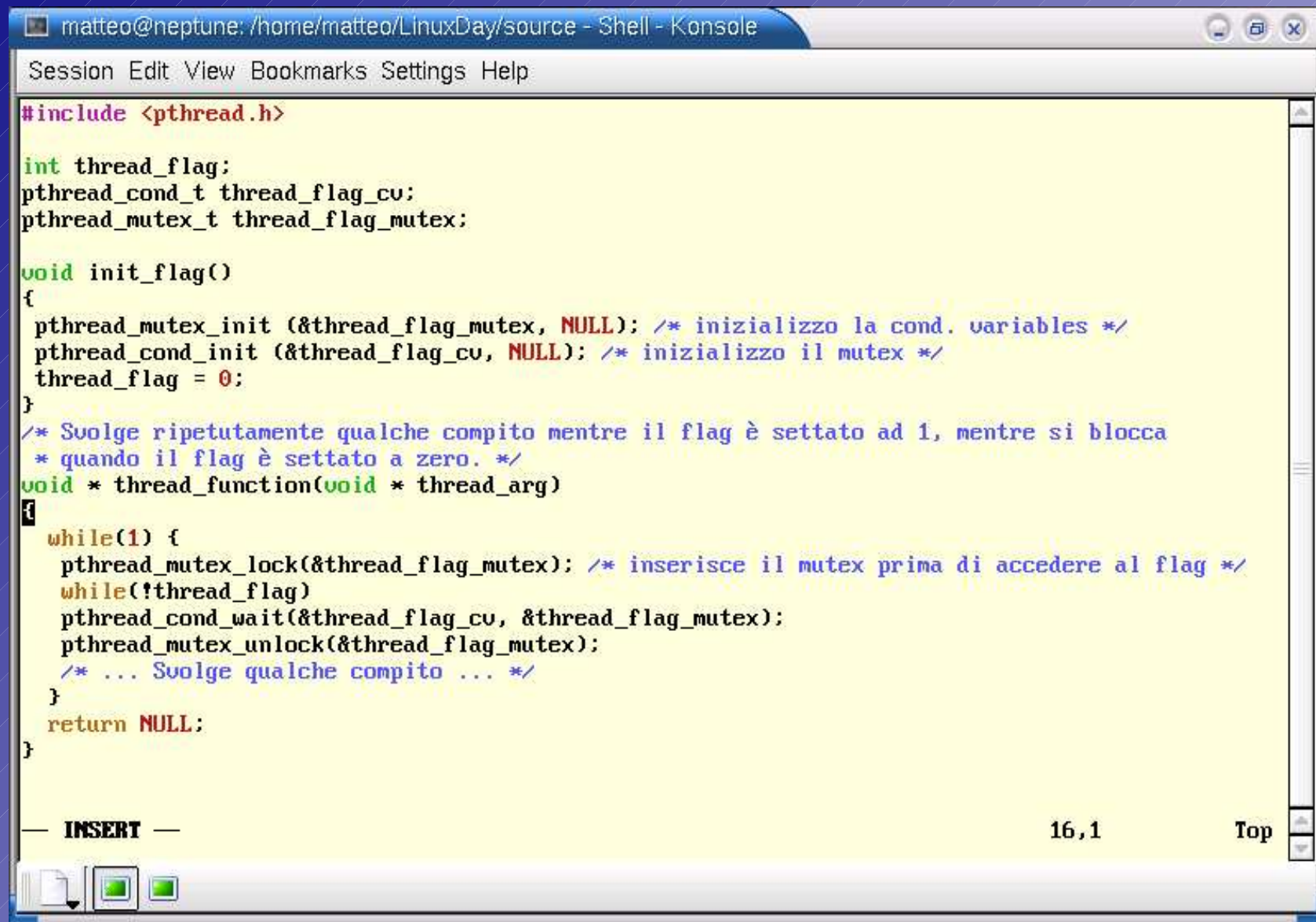
`pthread_cond_init(pthread_cond_t *COND, NULL)`

Linux non supporta attributi per le condition variables.

`Pthread_cond_signal()`: segnala una variabile condizione

`Pthread_cond_wait()`: blocca il thread chiamante finchè non la variabile condizione non è segnalata.

Gruppo Utenti Linux Cagliari h...?



The screenshot shows a terminal window titled "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The window contains a C program that uses pthreads to manage a shared flag. The code is as follows:

```
#include <pthread.h>

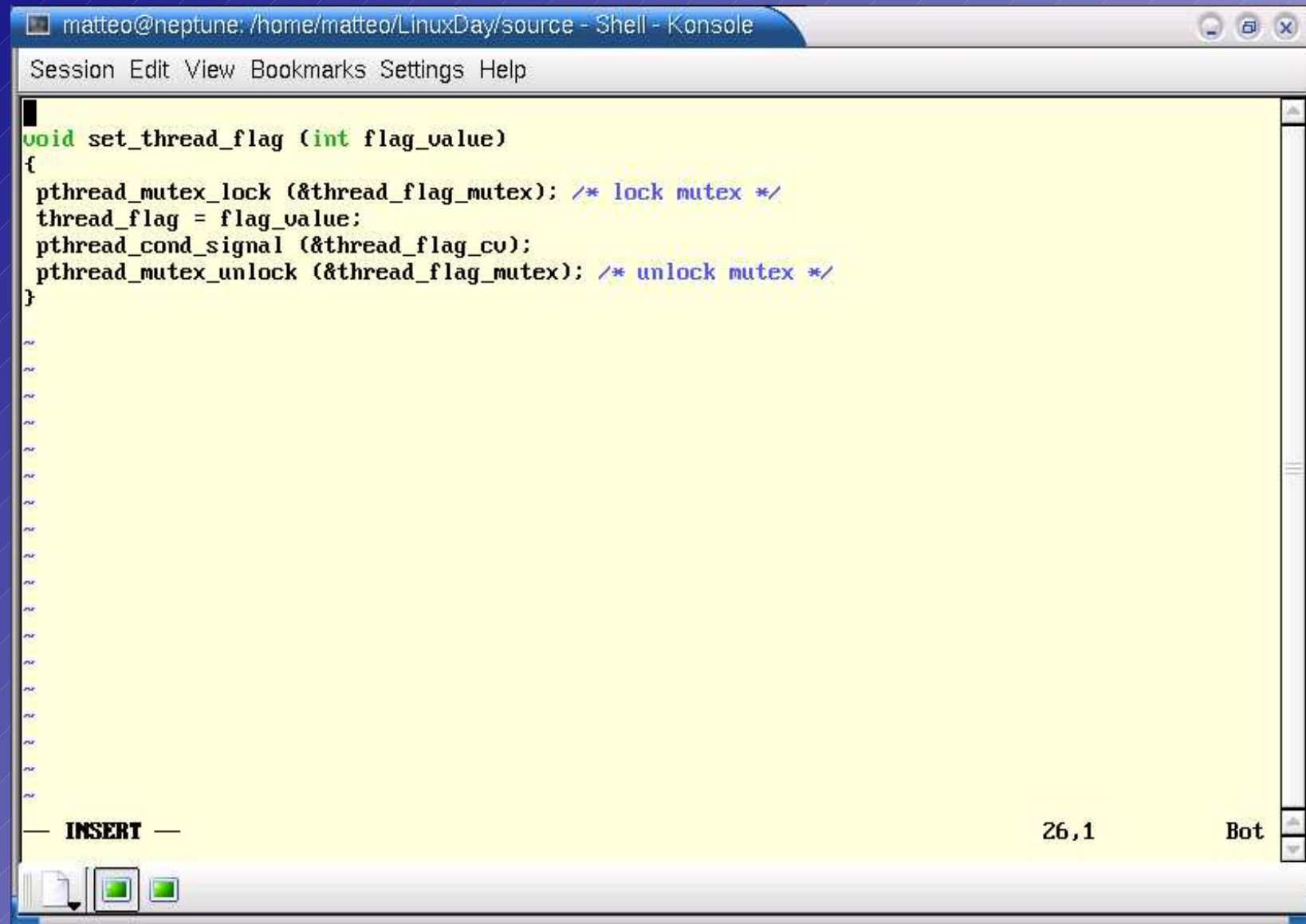
int thread_flag;
pthread_cond_t thread_flag_cv;
pthread_mutex_t thread_flag_mutex;

void init_flag()
{
    pthread_mutex_init (&thread_flag_mutex, NULL); /* inizializzo la cond. variables */
    pthread_cond_init (&thread_flag_cv, NULL); /* inizializzo il mutex */
    thread_flag = 0;
}

/* Svolge ripetutamente qualche compito mentre il flag è settato ad 1, mentre si blocca
 * quando il flag è settato a zero. */
void * thread_function(void * thread_arg)
{
    while(1) {
        pthread_mutex_lock(&thread_flag_mutex); /* inserisce il mutex prima di accedere al flag */
        while(!thread_flag)
            pthread_cond_wait(&thread_flag_cv, &thread_flag_mutex);
        pthread_mutex_unlock(&thread_flag_mutex);
        /* ... Svolge qualche compito ... */
    }
    return NULL;
}
```

At the bottom of the terminal window, there is a status bar with "INSERT" on the left, "16,1" in the center, and "Top" on the right. There are also some small icons on the far left of the status bar.

Gruppo Utenti Linux Cagliari h...?



The image shows a screenshot of a Konsole terminal window. The title bar reads "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The menu bar includes "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The main text area contains the following C code:

```
void set_thread_flag (int flag_value)
{
    pthread_mutex_lock (&thread_flag_mutex); /* lock mutex */
    thread_flag = flag_value;
    pthread_cond_signal (&thread_flag_cv);
    pthread_mutex_unlock (&thread_flag_mutex); /* unlock mutex */
}
```

Below the code, there are several lines of tilde (~) characters. At the bottom of the window, there is a status bar with "INSERT" on the left, "26,1" in the center, and "Bot" on the right. The bottom of the window also shows a taskbar with icons for a file manager, a terminal, and a web browser.

Gruppo Utenti Linux Cagliari h...?

Debugging

Lo standard Pthreads non definisce alcun sistema esplicito per il debug del codice.

Nel caso i threads si integrino con il s.o., può essere fornito uno strumento che permetta di listare e verificare lo stato dei thread, mutex, cond. variables.

Con implementazioni in user-space è improbabile che vi sia integrazione col debugger. Al massimo si può trovare un comando di stampa per le info.

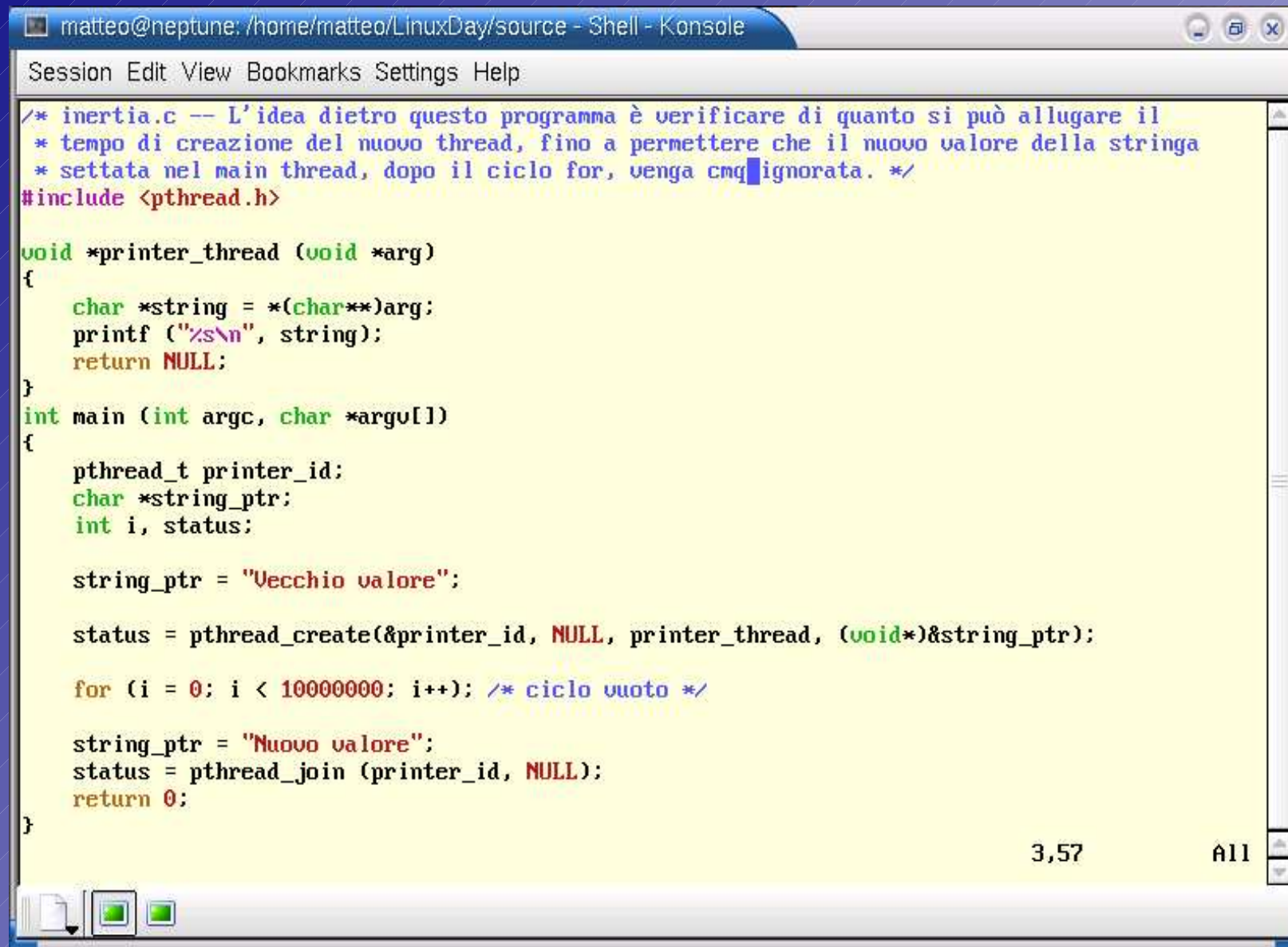
Gruppo Utenti Linux Cagliari h...?

I thread sono asincroni

Sebbene sia scontato, è bene non dimenticarlo mai. Tuttavia, considerando soprattutto le macchine che dispongono di un solo processore, non possiamo fare alcun tipo di previsione sulla velocità con la quale un nuovo thread viene creato e avviato.

Lo vediamo col programma seguente.

Gruppo Utenti Linux Cagliari h...?



```
matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole
Session Edit View Bookmarks Settings Help

/* inertia.c -- L'idea dietro questo programma è verificare di quanto si può allungare il
 * tempo di creazione del nuovo thread, fino a permettere che il nuovo valore della stringa
 * settata nel main thread, dopo il ciclo for, venga cmq ignorata. */
#include <pthread.h>

void *printer_thread (void *arg)
{
    char *string = *(char**)arg;
    printf ("%s\n", string);
    return NULL;
}

int main (int argc, char *argv[])
{
    pthread_t printer_id;
    char *string_ptr;
    int i, status;

    string_ptr = "Vecchio valore";

    status = pthread_create(&printer_id, NULL, printer_thread, (void*)&string_ptr);

    for (i = 0; i < 10000000; i++); /* ciclo vuoto */

    string_ptr = "Nuovo valore";
    status = pthread_join (printer_id, NULL);
    return 0;
}

3,57 All
```

Gruppo Utenti Linux Cagliari h...?

Identificare le race conditions

I processori eseguono i threads a differenti velocità, dipendenti dal carico. Il timeslicing può interrompere un thread in qualunque punto e per un tempo variabile.

In questo modo ricreare le condizioni di una race conditions può essere impossibile.

E' preferibile assumere che:

“Threads will run in the most evil order possibile”

Gruppo Utenti Linux Cagliari h...?

Deadlocks

I deadlocks si verificano quando vi sono dei conflitti nell'uso della sincronizzazione. In questo caso il debugging evidenzia, dopo qualche tempo, che vi sono dei threads che semplicemente stanno fermi ad aspettare, per un tempo indefinito, una qualche risorsa che per varie cause non otterranno mai.

Gruppo Utenti Linux Cagliari h...?

Vantaggi del multithreading

- I threads sono molto più leggeri dei processi. Questo comporta un minor utilizzo delle risorse per crearli e per effettuare lo switch context.
- Dato che i threads condividono la stessa area di memoria è estremamente più facile scambiare i dati, rispetto all'uso delle primitive IPC.
- I threads dovrebbero essere usati per programmi che richiedono “parallelismo fine”.

Gruppo Utenti Linux Cagliari h...?

Svantaggi del multithreading

- La facilità nello scambio di dati può essere anche un punto debole: si possono avere race conditions sui dati condivisi.
- In generale è più difficile effettuare il debug a causa della non riproducibilità di molte situazioni di errore.
- Un thread errato può corrompere lo spazio di memoria condiviso, causando un SEGFAULT.

Gruppo Utenti Linux Cagliari h...?

Modelli di implementazione

- **1:1** Ogni thread è visto come un processo separato nel kernel. E' sempre lo scheduler che si occuperà di loro, esattamente come fa per i processi. Questa è l'implementazione supportata da GNU/Linux.
- **M:1** M threads in user space sono mappati in un solo thread in kernel-space.
- **M:N** M threads in user-space sono mappati in N threads in kernel-space. Sono presenti in AIX, Solaris, IRIX, True64.

Gruppo Utenti Linux Cagliari h...?

User-space vs Kernel-space

- **Kernel-space:** quando i thread sono implementati a questo livello, la commutazione di contesto fra threads avviene senza che l'applicazione ne sappia nulla.
- **User-space:** in questo caso la commutazione di contesto avviene ad opera della libreria applicativa che fornisce il supporto per i threads, senza che il kernel venga avvertito.

Gruppo Utenti Linux Cagliari h...?

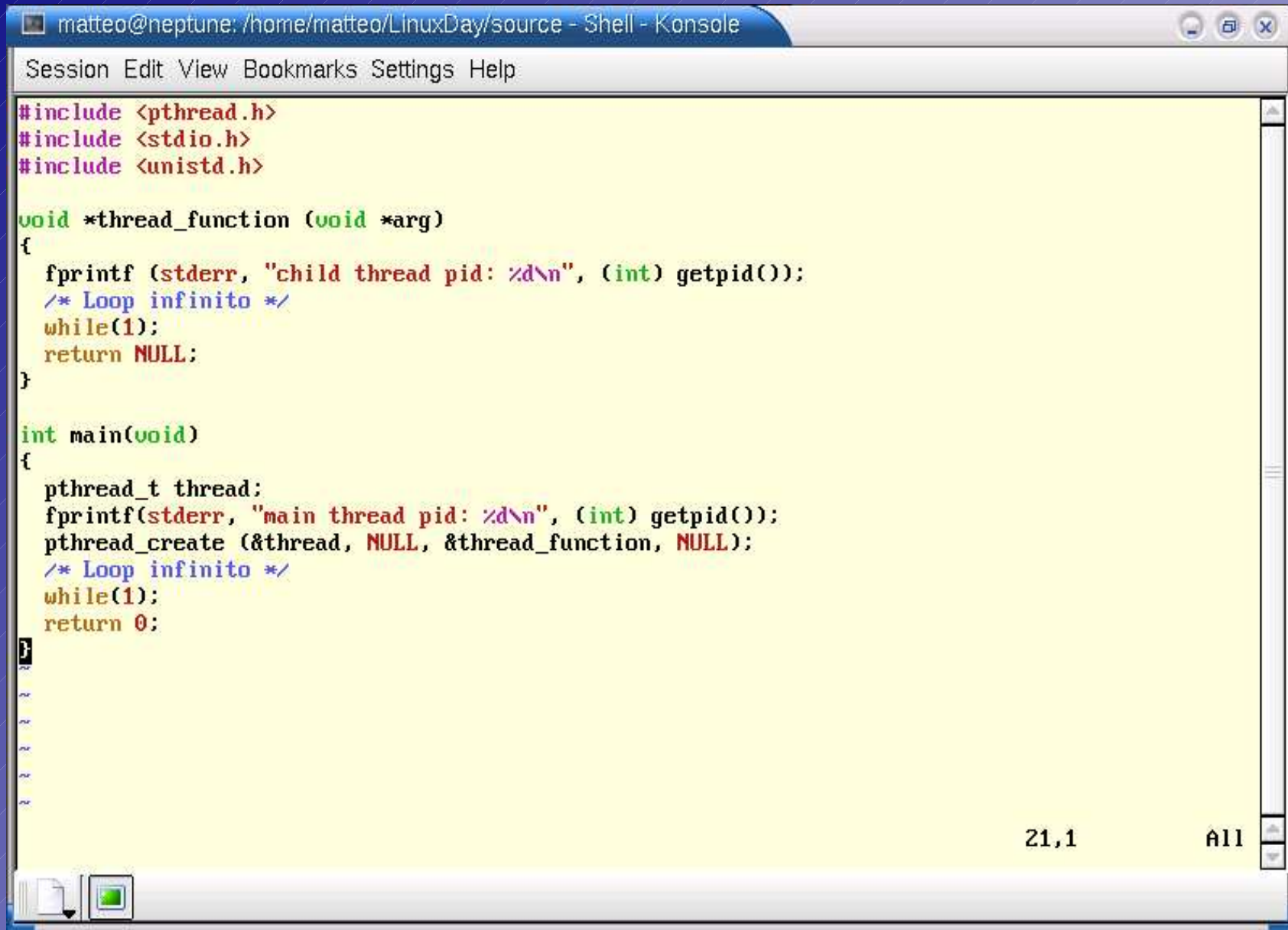
Il modello LinuxThreads

LinuxThreads è l'implementazione dei threads POSIX supportata da Linux a livello kernel.

In pratica, ogniqualvolta utilizziamo la chiamata *pthread_create*, quello che stiamo facendo è utilizzare la sys-call *__clone()*, una versione più generica della *fork()*.

Vediamo col programma seguente.

Gruppo Utenti Linux Cagliari h...?



The image shows a screenshot of a Konsole terminal window. The title bar reads "matteo@neptune: /home/matteo/LinuxDay/source - Shell - Konsole". The menu bar includes "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal content displays a C program with the following code:

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

void *thread_function (void *arg)
{
    fprintf (stderr, "child thread pid: %d\n", (int) getpid());
    /* Loop infinito */
    while(1);
    return NULL;
}

int main(void)
{
    pthread_t thread;
    fprintf(stderr, "main thread pid: %d\n", (int) getpid());
    pthread_create (&thread, NULL, &thread_function, NULL);
    /* Loop infinito */
    while(1);
    return 0;
}
```

At the bottom right of the terminal window, the text "21,1" and "All" is visible. The bottom status bar shows icons for file operations and a green square icon.

Gruppo Utenti Linux Cagliari h...?

Output di pidthd.c

Compiliamo il programma lanciando gcc:

```
gcc pidthd.c -o pid -lpthread
```

Lanciamo il programma da cmd line come:

```
./pid &
```

L'output sarà:

```
[1] 2092
```

```
main thread pid 2092
```

```
child thread pid 2094
```

Gruppo Utenti Linux Cagliari h...?

Output di ps

Lanciando da cmd line `ps -x` otteniamo:

....

2092	pts/2	R	1.46	./pid
------	-------	---	------	-------

2093	pts/2	S	0.00	./pid
------	-------	---	------	-------

2094	pts/2	R	1.46	./pid
------	-------	---	------	-------

Il thread 2093 è chiamato manager thread ed è parte dell'implementazione interna LinuxThreads. Viene creato la prima volta che il programma fa ricorso alla `pthread_create`.

Gruppo Utenti Linux Cagliari h...?

Signal handling & threads

Dato che su GNU/Linux i threads sono implementati come processi non esiste ambiguità nello spedire o ricevere segnali.

Di solito i segnali spediti dall'esterno al programma vengono inviati al main thread e da lì vengono ridistribuiti ai thread destinatari.

Gruppo Utenti Linux Cagliari h...?

Nuove librerie

- NPTL (Native POSIX Thread Library) di cui è possibile leggere un paper su:
(<http://people.redhat.com/drepper/nptl-design.pdf>)
a cura di Ulrich Drepper e Ingo Molnar.
- NGPL (Next Generation POSIX Library) è stata sviluppata presso IBM, ma il suo sviluppo si è interrotto alla release 2.2.2.
(www-124.ibm.com/developerworks/oss/pthreads/)

Gruppo Utenti Linux Cagliari h...?

Risorse utili

- Newsgroup sui threads e questioni correlate:
(comp.programming.threads)
- E' disponibile anche una FAQ del newsgroup:
(<http://w3imager.imag.fr/Membres/Eric.Ferley/pthreadsFAQ.html>)
- Tutorial sui thread POSIX (anche in PS):
(http://dis.cs.umass.edu/~wagner/threads_html/tutorial.html)

Gruppo Utenti Linux Cagliari h...?

Strumenti utilizzati

- Info sul GDB: (<http://sources.redhat.com/gdb>)
- Info specifiche su tutti gli strumenti GNU: (<http://www.gnu.org>)
- Lanciando da linea di comando:
- `info libc` (tutte le informazioni sulla lib C)
- `info gdb` (tutte le informazioni sul debugger)
- `info gcc` (tutte le informazioni sul compilatore)

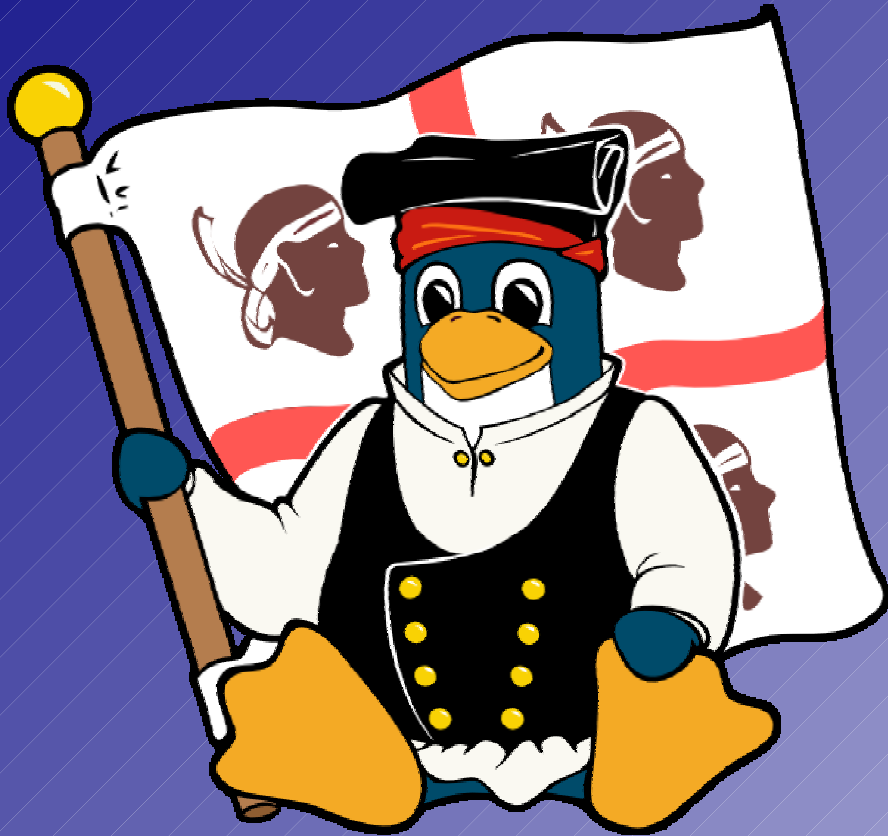
Gruppo Utenti Linux Cagliari h...?

Info su questo documento

- Per creare queste slides è stato usato OOImpress che è parte del pacchetto OpenOffice (ver. 1.0.2)
- Tutte le immagini e gli schemi sono in formato png e sono stati creati/modificati con KPaint.
- Per gli snapshot dello schermo è stato usato il programma Ksnapshot.
- Goppai, il logo GULCh, è stato creato da Ivana e Alceste Scalas.

Gruppo Utenti Linux Cagliari h...?

The end



- Grazie a tutti per l'attenzione.
- Ci si vede al LinuxDay 2004.
- Happy hacking!

Gruppo Utenti Linux Cagliari h...?