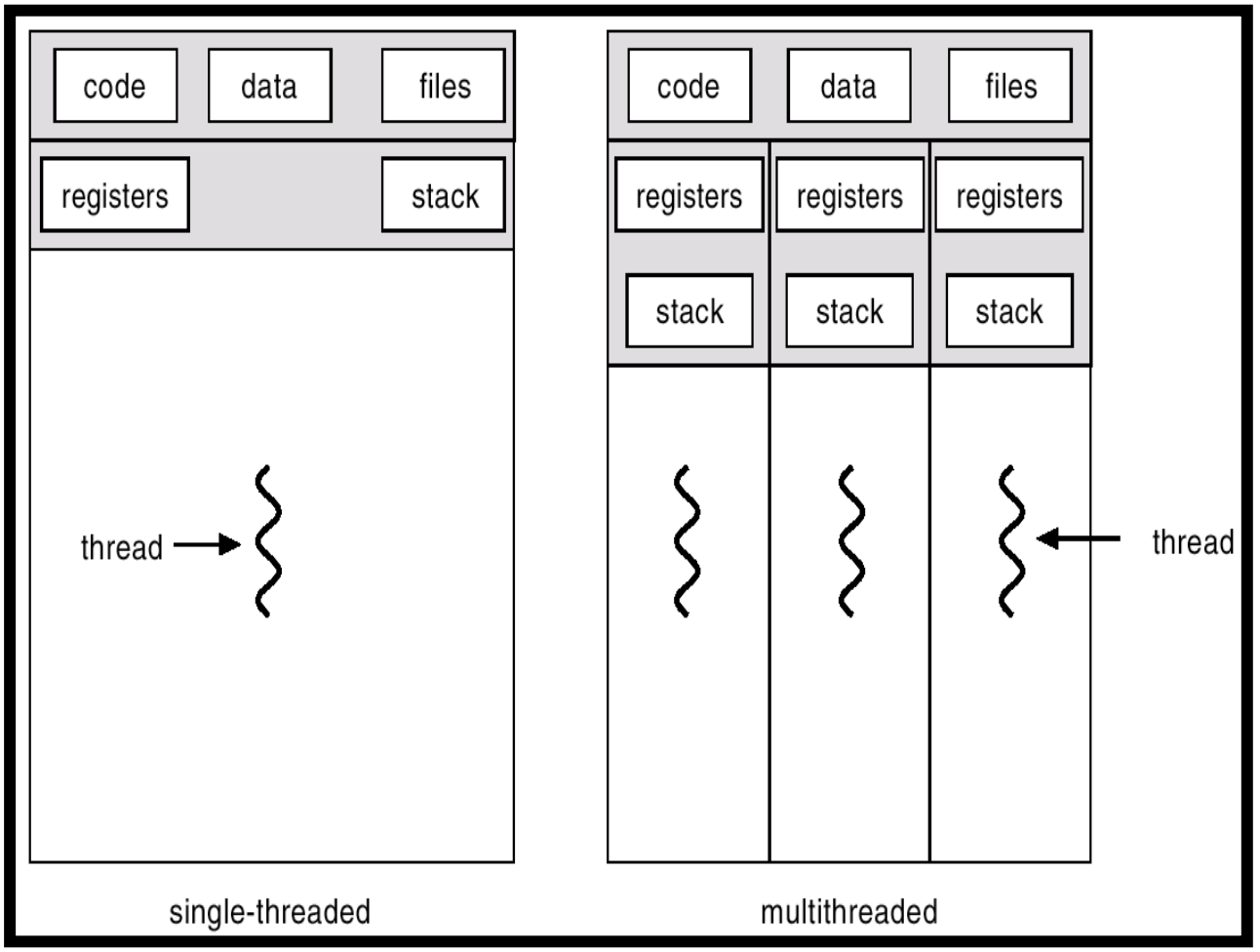
Introduzione ai POSIX Threads (libreria Pthread)

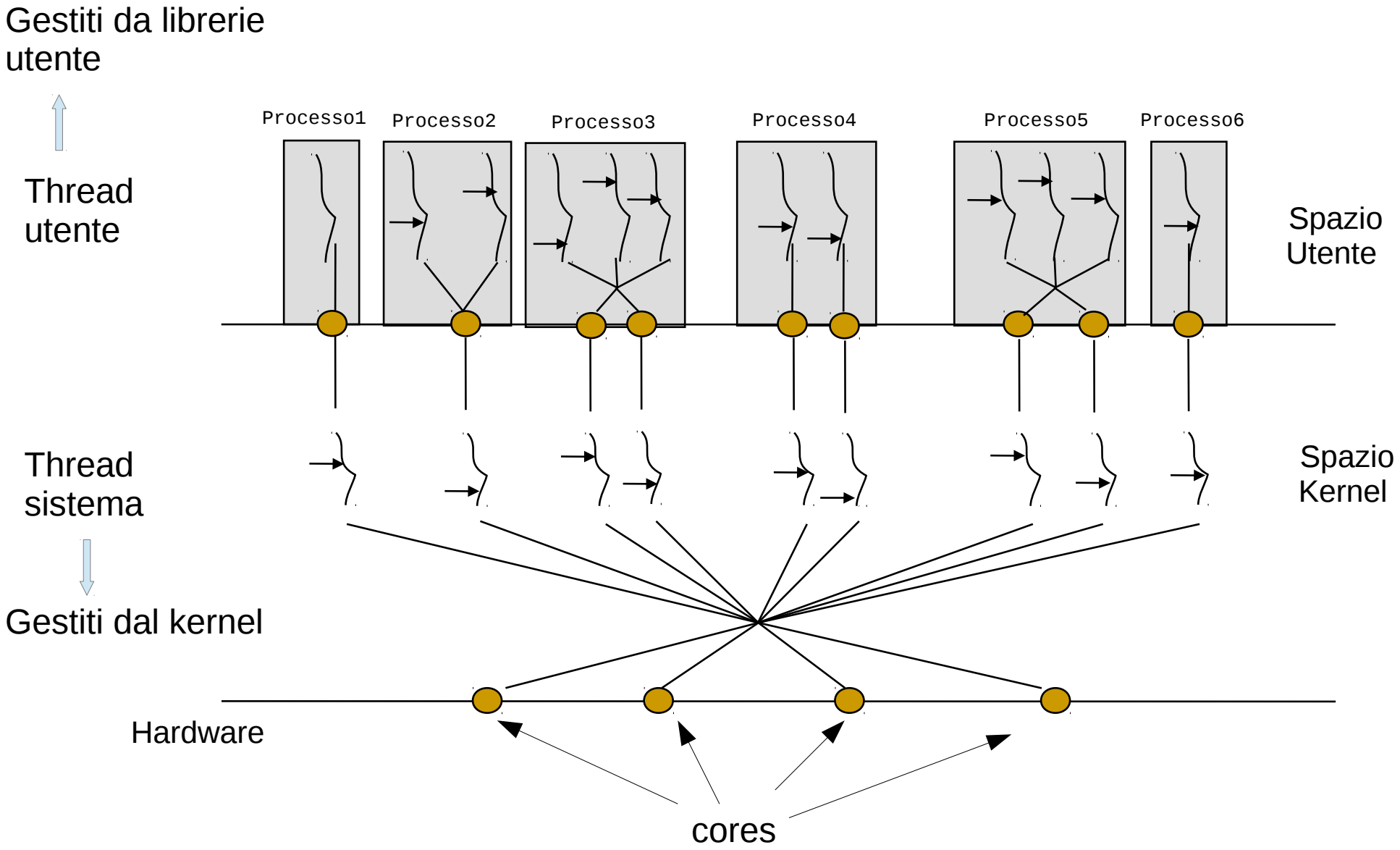
E.Mumolo, DIA

mumolo@units.it

Modello implementativo dei Thread



Architettura



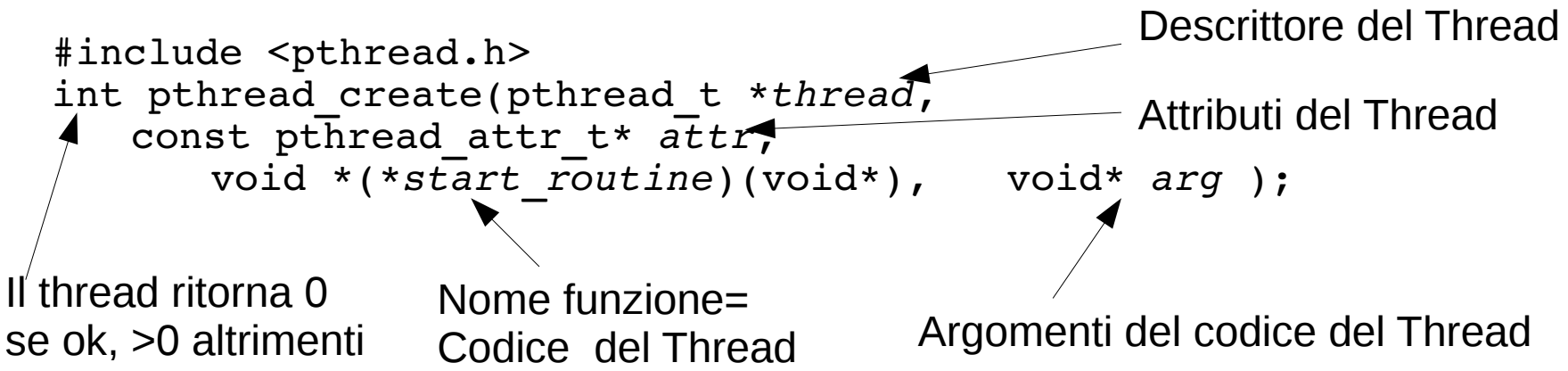
Implementazione dei thread

- POSIX (Portable Operating System Interface): famiglia di standard della IEEE Computer Society (IEEE Std 1003.1-1988)
- POSIX definisce le API, comandi a linea di comando e chiamate di sistema
- POSIX 1003.1c è la parte di POSIX che riguarda i thread.
- I thread sono forniti dalla maggioranza dei sistemi operativi
- Ma ciascuno ha una diversa interfaccia per l'uso delle funzioni associate ai thread
- I thread in Linux:
  - Presenti dalla versione 2.2
  - Chiamata di sistema clone: processo distinto che condivide lo spazio di indirizzi del processo chiamante

Info operative

- Il sorgente deve includere pthread.h:
  - `#include <pthread.h>`
- Librerie: in generale si usa gcc con opzione -lnome
- nome è il suffisso del nome della libreria → libnome.so
- libnome è il nome di una libreria. Possono essere:
  - Statiche (nome.a): collegate durante la compilazione
  - Dinamiche (nome.so): collegate in run-time → Shared objects
- Il sorgente deve essere compilato con la libreria pthread:
  - `gcc source.c -lpthread`
- libpthread è una libreria dinamica (libpthread.so)

Chiamate della libreria: *pthread_create*



- *thread* è il descrittore del thread. Contiene il thread_ID
- *attr* è il puntatore alla struttura delle caratteristiche del thread (*thread attribute object*). NULL = caratteristiche di default
- *start_routine* è il puntatore alla funzione che eseguirà il thread
- *arg* è il puntatore ai parametri eventualmente passati alla funzione.
- Codice d'errore: 0 se ok, maggiore di zero se errori
- Un thread termina quando:
    - chiama `pthread_exit()` – finisce - chiama `pthread_cancel()`
- Esempio:

```
pthread_create( &mythr, NULL, funzione, NULL )
```

Chiamate della libreria: *pthread_join*

Thread con il quale si sincronizza

```
#include <pthread.h>

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

- Questa funzione viene usata per bloccare il thread chiamante finchè il thread specificato con “thread” termina (o è già terminato).
  - Il puntatore “value_ptr” è usato per recuperare ogni valore d’uscita fornito dal thread terminato mediante la funzione pthread_exit.

```
int pthread_exit(void *value);
```

- Termina il thread tornando lo stato ad ogni thread che ne stava aspettando la terminazione con pthread_join

```
int pthread_kill(pthread_t thread, int sig);
```

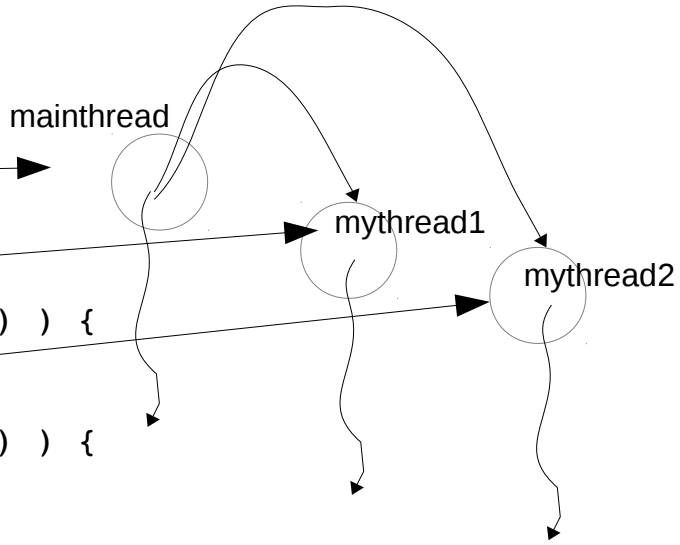
- Manda un segnale al thread specificato

```c
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
Float globale;
void *thread_function1(void *arg){
        int i; double x;
        for( i=0; i<5; i++) {
            x=(((float)rand())/RAND_MAX)*1e6;    usleep(x);
            printf(" Sono il thread1! \n");
        }
        return NULL;
}
void *thread_function2(void *arg){
        int i; double x;
        for( i=0; i<5; i++) {
            x=(((float)rand())/RAND_MAX)*1e6;    usleep(x);
            printf("Sono il thread2! \n");
        }
        return NULL;
}
int main(void) {
    srand(time(0));
    pthread_t mythread1, mythread2;
    if ( pthread_create( &mythread1, NULL, thread_function1, NULL) ) {
        printf("errore di creazione thread.");    abort();
    }
    if ( pthread_create( &mythread2, NULL, thread_function2, NULL) ) {
        printf("errore di creazione thread.");    abort();
    }
    if ( pthread_join ( mythread1, NULL ) ) {
        printf("errore di sincronizzazione thread.");    abort();
    }
    if ( pthread_join ( mythread2, NULL ) ) {
        printf("errore di sincronizzazione thread.");    abort();
    }
    exit(0);
}
```

Numero random

Ritardo in microsecondi

Duethr.c



Identificatori dei thread

- Ogni thread ha un ID unico
- Il thread ID può essere ottenuto con:

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

- Gli ID di due threads possono essere confrontati con:

```
int pthread_equal( pthread_t thread1, pthread_t thread2 );
```

idthread.c

ID del thread

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
void *thread_function1(void *arg){
          int i; double x;
          for( i=0; i<5; i++) {
               x=(((float)rand())/RAND_MAX)*1e6;    usleep(x);
               printf(" Sono il thread1 con ID=%lu! \n", pthread_self());
          }
          return NULL;
}
void *thread_function2(void *arg){
          int i; double x;
          for( i=0; i<5; i++) {
               x=(((float)rand())/RAND_MAX)*1e6;    usleep(x);
               printf("Sono il thread2 con ID=%lu! \n", pthread_self());
          }
          return NULL;
}
int main(void) {
     srand(time(0));
     pthread_t mythread1, mythread2;
     if ( pthread_create( &mythread1, NULL, thread_function1, NULL) ) {
          printf("errore di creazione thread.");    abort();
     }
     if ( pthread_create( &mythread2, NULL, thread_function2, NULL) ) {
          printf("errore di creazione thread.");    abort();
     }
     if ( pthread_join ( mythread1, NULL ) ) {
          printf("errore di sincronizzazione thread.");    abort();
     }
     if ( pthread_join ( mythread2, NULL ) ) {
          printf("errore di sincronizzazione thread.");    abort();
     }
     exit(0);
}
```

Passare parametri al pthread (I)

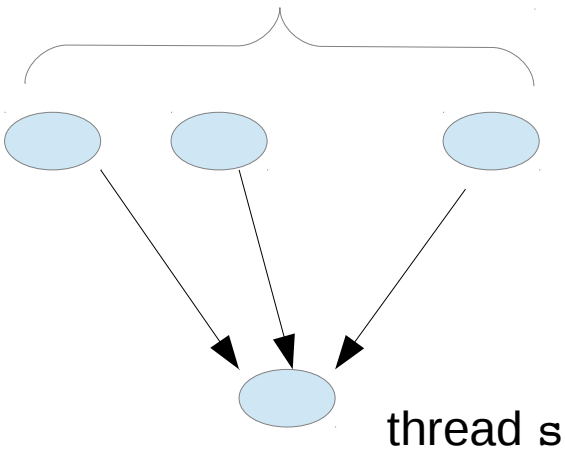
```
#include <pthread.h>
#include <stdio.h>

void thread1(int *arg)
{   printf("Sono il primo thread. Parametro = %d \n", *arg);}

void thread2(int *arg)
{   printf("Sono il secondo thread. Parametro = %d \n", *arg); }

main()
{
   pthread_t th1, th2;
   int i1 = 10, i2=20, n1, n2;

   n1=pthread_create(&th1, NULL, (void *)thread1, (void *)&i1);
   n2=pthread_create(&th2, NULL, (void *)thread2, (void *)&i2);

   printf("create ritornano %d %d\n",n1,n2);
}
```

- Attenzione: il casting puo' causare warning. Per esempio se void* ha dimensione 8 byte,

```
(int)void* oppure (void*)int  → warning
```

Sumthread.c

'dim' thread che eseguono inizializza



thread somma

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <sys/types.h>
#define Max_thread 1000
pthread_t tin[Max_thread],finale;
int vettore[Max_thread];

void *inizializza(void* arg) {
    long cont=(long)(arg);
    vettore[(long)arg]=cont;
}
void *somma(void* arg) {
    long s=0,ii;
    for(ii=0;ii<(long)arg;ii++) { s=s+vettore[ii]; }
    printf("\n La somma è = %ld \n",s);
}

main(){
    long dim,i; time_t t1,t2;
/*stampa la dimensione in byte di void* e di int*/
    printf("sizeof(void*)=%ld; sizeof(int)=%ld\n",sizeof(void*), sizeof(int));
/*chiede dim*/
    printf("dai la dimensione dell'array (max 1000) = "); scanf("%ld",&dim);
/*crea i thread*/
    for(i=0;i<dim;i++){ pthread_create(&tin[i],NULL,inizializza,(void*)i); }
/*aspetta la terminazione*/
    for(i=0;i<dim;i++){ pthread_join(tin[i],NULL); }
/thread somma*/
    pthread_create(&finale,NULL,somma,(void*)dim);
    pthread_join(finale,NULL);
}
```

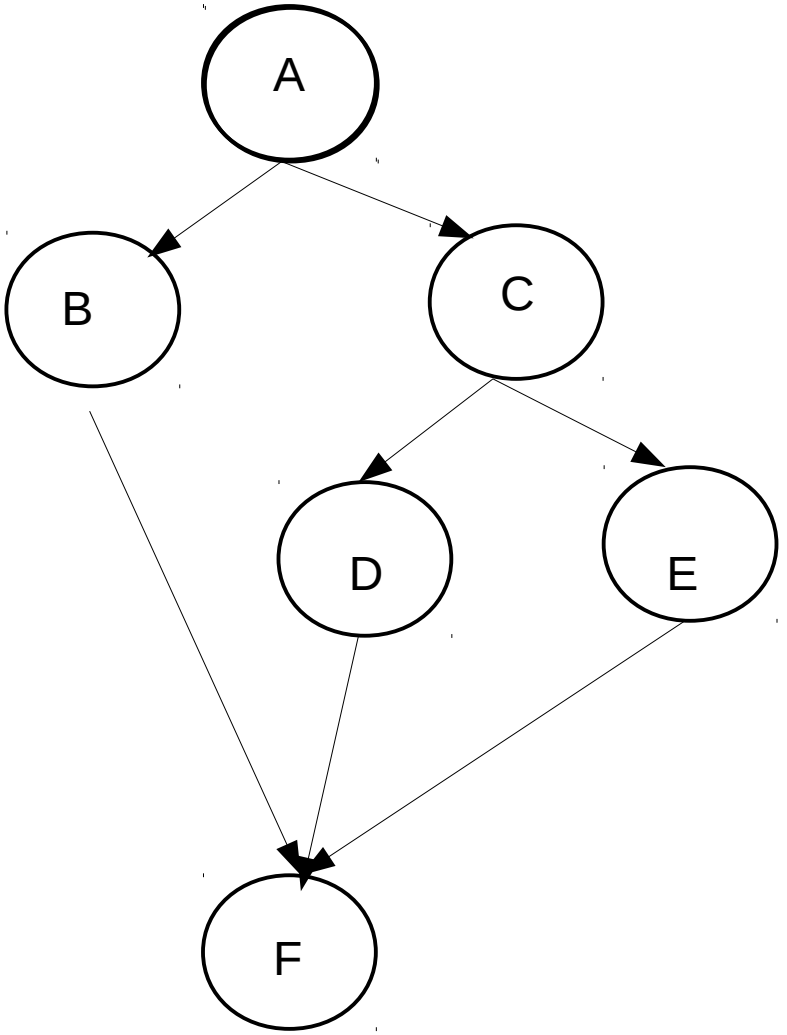
Passare parametri al pthread (II)

printhr.c

```
#include <pthread.h>
#include <stdio.h>
struct param{
    char character;    /*  i caratteri da stampare */
    int count;         /*  numero di volte */
}

void* stampa (void* parametri) {
    int i;
    struct param* p = (struct param*) parametri;

    for (i = 0; i < p->count; ++i) printf (“%c”,p->character);
    return NULL;
}

int main (){
    pthread_t thread1_id, thread2_id;
    int n1,n2;
    struct char_print_parms thread1_args, thread2_args;

    thread1_args.character = 'x';
    thread1_args.count = 300;
    n1=pthread_create (&thread1_id,NULL, stampa, thread1_args);/*stampa 300 x*/

    thread2_args.character = 'o';
    thread2_args.count = 200;
    n2=pthread_create (&thread2_id, NULL, stampa, thread2_args);/*stampa 200 o*/
    printf(“dal 1° thread %d. dal 2° thread %d\n”,n1,n2);
}
```

Passa l'indirizzo di una struttura statica

Implementazione di grafi di precedenza



graphthr.c

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

void *thread_functionA(void *arg){
        int i; double x=(((float)rand())/RAND_MAX)*1e6; usleep(x);
        printf("Sono il threadA ! \n"); return NULL;  }
void *thread_functionB(void *arg){
        int i; double x=(((float)rand())/RAND_MAX)*1e6; usleep(x);
        printf("Sono il threadB ! \n"); return NULL;  }
void *thread_functionC(void *arg){
        int i; double x=(((float)rand())/RAND_MAX)*1e6; usleep(x);
        printf("Sono il threadC ! \n"); return NULL;  }
void *thread_functionD(void *arg){
        int i; double x=(((float)rand())/RAND_MAX)*1e6; usleep(x);
        printf("Sono il threadD ! \n"); return NULL;  }
void *thread_functionE(void *arg){
        int i; double x=(((float)rand())/RAND_MAX)*1e6; usleep(x);
        printf("Sono il threadE ! \n"); return NULL;  }
void *thread_functionF(void *arg){
        int i; double x=(((float)rand())/RAND_MAX)*1e6; usleep(x);
        printf("Sono il threadF ! \n"); return NULL;  }

int main(void) {
    srand(time(0));
    pthread_t mythreadA, mythreadB, mythreadC, mythreadD, mythreadE, mythreadF;
    pthread_create( &mythreadA, NULL, thread_functionA, NULL); pthread_join ( mythreadA, NULL );
    pthread_create( &mythreadB, NULL, thread_functionB, NULL);
    pthread_create( &mythreadC, NULL, thread_functionC, NULL); pthread_join ( mythreadC, NULL );
    pthread_create( &mythreadD, NULL, thread_functionD, NULL);
    pthread_create( &mythreadE, NULL, thread_functionE, NULL);
    pthread_join(mythreadB,NULL);pthread_join(mythreadD,NULL); pthread_join ( mythreadE, NULL );
    pthread_create( &mythreadF, NULL, thread_functionF, NULL); pthread_join ( mythreadF, NULL );
    exit(0);
}
```

Introduzione alla sincronizzazione tra Thread. Semafori POSIX

- Operazioni:
  - Apertura
  - Inizializzazione
  - Operazione Down
  - Operazione Up

Creazione di un semaforo 'con nome'

```
sem_t *sem_open (char *name, int oflags,mode_t creation_mode,
                 unsigned init_val);
```

- Per creare un semaforo usare il flag O_CREAT, possibilmente in || con O_EXCL per ottenere la generazione di errori se il semaforo esiste già..
- creation_mode specifica i permessi d’accesso ad un semaforo creato.
- Generalmente si indica RWX per il gruppo di utenti desiderato (U, G o O).
  - Il parametro init_val viene usato per inizializzare il valore del semaforo.
  - Il valore ritornato è un puntatore al semaforo, o -1.

Creazione di un semaforo 'senza nome'

```
int sem_init (sem_t *sem, int pshared, unsigned value);
```

- Questa funzione crea un semaforo ‘senza nome’.
- Se il parametro pshared non è zero, allora il semaforo può essere condiviso tra i processi
- Il parametro value è usato per inizializzare il valore del semaforo.
- Questa funzione ritorna 0 o -1 se ha successo o no.

Chiusura

```
int sem_close (sem_t *sem);
```

- Questa funzione è usata per chiudere la connessione con un semaforo con nome, aperto con sem_open.
- I semafori con nome sono persistenti; cioè lo stato di un semaforo persiste anche se nessuno ha aperto il semaforo.
- Utilizzando il semaforo dopo che sia chiuso ha un effetto indefinito.

Lettura del semaforo

```
int sem_getvalue (sem_t *sem, int *value);
```

- Questa funzione viene usata per ottenere il valore di un semaforo con o senza nome.
- Valore di ritorno: puntatore value
- Il valore tornato è positivo se la risorsa controllata dal semaforo è sbloccata.
- Se il valore tornato è 0, allora la risorsa è bloccata.
- Alcune implementazioni ritornano un numero negativo per indicare che la risorsa è bloccata.

Operazione down

```
int sem_wait (sem_t *sem);
```

- Pseudocodice:

```
wait(S):
if(S<=0)
      aggiungi il descrittore del processo in coda su S;
else
      S--;
```

- Questa funzione cerca di decrementare il valore del semaforo identificato.
- Se ha avuto successo, il processo chiamante continua.
- Se il valore del semaforo è 0, il thread chiamante è bloccato
- La chiamata sem_wait della libreria Posix usa la chiamate di sistema ~semop(-1) del kernel

Operazione up

```
int sem_post (sem_t *sem);
```

- Pseudocodice:

```
signal(S):
if(c'e' un descrittore in attesa su S)
      esegui il processo;
else
S++;
```

- Questa funzione incrementa il valore del semaforo identificato.
- Se un processo è bloccato a causa di una chiamata sem_wait, il processo con priorità più alta che sta aspettando viene svegliato.
- La chiamata sem_post della libreria Posix usa la chiamate di sistema ~semop(+1) del kernel

Cancellazione di un semaforo

```
int sem_unlink (char *name);
```

- Questa funzione distrugge il semaforo con nome. Se altri processi hanno aperto il semaforo, possono continuare ad utilizzarlo finche' non lo chiudono con `sem_close`.

```
int sem_destroy (sem_t *sem);
```

- Questa funzione viene usata per cancellare un semaforo senza nome.
- Il semaforo che viene distrutto deve essere stato precedentemente inizializzato con `sem_init`.
- La distruzione di un semaforo sul quale altri processi sono bloccati causa il loro sblocco con un errore (errno = EINVAL).
- Usando un semaforo dopo che sia stato distrutto ha un effetto non definito.

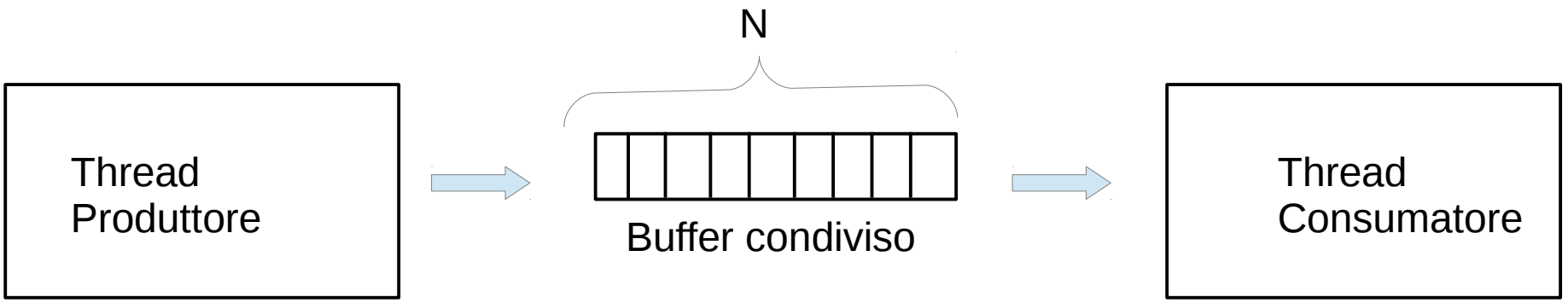
ccprinthr.c

```
/* primo esempio di utilizzo dei semafori nei thread: stampa concorrente */
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
sem_t mysem;

void *body(void *arg)
{
    int i,j;
    for (j=0; j<40; j++) {
        sem_wait(&mysem);
        for (i=0; i<1000000; i++); fprintf(stderr,(char *)arg);
        sem_post(&mysem);
    }
    return NULL;
}

int main()
{
    pthread_t t1,t2,t3;  pthread_attr_t myattr; int err;

    sem_init(&mysem,0,1);

    err = pthread_create(&t1, NULL, body, (void *)".");
    err = pthread_create(&t2, NULL, body, (void *)"#");
    err = pthread_create(&t3, NULL, body, (void *)"o");

    pthread_join(t1, NULL); pthread_join(t2, NULL); pthread_join(t3, NULL);

    printf("\n");
    return 0;
}
```

ccincthr.c

```
#include <stdio.h>     /* secondo esempio: incremento variabile condivisa */
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
int x = 0; /* variabili globali */
sem_t m;

void *thread(void *arg){   /* Thread function */
    sem_wait(&m); /* lock  m */
    x = x + 1;      /* sezione critica */
    sem_post(&m); /* unlock m */
}

void main (){
    pthread_t tid[10000];
    int i;
    if (sem_init(&m, 0, 1) == -1) {   /* inizializza il semaforo a 1 */
        printf("non posso inizializzare il semaforo");
        exit(2);
    }
    for (i=0; i<10000; i++)/* crea 10000 threads */
    {
        if (pthread_create(&tid[i], NULL, thread, NULL) < 0) {
            printf("Error: thread cannot be created");
            exit(1);
        }
    }
    for (i=0; i<10000; i++) pthread_join(tid[i], NULL);
    printf("valore finale di x = %d\n", x);
    exit(0);
}
```

Soluzione semaforica al problema produttore-consumatore



Schema:
3 semafori: 1 binario, mutex
1 contatore, empty inizializzato a N
1 contatore, full inizializzato a 0

Thread Produttore:

```
elp=produci();

down(empty) ;
down(mutex);
Inserisci elp;
up(mutex);
up(full)    ;
```

Thread Consumatore:

```
down(full);
down(mutex);
estrai elc;
up(mutex);
up(empty);
consuma(elc);
```

Thread Produttore

```
void *produci(){
  int i=0;
  char c;
  while (i<MAXITER){
       i++;
       c=(char)(random()%26)+97;

       sem_wait(&vuoto);
       sem_wait(&mutex);
       arr[in]=c;
       printf("Inserito %c\n",c);
       in = (in+1) % 10;
       sem_post(&mutex);
       sem_post(&pieno);
  }
}
```

Thread Consumatore

```
void *consuma(){
  int i=0;
  while (i<MAXITER){
       i++;

       sem_wait(&pieno);
       sem_wait(&mutex);
       char c = arr[out];
       printf("Prelevo %c\n",c);
       out = (out+1) % 10;
       sem_post(&mutex);
       sem_post(&vuoto);
  }
}
```

Main:

pcthr.c

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define MAXITER 100
sem_t mutex,pieno,vuoto;
char arr[10]; int in,out;

int main(){
        pthread_t prod,cons;
        pthread_attr_t attr;
        pthread_attr_init(&attr);

        sem_init(&mutex,0,1);
        sem_init(&pieno,0,0);
        sem_init(&vuoto,0,10);

        in=0; out=0; /*condizione iniziale*/

        pthread_create(&prod, NULL, produci, NULL);
        pthread_create(&cons, NULL, consuma, NULL);

        pthread_join(prod, NULL);
        pthread_join(cons, NULL);
        return 0;
}
```