Esempio: la Toilette Unisex

Si consideri la toilette di un ristorante.
La toilette è unica per uomini e donne.

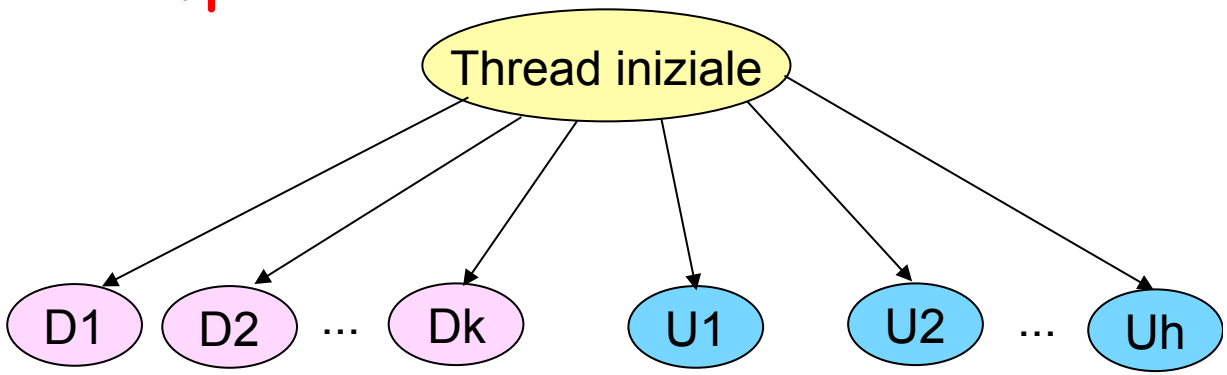
Utilizzando i semafori forniti dalla libreria LinuxThreads, si realizzi un'applicazione concorrente nella quale ogni utente della toilette (uomo o donna) è rappresentato da un processo e il bagno come una risorsa.

La politica di sincronizzazione tra i processi dovrà garantire che:
- nella toilette non vi siano contemporaneamente uomini e donne
- nell'accesso alla toilette, le donne abbiano la priorità sugli uomini.

Si supponga che la toilette abbia una capacità limitata a N persone.

È possibile realizzare una soluzione a questo problema utilizzando i mutex della libreria pthreads?

Impostazione



Quali thread?

- thread iniziale
- **Donne**: D1, D2,...Dk
- **Uomini**: U1,U2,..Uh

Quali risorse comuni?

- Toilette = pool di *risorse* (posti in bagno) **equivalenti**

➔introduciamo un gestore della toilette:

```
typedef struct{
/*      dati condivisi;
        strumenti di sincronizzazione
        (semafori & mutex)
*/
}gestore_toilet;
```

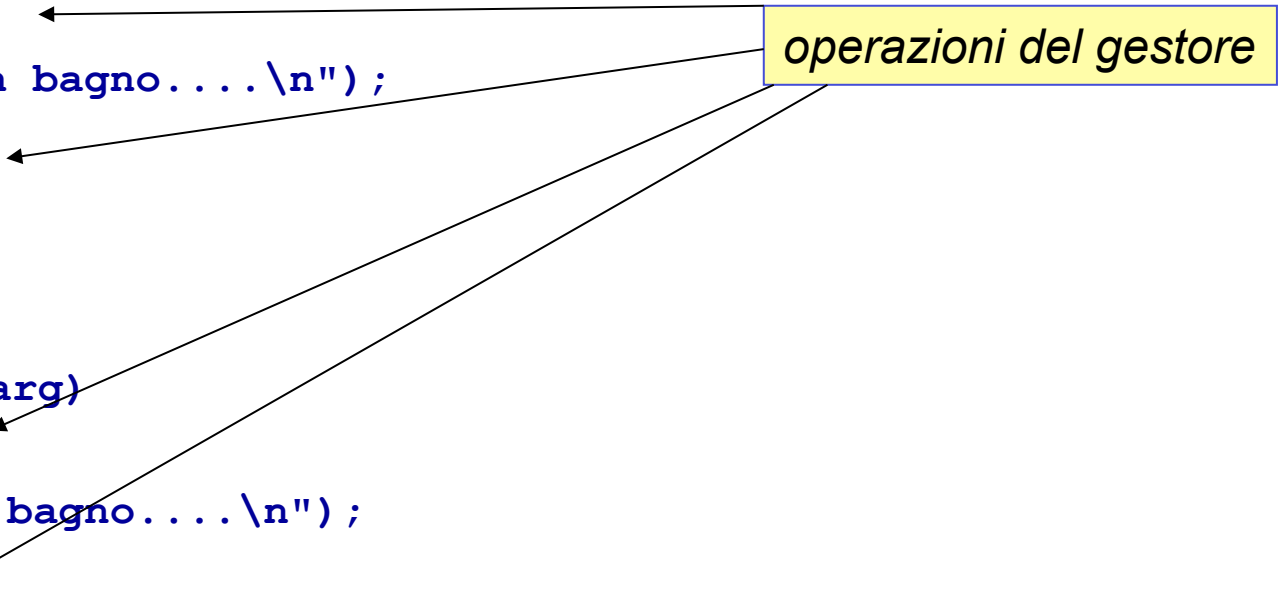
Spunti e suggerimenti (2)

Struttura dei thread:

```
gestore_toilet G;

void *donna(void * arg)
{   donna_entra(&G);
    printf("Donna in bagno....\n");
    donna_esce(&G);
}


void *uomo(void * arg)
{   uomo_entra(&G);
    printf("Uomo in bagno....\n");
    uomo_esce(&G);
}
```



Spunti e suggerimenti (3)

Quali condizioni di sincronizzazione?

- **donna (entra_donna):** il thread donna deve essere sospeso se:
  - ci sono uomini nel bagno;
  - il bagno e` pieno.
- **uomo (entra_uomo):** il thread uomo deve essere sospeso se:
  - ci sono donne nel bagno;
  - il bagno e` pieno;
  - ci sono donne in attesa.

Quali strumenti di sincronizzazione?

Necessita` di realizzare particolari politiche di allocazione:

- **semD**: semaforo ***privato*** per la sospensione dei thread donna
- **semU**: semaforo ***privato*** per la sospensione dei thread uomo

inoltre:

- **mutex**: un mutex per realizzare l'accesso al gestore in modo mutuamente esclusivo.

SOLUZIONE

Risorsa

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define N 3
#define MAX_T 60

typedef struct{
      pthread_mutex_t mutex; /* mutua esclusione*/
      int donne_in; /* numero di donne nella toilette*/
      int uomini_in; /* numero di uomini nella toilette*/
      sem_t semD; /* sospensione donne- sem. priv*/
      sem_t semU; /*sospensione uomini-sem. priv */
      int sosp_D; /* donne in attesa*/
      int sosp_U; /* uomini in attesa*/
}gestore_toilet;

gestore_toilet G;
```

Struttura Thread

```
void *thread_donna(void * arg) /*codice donna*/
{     donna_entra(&G);
      printf("Donna in bagno....\n");
      sleep(1); /*permanenza...*/
      donna_esce(&G);
      pthread_exit(0);
}

void *thread_uomo(void * arg) /*codice donna*/
{     uomo_entra(&G);
      printf("Uomo in bagno....\n");
      sleep(1); /*permanenza...*/
      uomo_esce(&G);
      pthread_exit(0);
}
```

Implementazione delle operazioni del *gestore*: soluzione basata sullo schema 1 (p. 106, modello a memoria comune)

Accesso donna

```
void donna_entra(gestore_toilet *g)
{     pthread_mutex_lock(&g->mutex);
      if( (g->donne_in+g->uomini_in<N)&&
           (g->uomini_in==0))
      {     g-> donne_in++;
            sem_post(&g->semD); /*v sul sem delle donne*/
      }
      else
            g->sosp_D++;
      pthread_mutex_unlock(&g->mutex);
      sem_wait(&g->semD);
}
```

Uscita Donna

```
void donna_esce(gestore_toilet *g)
{       int k;
        pthread_mutex_lock(&g->mutex);
        g->donne_in--;
        if (g->sosp_D)
        {       g->donne_in++;
                g->sosp_D--;
                sem_post(&g->semD); }
        else if (g->donne_in==0 && g->sosp_U)
        {       k=g->sosp_U;
               while(k>0 && g->uomini_in<=N )
                {       g->uomini_in++;
                        g->sosp_U--;
                        sem_post(&g->semU);
                        k--;
                }
        }
        pthread_mutex_unlock(&g->mutex);
}
```

Accesso Uomo

```
void uomo_entra(gestore_toilet *g)
{     pthread_mutex_lock(&g->mutex);
      if( (g->donne_in + g->uomini_in<N)&&
            (g->donne_in==0)&&
            (g->sosp_D==0))
      {     g-> uomini_in++;
            sem_post(&g->semU);
      }
      else
            g->sosp_U++;
      pthread_mutex_unlock(&g->mutex);
      sem_wait(&g->semU);
}
```

Uscita Uomo

```
void uomo_esce(gestore_toilet *g)
{       int k;
        pthread_mutex_lock(&g->mutex);
        g->uomini_in--;
        if (g->sosp_D && g->uomini_in==0) //risveglio donne
        {       k=g->sosp_D;
                while(k>0 && g->donne_in<=N )
                {       g->donne_in++;
                        g->sosp_D--;
                        k--;
                        sem_post(&g->semD); }}
        else if (g->sosp_D==0 && g->sosp_U>0)
        {       g->uomini_in++;
                g->sosp_U--;
                sem_post(&g->semU);
        }
        pthread_mutex_unlock(&g->mutex);}
```

Struttura main: avvio applicazione

```
main ()
{     pthread_t D[MAX_T], U[MAX_T];
      int i, nd, nu;
      /* inizializzazione G: */
      pthread_mutex_init(&G.mutex, NULL);
      sem_init(&G.semD,0,0);
      sem_init(&G.semU,0,0);
      G.sosp_D=0;
      G.sosp_U=0;
      G.donne_in=0; /* numero di donne in bagno*/
      G.uomini_in=0;/* numero di uomini in bagno */
      printf("Quante donne? "); scanf("%d", &nd);
      printf("Quanti uomini? "); scanf("%d", &nu);
      /* continua..*/
```

```
/* Creazione thread: */
for (i=0; i<nu; i++)
      pthread_create (&U[i], NULL, thread_uomo, NULL);
for (i=0; i<nd; i++)
      pthread_create (&D[i], NULL, thread_donna, NULL);
for (i=0; i<nd; i++)
      pthread_join (D[i], NULL);
for (i=0; i<nu; i++)
      pthread_join (U[i], NULL);
} /* fine main */
```

Implementazione delle operazioni del gestore:

(attesa circolare, v. Monitor)

Accesso donna

```
void donna_entra(gestore_toilet *g)
{     pthread_mutex_lock(&g->mutex);
      while ( (g->donne_in+g->uomini_in==N)||
            (g->uomini_in>0))
      {     g->sosp_D++;
            pthread_mutex_unlock(&g->mutex);
            sem_wait(&g->semD); /*sospensione donna*/
            pthread_mutex_lock(&g->mutex);
            g->sosp_D--;
      }
      g-> donne_in++;
      pthread_mutex_unlock(&g->mutex);
}
```

Accesso Uomo

```
void uomo_entra(gestore_toilet *g)
{     pthread_mutex_lock(&g->mutex);
      while ( (g->donne_in+g->uomini_in==N)||
            (g->donne_in>0)||
            (g->sosp_D)  )
      {     g->sosp_U++;
            pthread_mutex_unlock(&g->mutex);
            sem_wait(&g->semU); /*sospensione uomo*/
            pthread_mutex_lock(&g->mutex);
            g->sosp_U--;
      }
      g-> uomini_in++;
      pthread_mutex_unlock(&g->mutex);
}
```

Uscita Donna

```
void donna_esce(gestore_toilet *g)
{       int k;
        pthread_mutex_lock(&g->mutex);
        g->donne_in--;
        if (g->sosp_D)
                sem_post(&g->semD);
        else if (g->donne_in==0 && g->sosp_U)
        {       k=g->sosp_U;
                while(k>0)
                {       sem_post(&g->semU);
                        k--;
                }
        }
        pthread_mutex_unlock(&g->mutex);
}
```

Uscita Uomo

```
void uomo_esce(gestore_toilet *g)
{       int k;
        pthread_mutex_lock(&g->mutex);
        g->uomini_in--;
        if ((g->sosp_D)&& (g->uomini_in==0))
        {       k=g->sosp_D;
                while(k>0)
                {       sem_post(&g->semD);
                        k--;
                }
        }
        else if (g->sosp_D==0 && g->sosp_U)
                sem_post(&g->semU); /* risveglio un uomo */
        pthread_mutex_unlock(&g->mutex);
}
```