Esercizi thread

panini

January 2022

Contents

1 INCROCIO A 4 CORSIE FIFO

Presentazione Ci sono 4 strade a due corsie e doppio senso di marcia numerate da 0 a 3 in senso antiorario che confluiscono in un incrocio. Le auto girano sempre a destra e proseguono. Ogni auto impiega 2 secondi a disimpiegare l'incrocio Al massimo 2 automobili per volta possono impiegare l'incrocio Regole di precedenza Per evitare incidenti esiste una regola di precedenza: l'auto che proviene dalla i-esima strada ha la precedenza sulle strade di indice maggiore.

Code FIFO Ciascuna strada può mantenere più automobili in attesa per volta, affinché le cose siano eque occorre che tali code siano di tipo FIFO: la prima automobile che arriva sarà anche la prima ad attraversare all'incrocio.

```
#define STRADE 4
#define MAX_AUTO_IN_INCROCIO 2
#define AUTO_OGNI_15_SEC 6

typedef struct {
    int indice;
    int strada;
} InfoAuto;

/* auto in attesa di entrare nell'incrocio per ogni strada */
int autoInAttesa[STRADE];

/* distributore di biglietti per ciascuna strada */
int TurnoGlobale[STRADE];

/* turno corrente */
int Turno[STRADE];

int autoInIncrocio = 0;

pthread_mutex_t mutexBiglietto;
pthread_mutex_t mutexAttraversamento;
pthread_cond_t condAttraversamento;

int precedenza( int indicestrada ) {
    int i;
    for( i=indicestrada-1; i>=0; i-- ) {
        if( autoInAttesa[i] > 0 ) {
            return 0;
        }
    }
    return 1;
}

void *automobile( void *arg ) {

    char Alabel[128];
    int mioTurno;
    int mioIndice = (( InfoAuto *)arg)->indice;
    int miaStrada = (( InfoAuto *)arg)->strada;
    free( arg );
    sprintf( Alabel, "Auto %d proveniente dalla corsia %d", mioIndice, miaStrada );

    DBGpthread_mutex_lock( &mutexBiglietto, Alabel );
    mioTurno = TurnoGlobale[miaStrada]++;
    printf( "%s: prende biglietto %d\n", Alabel, mioTurno );
    DBGpthread_mutex_unlock( &mutexBiglietto, Alabel );

    DBGpthread_mutex_lock( &mutexAttraversamento, Alabel );
    while( autoInIncrocio >= MAX_AUTO_IN_INCROCIO
        || !precedenza( miaStrada )
        || mioTurno != Turno[miaStrada] ) {
        DBGpthread_cond_wait( &condAttraversamento, &mutexAttraversamento, Alabel );
```

```
    }
    printf( "%s: ( con biglietto %d ) può attraversare\n", Alabel, mioTurno );
    /* aumento le auto nell'incrocio */
    autoInIncrocio++;
    /* rimuovo un auto dalla lista di quelle in attesa */
    autoInAttesa[miaStrada]--;
    /* Aumento il ticket della mia strada */
    Turno[miaStrada]++;

    sleep( 2 );

    printf( "%s: ( con biglietto %d ) ha attraversato\n", Alabel, mioTurno );
    autoInIncrocio--;
    DBGpthread_cond_broadcast( &condAttraversamento, Alabel );
    DBGpthread_mutex_unlock( &mutexAttraversamento, Alabel );

    pthread_exit( NULL );
}

int main( void ) {

    pthread_t th;
    int rc, i, indiceAuto = 0;
    InfoAuto *info;
    srand( time( NULL ) );

    DBGpthread_mutex_init( &mutexBiglietto, NULL, "main" );
    DBGpthread_mutex_init( &mutexAttraversamento, NULL, "main" );
    DBGpthread_cond_init( &condAttraversamento, NULL, "main" );

    for( i=0; i<STRADE; i++ ) {
        autoInAttesa[i] = 0;
        TurnoGlobale[i] = 0;
        Turno[i] = 0;
    }

    while( 1 ) {
        for( i=0; i<AUTO_OGNI_15_SEC; i++ ) {
            info = ( InfoAuto *)malloc( sizeof( InfoAuto ));
            if( info == NULL ) {
                PrintErrnoAndExit( "Creazione intero" );
            }
            info->indice = indiceAuto++;
            info->strada = rand() % 4;
            rc = pthread_create( &th, NULL, automobile, ( void *)info );
            if( rc ) {
                PrintERROR_andExit( rc, "Creazione auto" );
            }
        }
        sleep( 15 );
        printf( "Main: altre 15 auto arrivate dioca\n" );
    }
    return 0;
}
```

2 ALIENI

Un gruppo di 5 alieni collaborativi ma claustrofobici ha dei problemi di spazio sul pianeta Zog, così occupa un piccolissimo monolocale. La casa può contenere al massimo 2 alieni e all'inizio è vuota. Per mantenere l'occupazione è indispensabile che in ogni istante ci sia almeno un alieno in casa. Gli alieni aspettano di entrare in casa in un gruppetto fuori dalla casa. A causa della claustrofobia, gli alieni cominciano a morire appena entrati in casa, così cercano di uscire appena possibile dalla casa, nell'ordine con cui sono entrati, per morire all'aperto. Subito dopo essere uscito di casa, ciascun alieno controlla il numero di alieni in attesa fuori di casa. Se l'alieno morente si accorge che ci sono meno di 3 alieni in attesa fuori casa, l'alieno crea altri 2 (thread) alieni e poi muore.

```
#define ALIENI_INIZIALI 5
#define ALIENI_MIN_FUORI 3
#define ALIENI_MAX_IN_CASA 2
#define ALIENI_MIN_IN_CASA 1

/* mutex per coordinare l'ingresso in casa */
pthread_mutex_t mutexIngressoCasa;
/* mutex per decrementare gli alieni fuori casa quando entrano e concorrentemente crearne 2 nuovi all'occorrenza */
pthread_mutex_t mutexControlloAlieniFuori;


/* condition per bloccare l'ingresso se ci sono 2 alieni già dentro casa */
pthread_cond_t condAttesaIngresso;
/* condition per bloccare l'uscita se sono l'unico alieno o non è il mio turno */
pthread_cond_t condAttesaUscita;

intptr_t nAlieniCreati = 0;
int distributoreTicket = 0;
int turnoCorrente = 0;
int nAlieniFuori = ALIENI_INIZIALI;
int nAlieniInCasa = 0;



void *alieno( void *arg ) {

    char Alabel[128];
    int mioTurno, i, rc;
    pthread_t th;
    sprintf( Alabel, "Alieno %" PRIiPTR "", ( intptr_t )arg );

    DBGpthread_mutex_lock( &mutexIngressoCasa, Alabel );
    while( nAlieniInCasa >= ALIENI_MAX_IN_CASA ) {
        printf( "%s: Casa PIENA (%d), non posso entrare\n", Alabel, nAlieniInCasa );
        DBGpthread_cond_wait( &condAttesaIngresso, &mutexIngressoCasa, Alabel );
    }

    nAlieniInCasa++;
    DBGpthread_mutex_lock( &mutexControlloAlieniFuori, Alabel );
    nAlieniFuori--;
    DBGpthread_mutex_unlock( &mutexControlloAlieniFuori, Alabel );

    /* appena entro prendo il ticket per segnarmi in che posizione sono entrato */
    mioTurno = distributoreTicket++;
    printf( "%s: Casa NON PIENA (%d), entra e prende ticket %d\n", Alabel, nAlieniInCasa, mioTurno );

    /* entrato in casa, inizia a morire */
    printf( "%s: CODDIO sto a mori\' devo uscire\n", Alabel );

    if( nAlieniInCasa > ALIENI_MIN_IN_CASA ) {
        DBGpthread_cond_broadcast( &condAttesaIngresso, Alabel );
    }
```

```
    while( nAlieniInCasa == ALIENI_MIN_IN_CASA || mioTurno != turnoCorrente ) {
        printf( "%s: non posso uscire (alieni in casa = %d) (mio ticket = %d, ticket corrente = %d) \n", Alabel, nAlieniInCasa, m
        DBGpthread_cond_wait( &condAttesaUscita, &mutexIngressoCasa, Alabel );
    }
    printf( "%s: posso uscire!!\n", Alabel );
    nAlieniInCasa--;
    turnoCorrente++;
    DBGpthread_cond_broadcast( &condAttesaIngresso, Alabel );
    DBGpthread_mutex_unlock( &mutexIngressoCasa, Alabel );

    DBGpthread_mutex_lock( &mutexControlloAlieniFuori, Alabel );
    if( nAlieniFuori < ALIENI_MIN_FUORI ) {
        printf( "%s: Alieni rimasti ad aspettare = %d, devo ricrearli\n", Alabel, nAlieniFuori );
        for( i=0; i<2; i++ ) {
            rc = pthread_create( &th, NULL, alieno, ( void *)nAlieniCreati++ );
            if( rc ) {
                PrintERROR_andExit( rc, "Creazione alieno" );
            }
        }
        nAlieniFuori+=2;
    }
    DBGpthread_mutex_unlock( &mutexControlloAlieniFuori, Alabel );
    pthread_exit( NULL );
}

int main( void ) {

    int rc;
    pthread_t th;
    intptr_t i;

    DBGpthread_mutex_init( &mutexIngressoCasa, NULL, "main" );
    DBGpthread_mutex_init( &mutexControlloAlieniFuori, NULL, "main" );
    DBGpthread_cond_init( &condAttesaIngresso, NULL, "main" );
    DBGpthread_cond_init( &condAttesaUscita, NULL, "main" );

    for( i=0; i<ALIENI_INIZIALI; i++ ) {
        rc = pthread_create( &th, NULL, alieno, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione alieno" );
        }
    }

    pthread_exit( NULL );
    return 0;
}
```

3 ELEFANTI E PECARI

Nella savana africana c'e' una pozza d'acqua fresca alimentata da un acquedotto. La pozza può contenere al massimo 500 litri d'acqua. C'e' un branco di N=5 elefanti che bevono ciascuno ogni volta 100 litri impiegando ciascuno 1/10 di secondo. Dopo avere bevuto ogni elefante fa un giretto e torna a bere dopo 1 secondo. C'e' un branco di M=10 pekari che bevono ciascuno ogni volta 1 litro impiegando ciascuno 1/100 di secondo. Dopo avere bevuto ogni pekari fa un giretto e torna a bere dopo 2.2 secondi. Potenzialmente alla pozza c'e' posto sufficiente affinché tutti gli animali possano bere assieme, però ci sono delle complicazioni. Più pekari possono bere contemporaneamente ad altri pekari. Più elefanti possono bere contemporaneamente ad altri elefanti. Se uno o più elefanti sta bevendo, ed un pekari inizia a bere, gli elefanti completano la bevuta. Se uno o più pekari stanno bevendo, nessun elefante si avvicina perché i pekari puzzano maledettamente, quindi gli elefanti aspettano. Un animale può cominciare a bere solo se nella pozza c'è acqua sufficiente per - far completare la bevuta di tutti gli animali che stanno già bevendo, - ed anche a far completare la bevuta dell'animale che vuole cominciare a bere. Ogni secondo, ed impiegando un tempo infinitesimo, l'acquedotto aggiunge alla pozza 70 litri. Se la pozza è già abbastanza piena, l'acqua in eccesso fuoriesce dalla pozza e viene persa

```c
#define CAPIENZA_POZZA 500
#define ELEFANTI 5
#define L_BEVUTI_DA_ELEFANTI 100
#define PEKARI 10
#define L_BEVUTI_DA_PEKARI 1
#define L_AGGIUNTI_DA_ACQUEDOTTO 70

int nElefantiInBevuta = 0;
int nPekariInBevuta = 0;
int litriPozza = CAPIENZA_POZZA;

pthread_mutex_t mutex;
pthread_cond_t condElefanti;
pthread_cond_t condPekari;


int calcoloAcquaNecessaria( int pekari, int elefanti ) {
    return ( pekari * L_BEVUTI_DA_PEKARI ) + ( elefanti * L_BEVUTI_DA_ELEFANTI );
}

void *pekari( void *arg ) {

    char Plabel[128];
    sprintf( Plabel, "Pekari %" PRIiPTR "", ( intptr_t )arg );

    while( 1 ) {
        DBGpthread_mutex_lock( &mutex, Plabel );
        printf( "%s: vorrei bere\n", Plabel );
        while( litriPozza < calcoloAcquaNecessaria( nPekariInBevuta+1, nElefantiInBevuta ) ){
            printf( "%s: Pozza non capiente abbastanza per fare bere anche me (litri rimasti %d, necessari %d)\n", Plabel, litriP
            DBGpthread_cond_wait( &condPekari, &mutex, Plabel );
        }
        nPekariInBevuta++;
        printf( "%s: Posso bere (elfeanti in bevuta %d, pekari in bevuta %d)\n", Plabel, nElefantiInBevuta, nPekariInBevuta );
        DBGpthread_mutex_unlock( &mutex, Plabel );
        DBGnanosleep( 10000000, Plabel );
        DBGpthread_mutex_lock( &mutex, Plabel );
        litriPozza -= L_BEVUTI_DA_PEKARI;
        nPekariInBevuta--;
        printf( "%s: ho finito di bere\n", Plabel );
        if( nPekariInBevuta == 0 ) {
            printf( "%s: ero l'ultimo pekaro in bevuta, sveglio gli elefanti\n", Plabel );
```

```
            DBGpthread_cond_broadcast( &condElefanti, Plabel );
        }
        DBGpthread_mutex_unlock( &mutex, Plabel );
       DBGnanosleep( 2200000000, Plabel );
    }

    pthread_exit( NULL );
}

void *elefante( void *arg ) {

    char Elabel[128];
    sprintf( Elabel, "Elefante %" PRIiPTR "", ( intptr_t )arg );

    while( 1 ) {
        DBGpthread_mutex_lock( &mutex, Elabel );
        printf( "%s: vorrei bere\n", Elabel );
        while( nPekariInBevuta > 0 || litriPozza < calcoloAcquaNecessaria( nPekariInBevuta, nElefantiInBevuta+1 ) ) {
            printf( "%s: Pozza non capiente abbastanza per fare bere anche me (litri rimasti %d, necessari %d) o pekari in bevuta
            DBGpthread_cond_wait( &condElefanti, &mutex, Elabel );
        }
        nElefantiInBevuta++;
        printf( "%s: Posso bere (elfeanti in bevuta %d, pekari in bevuta %d)\n", Elabel, nElefantiInBevuta, nPekariInBevuta );
        DBGpthread_mutex_unlock( &mutex, Elabel );
        DBGnanosleep( 100000000, Elabel );
        DBGpthread_mutex_lock( &mutex, Elabel );
        litriPozza -= L_BEVUTI_DA_ELEFANTI;
        nElefantiInBevuta--;
        printf( "%s: ho finito di bere, sveglio i pekari... magari c'e' ancora acqua\n", Elabel );
        DBGpthread_cond_broadcast( &condPekari, Elabel );
        DBGpthread_mutex_unlock( &mutex, Elabel );
        DBGnanosleep( 1000000000, Elabel );
    }

    pthread_exit( NULL );
}

void *acquedotto( void *arg ) {

    char Alabel[128];
    sprintf( Alabel, "Acquedotto" );

    while( 1 ) {
        DBGpthread_mutex_lock( &mutex, Alabel );
        printf( "%s: Pozza rifornita!\n", Alabel );
        litriPozza += (litriPozza + L_AGGIUNTI_DA_ACQUEDOTTO > CAPIENZA_POZZA ? CAPIENZA_POZZA : L_AGGIUNTI_DA_ACQUEDOTTO);
        DBGpthread_cond_broadcast( &condPekari, Alabel );
        DBGpthread_cond_broadcast( &condElefanti, Alabel );
        DBGpthread_mutex_unlock( &mutex, Alabel );
        DBGnanosleep( 1000000000, Alabel );
    }

    pthread_exit( NULL );
}

int main( void ) {
    int rc;
    intptr_t i;
    pthread_t th;

    DBGpthread_mutex_init( &mutex, NULL, "main" );
    DBGpthread_cond_init( &condElefanti, NULL, "main" );
    DBGpthread_cond_init( &condPekari, NULL, "main" );

    /* Creazione pekari */
    for( i=0; i<PEKARI; i++ ) {
        rc = pthread_create( &th, NULL, pekari, ( void *)i );
        if( rc ) {
```

```
            PrintERROR_andExit( rc, "Creazione pekari" );
        }
    }

    /* Creazione elefanti */
    for( i=0; i<ELEFANTI; i++ ) {
        rc = pthread_create( &th, NULL, elefante, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione elefante" );
        }
    }

    /* creazione acquedotto */
    rc = pthread_create( &th, NULL, acquedotto, NULL );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione acquedotto" );
        }

    pthread_exit( NULL );
    return 0;

}
```

4 CONIGLI

Un programma crea inizialmente 5 thread di tipo coniglio. Ciascun thread coniglio si accoppia con un altro thread coniglio. Dopo l'accoppiamento ciascuno dei thread coniglio crea un altro thread coniglio e poi termina. L'accoppiamento avviene secondo queste regole:

1. Per l'accoppiamento viene usata una sola ed unica tana, che all'inizio è vuota.
2. Nella tana possono entrare al massimo due conigli.
3. Per l'accoppiamento occorrono due conigli insieme nella tana.
4. Quando due conigli sono nella tana l'accoppiamento è istantaneo.
5. Prima che due altri conigli possano entrare, entrambi i conigli che erano precedentemente nella tana devono essere usciti

```
#define CONIGLI_INIZIALI 5
#define CAPIENZA_MAX_TANA 2
/*
 * true -> 1
 * false -> 0
 */

int tanaPiena = 0;
int conigliNellaTana = 0;
intptr_t numeroConigliCreati = 0;
pthread_mutex_t mutex;
pthread_cond_t condConigliNellaTana;
pthread_cond_t condConigliFuoriTana;


void *coniglio( void *arg ) {

    int rc;
    pthread_t th;
    char Clabel[128];
    sprintf( Clabel, "Coniglio %" PRIiPTR "", ( intptr_t )arg );

   /* while( 1 ) { */
        DBGpthread_mutex_lock( &mutex, Clabel );
        while( tanaPiena ) {
            printf("%s: tana piena, deve aspettare\n", Clabel );
            DBGpthread_cond_wait( &condConigliFuoriTana, &mutex, Clabel );
        }
        conigliNellaTana++;
        printf("%s: entrato nella tana, (numero conigli nella tana = %d)\n", Clabel, conigliNellaTana );
        if( conigliNellaTana == CAPIENZA_MAX_TANA ) {
            tanaPiena = 1;
        }
        while( !tanaPiena ) {
            printf("%s: deve aspettare il secondo coniglio\n", Clabel );
            DBGpthread_cond_wait( &condConigliNellaTana, &mutex, Clabel );
        }
        DBGpthread_cond_signal( &condConigliNellaTana, Clabel );

        rc = pthread_create( &th, NULL, coniglio, ( void *)numeroConigliCreati++ );
        if( rc != 0 ) {
            PrintERROR_andExit( rc, "Creazione coniglio" );
        }
        printf("%s: coniglio nato\n", Clabel );

        conigliNellaTana--;
        printf("%s: se ne va\n", Clabel );
        if( conigliNellaTana == 0 ) {
            printf("%s: tana svuotata\n", Clabel );
```

```
            tanaPiena = 0;
            DBGpthread_cond_broadcast( &condConigliFuoriTana, Clabel );
        }
        DBGpthread_mutex_unlock( &mutex, Clabel );
    /* } */

    pthread_exit( NULL );
}

int main( void ) {

    int rc;
    pthread_t th;

    DBGpthread_mutex_init( &mutex, NULL, "main" );
    DBGpthread_cond_init( &condConigliFuoriTana, NULL, "main" );
    DBGpthread_cond_init( &condConigliNellaTana, NULL, "main" );

    for( numeroConigliCreati = 0; numeroConigliCreati < CONIGLI_INIZIALI; numeroConigliCreati++ ) {
        rc = pthread_create( &th, NULL, coniglio, (void *)numeroConigliCreati );
        if( rc != 0 ) {
            PrintERROR_andExit( rc, "Creazione figlio" );
        }
    }

    pthread_exit( NULL );
    return 0;
}
```

5 CANI E VIC

Uno sconosciuto ciclista, vic, pedala sulle strade sterrate che si inerpicano verso Ciola Araldi, nelle colline tra Cesena e Sogliano. Nelle peggiori salite, ogni cane che incontra cerca di morderlo alle caviglie, e vic si difende ringhiando e scalciando mentre continua a pedalare. I cani sono talmente tanti che non riescono ad avvicinarsi tutti alle caviglie di vic, così si organizzano formando una fila che rincorre vic. Al massimo due cani possono mordere contemporaneamente la stessa caviglia. Dopo un primo tentativo di morso, il cane si becca un calcio nei denti e torna in fondo alla fila, aspettando il proprio turno. Quando una caviglia è occupata da meno di due cani, il primo cane della fila va a mordere quella caviglia

```
#define CANI 5
#define MAX_CANI 4

pthread_mutex_t mutexTicket;
pthread_mutex_t mutexTurno;

pthread_cond_t condTurno;
pthread_cond_t condCalcioDiVic;
pthread_cond_t condVicPedala;

int caniCheMordono = 0;
int distributoreTicket = 0;
int turnoCorrente = 0;

void *cane( void *arg ) {
    char Clabel[128];
    int mioTurno;
    intptr_t indice = ( intptr_t )arg;

    while( 1 ) {
        DBGpthread_mutex_lock( &mutexTicket, "cane prende ticket\n" );
        mioTurno = distributoreTicket++;
        DBGpthread_mutex_unlock( &mutexTicket, "cane si mette in coda\n" );
        sprintf( Clabel, "Cane %" PRIiPTR " con turno %d", indice, mioTurno );
        DBGpthread_mutex_lock( &mutexTurno, Clabel );
        while( caniCheMordono == MAX_CANI || mioTurno != turnoCorrente ) {
            printf( "%s: in ATTESA, caviglie piene o non il suo turno (turno corrente = %d, caviglie morse = %d)\n", Clabel, turn
            DBGpthread_cond_wait( &condTurno, &mutexTurno, Clabel );
        }
        turnoCorrente++;
        printf( "%s: INIZIA a mordere, ATTENDE calcio di Vic\n", Clabel );
        caniCheMordono++;
        if( caniCheMordono == 1 ) {
            printf( "%s: e\' il PRIMO: sveglia Vic\n", Clabel );
            DBGpthread_cond_signal( &condVicPedala, Clabel );
        }
        DBGpthread_cond_wait( &condCalcioDiVic, &mutexTurno, Clabel );
        printf( "%s: AHIAAA, calcio ricevuto\n", Clabel );
        DBGpthread_cond_broadcast( &condTurno, Clabel );
        DBGpthread_mutex_unlock( &mutexTurno, Clabel );
    }
    pthread_exit( NULL );
}

void *vic( void *arg ) {

    char Vlabel[128];
    sprintf( Vlabel, "Vic" );
    while( 1 ) {
        DBGpthread_mutex_lock( &mutexTurno, Vlabel );
        while( caniCheMordono <= 0 ) {
            printf( "%s: infarto in arrivo (pedala)\n", Vlabel );
            DBGpthread_cond_wait( &condVicPedala, &mutexTurno, Vlabel );
        }
        printf( "%s: DIO $@#! che male!!! vai via bastardo!\n", Vlabel );
        caniCheMordono--;
        DBGpthread_cond_signal( &condCalcioDiVic, Vlabel );
```

```
        DBGpthread_mutex_unlock( &mutexTurno, Vlabel );
        sleep( 1 );
    }
    pthread_exit( NULL );
}

int main( void ) {

    int rc;
    intptr_t i;
    pthread_t th;

    DBGpthread_mutex_init( &mutexTicket, NULL, "main" );
    DBGpthread_mutex_init( &mutexTurno, NULL, "main" );
    DBGpthread_cond_init( &condCalcioDiVic, NULL, "main" );
    DBGpthread_cond_init( &condVicPedala, NULL, "main" );
    DBGpthread_cond_init( &condTurno, NULL, "main" );

    rc = pthread_create( &th, NULL, vic, NULL );
    if( rc ) {
        PrintERROR_andExit( rc, "Creazione vic" );
    }

    for( i=0; i<CANI; i++ ){
        rc = pthread_create( &th, NULL, cane, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione vic" );
        }
    }
    pthread_exit( NULL );
    return 0;
}
```

6 SETTA

Una setta religiosa, con unica sede a Gattolino di Cesena, venera un Dio bizzarro che mira a sincronizzare in maniera egualitaria i suoi adepti. Nel luogo di culto ci sono N=16 sacerdoti, di cui M=10 sono novizi e L=6 sono anziani. I sacerdoti novizi devono svolgere ripetutamente le seguenti azioni: - prendere un bicchiere di vino (santo) da un primo distributore inesauribile, - inginocchiarsi di fronte alla statua del Dio e versare il vino ai suoi piedi. I sacerdoti anziani, invece, devono svolgere ripetutamente le seguenti azioni: - prendere un bicchiere di vino (buono) da un secondo distributore inesauribile, - allontanarsi dal distributore e bere il bicchiere di vino buono, - prendere un altro bicchiere di vino (santo) dal primo distributore inesauribile, - inginocchiarsi di fronte alla statua del Dio e versare il vino ai suoi piedi salmodiando frasi sconnesse . Vanno rispettati alcuni dettami divini: - un solo sacerdote per volta può attingere al distributore di vino santo. - un solo sacerdote per volta può attingere al distributore di vino buono. - un solo sacerdote per volta può inginocchiarsi di fronte alla statua. - ciascun sacerdote può tornare ad inginocchiarsi nuovamente per la n+1 esima volta solo dopo che anche tutti gli altri sacerdoti si sono inginocchiati per n volte. Inoltre, tutti conoscono il numero totale di sacerdoti.

```c
#define SACERDOTI_TOT 16
#define SACERDOTI_NOVIZI 10
#define SACERDOTI_ANZIANI 6

int nInginocchiamentiTutti = 0; /* nr. di volte in cui tutti si sono inginocchiati */
int nPersoneInginocchiate = 0; /* persone attualmente inginocchiate */

pthread_mutex_t mutexDistributoreBuono;
pthread_mutex_t mutexDistributoreSanto;
pthread_mutex_t mutexInginocchiamento;
pthread_cond_t condInginocchio;

void *anziano( void *arg ) {
    char Alabel[128];
    int nInginocchiamentiPersonali = 0;
    sprintf( Alabel, "Sacerdote anziano %" PRIiPTR "", ( intptr_t )arg );
    while( 1 ) {

        DBGpthread_mutex_lock( &mutexDistributoreBuono, Alabel );
        printf( "%s: prende VINO BUONO e si sbronza\n", Alabel );
        DBGpthread_mutex_unlock( &mutexDistributoreBuono, Alabel );

        DBGpthread_mutex_lock( &mutexDistributoreSanto, Alabel );
        printf( "%s: prende VINO SANTO\n", Alabel );
        DBGpthread_mutex_unlock( &mutexDistributoreSanto, Alabel );

        DBGpthread_mutex_lock( &mutexInginocchiamento, Alabel );
        while( nInginocchiamentiPersonali != nInginocchiamentiTutti ) {
            DBGpthread_cond_wait( &condInginocchio, &mutexInginocchiamento, Alabel );
        }
        sleep( 1 );
        printf( "%s: si inginocchia per la %d volta (totali = %d) e versa il VINO SANTO\n", Alabel, nInginocchiamentiPersonali, n
        nInginocchiamentiPersonali++;
        nPersoneInginocchiate++;
        if( nPersoneInginocchiate == SACERDOTI_TOT ) {
            printf( "%s: Sono l\'ultimo\n", Alabel );
            nInginocchiamentiTutti++;
            nPersoneInginocchiate = 0;
        }
        printf( "%s: ha finito di inginocchiarsi\n", Alabel );
        DBGpthread_cond_broadcast( &condInginocchio, Alabel );
        DBGpthread_mutex_unlock( &mutexInginocchiamento, Alabel );
    }
```

```
    pthread_exit( NULL );
}

void *novizio( void *arg ) {
    char Nlabel[128];
    int nInginocchiamentiPersonali = 0;
    sprintf( Nlabel, "Sacerdote novizio %" PRIiPTR "", ( intptr_t )arg );
    while( 1 ) {

        DBGpthread_mutex_lock( &mutexDistributoreSanto, Nlabel );
        printf( "%s: prende VINO SANTO\n", Nlabel );
        DBGpthread_mutex_unlock( &mutexDistributoreSanto, Nlabel );

        DBGpthread_mutex_lock( &mutexInginocchiamento, Nlabel );
        while( nInginocchiamentiPersonali != nInginocchiamentiTutti ) {
            DBGpthread_cond_wait( &condInginocchio, &mutexInginocchiamento, Nlabel );
        }
        sleep( 1 );
        printf( "%s: si inginocchia per la %d volta (totali = %d) e versa il VINO SANTO\n", Nlabel, nInginocchiamentiPersonali, n
        nInginocchiamentiPersonali++;
        nPersoneInginocchiate++;
        if( nPersoneInginocchiate == SACERDOTI_TOT ) {
             printf( "%s: Sono l\'ultimo\n", Nlabel );
            nInginocchiamentiTutti++;
            nPersoneInginocchiate = 0;
        }
        printf( "%s: ha finito di inginocchiarsi\n", Nlabel );
        DBGpthread_cond_broadcast( &condInginocchio, Nlabel );
        DBGpthread_mutex_unlock( &mutexInginocchiamento, Nlabel );

    }
    pthread_exit( NULL );
}

int main( void ) {
    int rc;
    pthread_t th;
    intptr_t i;

    DBGpthread_mutex_init( &mutexDistributoreBuono, NULL, "main" );
    DBGpthread_mutex_init( &mutexDistributoreSanto, NULL, "main" );
    DBGpthread_mutex_init( &mutexInginocchiamento, NULL, "main" );
    DBGpthread_cond_init( &condInginocchio, NULL, "main" );

    for( i=0; i<SACERDOTI_ANZIANI; i++ ) {
        rc = pthread_create( &th, NULL, anziano, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione anziano" );
        }
    }

    for( i=0; i<SACERDOTI_NOVIZI; i++ ) {
        rc = pthread_create( &th, NULL, novizio, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione anziano" );
        }
    }

    pthread_exit( NULL );
    return 0;
}
```

7 FORNAIO FIFO

Un fornaio ha deciso di installare una macchinetta che distribuisce dei biglietti con un numero crescente per dirimere le liti tra i clienti in coda per essere serviti. Ciascun cliente che entra accede al distributore e prende un biglietto, poi aspetta di essere chiamato. Ogni volta che il fornaio finisce di servire un cliente, guarda se c'è qualcuno in attesa. Se nel negozio non c'è nessuno, il fornaio aspetta. Appena entra un cliente, il fornaio aspetta che il cliente abbia preso il biglietto, poi il fornaio guarda il biglietto consegnatogli dall'ultimo cliente che ha servito e chiama il numero del biglietto successivo. Il cliente chiamato consegna al fornaio il proprio biglietto e viene servito. Se invece nel negozio c'è qualcuno in coda, il fornaio guarda il biglietto consegnatogli dal cliente appena servito e chiama il numero del biglietto successivo. Il cliente chiamato si avvicina al bancone, consegna al fornaio il proprio biglietto e viene servito. Ipotizziamo che:

1. ad apertura negozio il fornaio prende lui stesso il biglietto col numero 0.
2. il primo cliente prende il biglietto col numero 1.
3. nessun cliente va via dopo avere preso il biglietto; tutti restano in coda fino ad essere serviti.

```
#define CLIENTI 10

int distributoreTicket = 1; /* primo ticket (lo 0) preso dal fornaio */
int turnoCorrente = 0;
int personeInCoda = 0;

pthread_mutex_t mutexDistributore;
pthread_mutex_t mutexTurno;
pthread_cond_t condFornaio;
pthread_cond_t condTurno;
pthread_cond_t condClienteServito;


void *fornaio( void *arg ) {

    char Flabel[128];
    sprintf( Flabel, "Fornaio" );

    while( 1 ) {
        DBGpthread_mutex_lock( &mutexDistributore, Flabel );
        while( personeInCoda == 0 ){
            printf( "%s: non ci sono persone in coda, pisolo\n", Flabel );
            DBGpthread_cond_wait( &condFornaio, &mutexDistributore, Flabel );
        }
        DBGpthread_mutex_unlock( &mutexDistributore, Flabel );
        DBGpthread_mutex_lock( &mutexTurno, Flabel );
        /* entrato qui c'è sicuramente qualcuno in coda */
        turnoCorrente++;
        printf( "%s: sono stato svegliato! c'è qualcuno in coda, incremento il turno corrente che diventa %d!\n", Flabel, turnoCo
        DBGpthread_cond_broadcast( &condTurno, Flabel );
        /* serve il cliente */
        DBGpthread_cond_wait( &condClienteServito, &mutexTurno, Flabel );
        DBGpthread_mutex_unlock( &mutexTurno, Flabel );
    }
    pthread_exit( NULL );
}

void *cliente( void *arg ){

    char Clabel[128];
    int mioTicket;
```

```
    DBGpthread_mutex_lock( &mutexDistributore, Clabel );
    mioTicket = distributoreTicket++;
    sprintf( Clabel, "Cliente %" PRIiPTR " con ticket %d", ( intptr_t )arg, mioTicket );
    personeInCoda++;
    printf( "%s: si aggiunge alla coda che conta %d persone\n", Clabel, personeInCoda );
    if( personeInCoda == 1 ){
        printf( "%s: sveglia il fornaio\n", Clabel );
        DBGpthread_cond_signal( &condFornaio, Clabel );
    }
    DBGpthread_mutex_unlock( &mutexDistributore, Clabel );

    DBGpthread_mutex_lock( &mutexTurno, Clabel );
    while( mioTicket != turnoCorrente ){
        printf( "%s: non è il mio turno (turnoCorrente = %d)\n", Clabel, turnoCorrente );
        DBGpthread_cond_wait( &condTurno, &mutexTurno, Clabel );
    }
    DBGpthread_mutex_lock( &mutexDistributore, Clabel );
    personeInCoda--;
    DBGpthread_mutex_unlock( &mutexDistributore, Clabel );

    /* il fornaio mi ha svegliato ed è il mio turno, vengo servito */
    sleep( 1 );
    printf( "%s: è il mio turno, mi faccio servire dal fornaio\n", Clabel );
    DBGpthread_cond_signal( &condClienteServito, Clabel );
    DBGpthread_cond_broadcast( &condTurno, Clabel );
    DBGpthread_mutex_unlock( &mutexTurno, Clabel );
    pthread_exit( NULL );
}

int main( void ) {
    int rc;
    pthread_t th;
    intptr_t i;

    DBGpthread_mutex_init( &mutexDistributore, NULL, "main" );
    DBGpthread_mutex_init( &mutexTurno, NULL, "main" );
    DBGpthread_cond_init( &condFornaio, NULL, "main" );
    DBGpthread_cond_init( &condTurno, NULL, "main" );

    rc = pthread_create( &th, NULL, fornaio, NULL );
    if( rc ) {
        PrintERROR_andExit( rc, "Creazione fornaio" );
    }

    for( i=0; i<CLIENTI; i++ ){
        rc = pthread_create( &th, NULL, cliente, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione fornaio" );
        }
    }

    pthread_exit( NULL );
    return 0;
```

8 LEGIONARI

Nella caserma della legione straniera, alla mattina ci si mette in coda per radersi. N legionari attendono, in due diverse file, che vengano loro assegnati, in una fila un rasoio e nell'altra fila una scodella con la schiuma da barba. Il numero di rasoi e' M<N. Il numero delle scodelle e' K<M. Dopo essersi rasati, i legionari restituiscono i rasoi e le scodelle ai due sottoufficiali che gestiscono una fila ciascuno.

```
#define RASOIO 0
#define SCODELLA 1

#define RASOI 1
#define SCODELLE 2
#define LEGIONARI 10

int rasoiRimanenti = RASOI;
int scodelleRimanenti = SCODELLE;
pthread_mutex_t mutexRasoio;
pthread_mutex_t mutexScodella;
pthread_cond_t condRasoio;
pthread_cond_t condScodella;

pthread_mutex_t *mutex( intptr_t indice ) {
    return ( indice == RASOIO ) ? &mutexRasoio : &mutexScodella;
}

pthread_cond_t *filaControllata( intptr_t indice ) {
    return (indice == RASOIO ) ? &condRasoio : &condScodella;
}

int oggettoControllato( intptr_t indice ) {
    return ( indice == RASOIO ) ? rasoiRimanenti : scodelleRimanenti;
}

void *sottoufficiale( void *arg ) {
    char Slabel[128];
    sprintf( Slabel, "sottoufficiale addetto %s", ( ( intptr_t )arg == RASOIO ) ? "ai rasoii" : "alle scodelle" );

    while( 1 ) {
        DBGpthread_mutex_lock( mutex( ( intptr_t )arg ), Slabel );
        if( oggettoControllato( ( intptr_t )arg ) > 0 ) {
            /* printf( "%s: può darlo (rimanenti = %d)\n", Slabel, oggettoControllato( ( intptr_t )arg )); */
            DBGpthread_cond_broadcast( filaControllata( ( intptr_t )arg ), Slabel );
        } else {
            /* printf( "%s: NON può darlo (rimanenti = %d)\n", Slabel, oggettoControllato( ( intptr_t )arg )); */
        }
        DBGpthread_mutex_unlock( mutex( ( intptr_t )arg ), Slabel );
    }
    pthread_exit( NULL );
}

void *legionario( void *arg ) {

    char Llabel[128];
    sprintf( Llabel, "legionario %" PRIiPTR "", ( intptr_t )arg );

    while( 1 ) {

        DBGpthread_mutex_lock( &mutexRasoio, Llabel );
        printf( "%s: si mette in coda per il rasoio\n", Llabel );
        DBGpthread_cond_wait( &condRasoio, &mutexRasoio, Llabel );
        printf( "%s: ha preso il rasoio\n", Llabel );
        rasoiRimanenti--;
        DBGpthread_mutex_unlock( &mutexRasoio, Llabel );

        DBGpthread_mutex_lock( &mutexScodella, Llabel );
        printf( "%s: si mette in coda per la scodella\n", Llabel );
```

```
        DBGpthread_cond_wait( &condScodella, &mutexScodella, Llabel );
        printf( "%s: ha preso la scodella\n", Llabel );
        scodelleRimanenti--;
        DBGpthread_mutex_unlock( &mutexScodella, Llabel );

        /* si può radere porcanna la madonna */

        DBGpthread_mutex_lock( &mutexRasoio, Llabel );
        printf( "%s: ha finito di radersi, restituisce le cose\n", Llabel );
        rasoiRimanenti++;
        DBGpthread_cond_broadcast( &condRasoio, Llabel );
        DBGpthread_mutex_unlock( &mutexRasoio, Llabel );

        DBGpthread_mutex_lock( &mutexScodella, Llabel );
        scodelleRimanenti++;
        DBGpthread_cond_broadcast( &condScodella, Llabel );
        DBGpthread_mutex_unlock( &mutexScodella, Llabel );
    }
    pthread_exit( NULL );
}

int main( void ) {
    int rc;
    pthread_t th;
    intptr_t i;

    DBGpthread_mutex_init( &mutexRasoio, NULL, "Main" );
    DBGpthread_mutex_init( &mutexScodella, NULL, "Main" );
    DBGpthread_cond_init( &condRasoio, NULL, "Main" );
    DBGpthread_cond_init( &condScodella, NULL, "Main" );

    rc = pthread_create( &th, NULL, sottoufficiale, (void *)RASOIO );
    if( rc != 0 ) {
        PrintERROR_andExit( rc, "Creazione sottoufficiale" );
    }

    rc = pthread_create( &th, NULL, sottoufficiale, (void *)SCODELLA );
    if( rc != 0 ) {
        PrintERROR_andExit( rc, "Creazione sottoufficiale" );
    }

    for( i = 0; i < LEGIONARI; i++ ) {
        rc = pthread_create( &th, NULL, legionario, ( void *)i );
        if( rc != 0 ) {
            PrintERROR_andExit( rc, "Creazione sottoufficiale" );
        }
    }
    pthread_exit( NULL );
}
```

9 PIATTELLO

In un poligono di tiro 10 tiratori si allenano sparando a dei piattelli lanciati in volo. Ogni 8 secondi una macchina (il main) lancia in aria un piattello, il piattello vola per 3 secondi e infine cade a terra. Nel momento in cui il piattello viene lanciato, tutti i tiratori cercano di colpirlo. ogni tiratore impiega un tempo variabile tra 2 e 4 secondi per mirare e poi, SE IL PIATTELLO E' ANCORA IN VOLO, spara impiegando 0 secondi e SICURAMENTE colpisce il piattello. Se invece il piattello è già caduto a terra il tiratore grida di rabbia e non spara. Il piattello se colpito comunque continua il volo e può essere colpito da altri tiratori. I tiratori possono mirare e sparare anche tutti assieme. Ogni piattello è un diverso thread. Circa ogni 8 secondi la macchina (il main) crea un nuovo thread piattello. Ogni thread piattello inizia il volo e alla fine del volo cade a terra e termina. Prima di terminare ogni piattello stampa un messaggio e dice se è stato colpito o no. Ciascuno dei 10 tiratori è un thread che aspetta il lancio dei piattelli e cerca di colpirli, senza mai terminare.

```
#define NUMTIRATORI 10
#define DELAYTRADUEPIATTELLI8sec 8

/* dati da proteggere - AGGIUNGETE QUELLO CHE VI SERVE */
int piattelloInVolo = 0;
int piattelloColpito = 0;

/* variabili per la sincronizzazione - AGGIUNGETE QUELLO CHE VI SERVE */
pthread_mutex_t mutexPiattello;
pthread_cond_t condTiratore;

void attendi( int min, int max) {
        int secrandom=0;
        if( min > max ) return;
        else if ( min == max )
                secrandom = min;
        else
                secrandom = min + ( random()%(max-min+1) );
        do {
                /* printf("attendi %i\n", secrandom);fflush(stdout); */
                secrandom=sleep(secrandom);
                if( secrandom>0 )
                        { printf("sleep interrupted - continue\n"); fflush(stdout); }
        } while( secrandom>0 );
        return;
}

void *Tiratore (void *arg)
{
        char Plabel[128];
        intptr_t indice;

        indice=(intptr_t)arg;
        sprintf(Plabel,"Tiratore%" PRIiPTR "",indice);

        DBGpthread_mutex_lock( &mutexPiattello, Plabel );
        while ( 1 ) {
                /* il tiratore  attende l'inizio del volo del piattello */
                printf("tiratore %s attende piattello \n", Plabel);
                fflush(stdout);
                DBGpthread_cond_wait( &condTiratore, &mutexPiattello, Plabel );
                DBGpthread_mutex_unlock( &mutexPiattello, Plabel );
                printf("tiratore %s mira e .... \n", Plabel);
                fflush(stdout);


                /* il tiratore si prepara a sparare impiegando da 2 a 4 secondi */
                attendi( 2, 4 );
                DBGpthread_mutex_lock( &mutexPiattello, Plabel );
                if( piattelloInVolo ) {
                        piattelloColpito = 1;
                        printf("tiratore %s daije: colpito!!\n", Plabel);
```

```
                } else {
                        printf("tiratore %s non colpito $@&!\n", Plabel);
                }
                        fflush(stdout);
        }
        pthread_exit(NULL);
}

void *Piattello (void *arg)
{
        char Plabel[128];
        intptr_t indice;

        indice=(intptr_t)arg;
        sprintf(Plabel,"Piattello%" PRIiPTR "",indice);

        DBGpthread_mutex_lock( &mutexPiattello, Plabel );
        piattelloInVolo = 1;
        piattelloColpito = 0;
        DBGpthread_cond_broadcast( &condTiratore, Plabel );
        DBGpthread_mutex_unlock( &mutexPiattello, Plabel );

        printf("piattelo %s inizia volo\n", Plabel);
        fflush(stdout);

        /* il piattello vola per tre secondi */
        attendi( 3, 3 );

        DBGpthread_mutex_lock( &mutexPiattello, Plabel );
        piattelloInVolo = 0;
        printf("piattelo %s finisce volo e termina\n", Plabel);
               fflush(stdout);
        if( piattelloColpito ) {
                printf( "piattello %s è stato colpito!\n", Plabel );
        }
        DBGpthread_mutex_unlock( &mutexPiattello, Plabel );
        pthread_exit(NULL);
}

int main ( int argc, char* argv[] )
{
        pthread_t    th;
        int  rc;
        uintptr_t i=0;
        int seme;


        seme=time(NULL);
        srandom(seme);

        /* INIZIALIZZATE LE VOSTRE VARIABILI CONDIVISE e tutto quel che serve - fate voi */
        DBGpthread_mutex_init(&mutexPiattello, NULL, "main");
        DBGpthread_cond_init(&condTiratore, NULL, "main");

        /* CREAZIONE PTHREAD dei tiratori */
        for(i=0;i<NUMTIRATORI;i++) {
                rc=pthread_create(&th,NULL,Tiratore,(void*)i);
                if(rc) PrintERROR_andExit(rc,"pthread_create failed");
        }

        /* CREAZIONE NUOVO PIATTELLO OGNI 8 secondi */
        i=0;
        while(1) {
                        /* un nuovo piattello ogni 8 secondi */
                        attendi( DELAYTRADUEPIATTELLI8sec, DELAYTRADUEPIATTELLI8sec );
                        i++;

                        /* crea pthread piattello - completate voi */
```

```
                        rc = pthread_create( &th, NULL, Piattello, ( void *)i );
                        if( rc ) {
                                PrintERROR_andExit(rc, "Creazione piattello");
                        }
        }

        pthread_exit(NULL);
        return(0);
}
```

10 SYNC

Un main genera 5 pthread e ne attende la fine. Ciascun thread aspetta 1 secondo e poi aspetta che tutti i 5 thread siano dentro la funzione che li implementa e solo allora i thread possono terminare. Però, ogni thread ottiene il permesso di terminare nello stesso ordine con cui i thread si sono messi in attesa dell'arrivo degli altri thread.

```
#define THREADS 5

int threadDentroFunzione = 0;
int distributoreTicket = 0;
int turnoCorrente = 0;

pthread_mutex_t mutexDistributore;
pthread_mutex_t mutexTurno;
pthread_cond_t cond;

void *Thread( void *arg ){
    char Tlabel[128];
    int mioTicket;
    sprintf( Tlabel, "Thread %" PRIiPTR "", ( intptr_t )arg );

    DBGpthread_mutex_lock( &mutexDistributore, Tlabel );
    mioTicket = distributoreTicket++;
    printf( "%s: thread attende con ticket %d\n", Tlabel, mioTicket );
    DBGpthread_mutex_unlock( &mutexDistributore, Tlabel );

    sleep( 1 );

    DBGpthread_mutex_lock( &mutexTurno, Tlabel );
    threadDentroFunzione++;
    if( threadDentroFunzione == THREADS ) {
        DBGpthread_cond_broadcast( &cond, Tlabel );
    }
    while( mioTicket != turnoCorrente || threadDentroFunzione != THREADS ) {
        DBGpthread_cond_wait( &cond, &mutexTurno, Tlabel );
    }
    turnoCorrente++;
    printf( "%s: mi levo dalle pa**e\n", Tlabel );
    DBGpthread_cond_broadcast( &cond, Tlabel );
    DBGpthread_mutex_unlock( &mutexTurno, Tlabel );
    pthread_exit( NULL );
}

int main( void ) {
    int rc;
    pthread_t id[THREADS];
    intptr_t i;

    DBGpthread_mutex_init( &mutexDistributore, NULL, "main" );
    DBGpthread_mutex_init( &mutexTurno, NULL, "main" );
    DBGpthread_cond_init( &cond, NULL, "main" );

    for( i=0; i<THREADS; i++ ) {
        rc = pthread_create( &id[i], NULL, Thread, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione thread" );
        }
    }

    for( i=0; i<THREADS; i++ ){
        rc = pthread_join( id[i], NULL );
        if( rc ) {
            PrintERROR_andExit( rc, "Join thread" );
        }
    }

    pthread_exit( NULL );
    return 0;
```

```
}
```

11 VACCHE E BOVARO

In un allevamento di bovini, su una parete ci sono 3 mangiatoie una a fianco all'altra, numerate da 0 a 2. Le mangiatoie sono sempre piene di mangime. Solo una vacca per volta può mangiare in una mangiatoia. Circa ogni 20 secondi, all'esterno della stalla un bovaro pazzo incendia della paglia, la fa bruciare per 5 secondi e poi la spegne e così all'infinito. Ciascuna vacca impiega per mangiare un tempo casuale compreso tra 5 e 7 secondi, poi va in giro per 10 secondi e poi torna alla mangiatoia e aspetta che ci sia posto per mangiare ancora, e così via. Però, solo mentre sta mangiando, ad ogni secondo la vacca alza la testa, se vede che la paglia sta bruciando si spaventa, smette di mangiare e va in giro per 10 secondi, poi ritorna e cerca di trovare ancora una mangiatoia in cui ricominciare a mangiare. Ci sono 6 vacche ed 1 bovaro pazzo.

```
#define NUMMANGIATOIE 3
#define NUMVACCHE 6
#define SECGIRO 10
#define MINSECMANGIARE 5
#define MAXSECMANGIARE 7
#define SECINTERVALLOTRAINCENDI 15
#define SECDURATAINCENDIO 5

/* dati da proteggere */
int pagliabrucia=0;
int nMangiatoiePiene = 0;
int mangiatoiePiene[NUMMANGIATOIE];

/* variabili per la sincronizzazione */
pthread_mutex_t  mutex;
pthread_cond_t condMucche;


void attendi( int min, int max) {
        int secrandom=0;
        if( min > max ) return;
        else if ( min == max )
                secrandom = min;
        else
                secrandom = min + ( random()%(max-min+1) );
        do {
                /* printf("attendi %i\n", secrandom);fflush(stdout); */
                secrandom=sleep(secrandom);
                if( secrandom>0 )
                        { printf("sleep interrupted - continue\n"); fflush(stdout); }
        } while( secrandom>0 );
        return;
}
/*
 ritorna 1 se le mangiatoie sono TUTTE piene, 0 altrimenti
int mangiatoiePiene() {
        int i;
        for( i=0; i<NUMMANGIATOIE; i++ ){
                if( mangiatoiePiene[i]==0 ) {
                        return 0;
                }
        }
        return 1;
}
*/
/* ritorna l'indice della prima mangiatoia vuota */
int indiceMangiatoiaVuota() {
        int i;
        for( i=0; i<NUMMANGIATOIE; i++ ){
                if( mangiatoiePiene[i] == 0 ){
                        return i;
                }
```

```
        }
        return -1;
}

void *Vacca (void *arg)
{
        char Plabel[128];
        intptr_t indice;

        indice=(intptr_t)arg;
        sprintf(Plabel,"Vacca%" PRIiPTR "",indice);

        /*  da completare  */

        while ( 1 ) {
                int indicemangiatoia, secmangiare;

                /* la vacca attende di poter mangiare */
                DBGpthread_mutex_lock( &mutex, Plabel );
                while( nMangiatoiePiene >= NUMMANGIATOIE ) {
                        printf("%s: mangiatoie piene, devo aspettare\n", Plabel);
                        DBGpthread_cond_wait( &condMucche, &mutex, Plabel );
                }
                nMangiatoiePiene++;
                indicemangiatoia = indiceMangiatoiaVuota();
                printf("%s: posso mangiare alla mangiatoia %d\n", Plabel, indicemangiatoia);
                mangiatoiePiene[indicemangiatoia] = 1;
                DBGpthread_mutex_unlock( &mutex, Plabel );
                /* la vacca mangia */
                printf("vacca %s mangia in mangiatoia %i\n", Plabel, indicemangiatoia);
                       fflush(stdout);
                /* quale e' il tempo in secondi durante il quale la vacca mangia? */
                secmangiare=MINSECMANGIARE + ( random()%(MAXSECMANGIARE-MINSECMANGIARE+1) );
                /* ogni secondo la vacca guarda se la paglia brucia */
                while( secmangiare > 0 ) {
                        printf("%s: controllo se brucia la paglia\n", Plabel);
                        DBGpthread_mutex_lock( &mutex, Plabel );
                        if( pagliabrucia == 1 ) {
                                printf("%s: sta bruciando!!! me ne vado\n", Plabel);
                                DBGpthread_mutex_unlock( &mutex, Plabel );
                                break;
                        }
                        DBGpthread_mutex_unlock( &mutex, Plabel );
                        sleep( 1 );
                        secmangiare--;
                }
                /* la vacca libera la mangiatoia */
                DBGpthread_mutex_lock( &mutex, Plabel );
                nMangiatoiePiene--;
                mangiatoiePiene[indicemangiatoia] = 0;
                DBGpthread_cond_broadcast( &condMucche, Plabel );
                DBGpthread_mutex_unlock( &mutex, Plabel );

                /* la vacca fa un giro di 10 secondi */
                attendi( SECGIRO, SECGIRO );
                printf("vacca %s finisce giro\n", Plabel);
                       fflush(stdout);

        }
        pthread_exit(NULL);
}


void Bovaro (void)
{
        char Plabel[128];

        sprintf(Plabel,"Bovaro");
```

```
        while( 1 ) {
                /* attesa 30 sec tra incendi */
                attendi( SECINTERVALLOTRAINCENDI, SECINTERVALLOTRAINCENDI );

                /* bovaro incendia paglia */
                DBGpthread_mutex_lock(&mutex,Plabel);
                pagliabrucia=1;
                printf("bovaro incendia paglia\n");
                fflush(stdout);
                DBGpthread_mutex_unlock(&mutex,Plabel);

                /* durata incendio 3 sec */
                attendi( 3, 3 );

                /* bovaro spegne paglia */
                DBGpthread_mutex_lock(&mutex,Plabel);
                pagliabrucia=0;
                printf("paglia spenta\n");
                fflush(stdout);
                DBGpthread_mutex_unlock(&mutex,Plabel);
        }

        pthread_exit(NULL);
}

int main ( int argc, char* argv[] )
{
        pthread_t    th;
        int  rc;
        uintptr_t i=0;
        int seme;

        seme=time(NULL);
        srandom(seme);

        rc = pthread_mutex_init(&mutex, NULL);
        if( rc ) PrintERROR_andExit(rc,"pthread_mutex_init failed");

        pagliabrucia=0; /* la paglia non brucia */

        /* INIZIALIZZATE LE VOSTRE ALTRE VARIABILI CONDIVISE / fate voi */
        for( i=0; i<NUMMANGIATOIE; i++ ){
                mangiatoiePiene[i] = 0;
        }

        /* CREAZIONE PTHREAD */
        for(i=0;i<NUMVACCHE;i++) {
                rc=pthread_create(&th,NULL,Vacca,(void*)i);
                if(rc) PrintERROR_andExit(rc,"pthread_create failed");
        }

        Bovaro();

        return(0);
}
```

12 FUNIVIA SEMPLICE

Dal centro del paese di Gattolino di Cesena una piccola funivia sale fino in cima al campanile per ammirare l'autostrada A14 dall'alto e poi ridiscende subito riportando giù i passeggeri. Nella funivia devono viaggiare esattamente due persone sobrie per volta. Gli ubriachi non sono ammessi. Da terra, la funivia fa salire due persone, poi chiude le porte, stampa a video la composizione dei passeggeri (gli indici delle persone), ed infine parte, ed impiega un secondo per giungere in cima al campanile e poi subito ridiscende impiegando un altro secondo. Tornata giù la funivia apre le porte e fa uscire i passeggeri. Dopo che tutti i passeggeri sono usciti allora possono entrarne degli altri. Al termine del giro panoramico, le persone che scendono dalla seggiovia gridano di gioia (stampare un avviso) e DOPO QUESTO si rimettono subito in fila per fare ancora un giro. A sfruttare la funivia ci sono 4 sobri

```
#define MAX_PERSONE_IN_FUNIVIA 2
#define PERSONE 4

int funiviaInTransito = 0;
intptr_t indicePersoneSuFunivia[MAX_PERSONE_IN_FUNIVIA];

pthread_mutex_t mutex;
pthread_cond_t condPasseggeriInAttesa;
pthread_cond_t condPasseggeriInTransito;
pthread_cond_t condFunivia;

int personeSuFunivia() {
    int i, nPersone = 0;
    for( i=0; i<MAX_PERSONE_IN_FUNIVIA; i++ ){
        if(indicePersoneSuFunivia[i] >= 0) {
            nPersone++;
        }
    }
    return nPersone;
}

int cercaPostoInFunivia() {
    int i;
    for( i=0; i<MAX_PERSONE_IN_FUNIVIA; i++ ) {
        if( indicePersoneSuFunivia[i] < 0 ){
            return i;
        }
    }
    return -1;
}

void *sobrio( void *arg ) {

    int mioPostoInFunivia;
    char Slabel[128];
    intptr_t mioIndice = ( intptr_t )arg;
    sprintf( Slabel, "Persona sobria %" PRIiPTR "", mioIndice );

    while( 1 ) {
        DBGpthread_mutex_lock( &mutex, Slabel );
        while( funiviaInTransito || personeSuFunivia() >= MAX_PERSONE_IN_FUNIVIA ) {
            DBGpthread_cond_wait( &condPasseggeriInAttesa, &mutex, Slabel );
        }
        mioPostoInFunivia = cercaPostoInFunivia();
        indicePersoneSuFunivia[mioPostoInFunivia] = mioIndice;
        /* la funivia è piena */
        if( personeSuFunivia() >= MAX_PERSONE_IN_FUNIVIA ){
            DBGpthread_cond_signal( &condFunivia, Slabel );
        }
        /* attendono che il giro sia finito */
```

```
        DBGpthread_cond_wait( &condPasseggeriInTransito, &mutex, Slabel );
        /* giro finito */
        indicePersoneSuFunivia[mioPostoInFunivia] = -1;
        if( personeSuFunivia() <= 0 ) {
            DBGpthread_cond_signal( &condFunivia, Slabel );
        }
        printf( "%s: Arrivato in fondo, seee!!\n", Slabel );
        DBGpthread_mutex_unlock( &mutex, Slabel );
    }

    pthread_exit( NULL );

}

void *funivia( void *arg ){

    int i;
    char Flabel[128];
    sprintf( Flabel, "funivia" );

    while( 1 ) {
        DBGpthread_mutex_lock( &mutex, Flabel );
        while( personeSuFunivia() < MAX_PERSONE_IN_FUNIVIA ) {
            DBGpthread_cond_broadcast( &condPasseggeriInAttesa, Flabel );
            printf("%s: Poche persone in funivia, aspetto\n", Flabel);
            DBGpthread_cond_wait( &condFunivia, &mutex, Flabel );
        }
        for( i=0; i<MAX_PERSONE_IN_FUNIVIA; i++ ) {
            printf( "%s: persona %" PRIiPTR " a bordo\n", Flabel, indicePersoneSuFunivia[i] );
        }
        funiviaInTransito = 1;
        DBGpthread_mutex_unlock( &mutex, Flabel );

        printf( "%s: parte!\n", Flabel );
        sleep( 1 );
        printf( "%s: arrivato in cima\n", Flabel );
        sleep( 1 );
        printf( "%s: arrivato a valle\n", Flabel );

        DBGpthread_mutex_lock( &mutex, Flabel );
        funiviaInTransito = 0;
        DBGpthread_cond_broadcast( &condPasseggeriInTransito, Flabel );
        /* attende che entrambi i passeggeri siano scesi */
        DBGpthread_cond_wait( &condFunivia, &mutex, Flabel );
        DBGpthread_mutex_unlock( &mutex, Flabel );
    }

    pthread_exit( NULL );

}

int main( void ) {

    int rc;
    pthread_t th;
    intptr_t i;

    DBGpthread_mutex_init( &mutex, NULL, "main" );
    DBGpthread_cond_init( &condPasseggeriInAttesa, NULL, "main" );
    DBGpthread_cond_init( &condPasseggeriInTransito, NULL, "main" );
    DBGpthread_cond_init( &condFunivia, NULL, "main" );

    for( i=0; i<MAX_PERSONE_IN_FUNIVIA; i++ ) {
        indicePersoneSuFunivia[i] = -1;
    }

    rc = pthread_create( &th, NULL, funivia, NULL );
    if( rc ) {
```

```
        PrintERROR_andExit( rc, "Creazione funivia" );
    }

    for( i=0; i<PERSONE; i++ ) {
        rc = pthread_create( &th, NULL, sobrio, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione funivia" );
        }
    }

    pthread_exit( NULL );
    return 0;
}
```

13 FUNIVIA DIFFICILE

La piccola funivia che sale dal centro del paese di Gattolino di Cesena fino in cima al campanile e poi ridiscende può far fare il giro panoramico a due persone per volta. Non può viaggiare vuota. Però la legge impone che gli ubriachi debbano viaggiare da soli perché disturbano gli altri. Ogni volta che la funivia deve partire si scatena una rissa selvaggia per entrare: - se il primo che entra è sobrio allora bisogna far entrare un altro sobrio prima di partire, impedendo agli ubriachi di entrare, - se il primo che entra è ubriaco la seggiovia parte con il solo ubriaco a bordo. Una volta fatto il carico, la funivia chiude le porte, stampa a video la composizione dei passeggeri (quanti sobri e quanti ubriachi), ed infine parte, ed impiega un secondo per giungere in cima e poi ridiscende impiegando un altro secondo. Tornata giù la funivia apre le porte e fa uscire i passeggeri. Dopo che tutti i passeggeri sono usciti allora possono entrarne degli altri. Al termine del giro panoramico, le persone che scendono dalla seggiovia gridano di gioia (stampare un avviso) e DOPO QUESTO si rimettono subito in fila per fare ancora un giro. A sfruttare la funivia ci sono 4 sobri e 2 ubriachi.

```
#define CAPIENZA_FUNIVIA 2
#define SOBRI 4
#define UBRIACHI 2

int funiviaInTransito = 0;
int personeUbriacheABordo = 0;
int personeSobrieABordo = 0;

/* 1 per SOBRIO 0 per UBRIACO */
int personeSuFunivia[CAPIENZA_FUNIVIA];

pthread_mutex_t mutex;
pthread_cond_t condPasseggeriInAttesa;
pthread_cond_t condPasseggeriInFunivia;
pthread_cond_t condFunivia;

int contaPersoneInFunivia() {
    return 2*personeUbriacheABordo + personeSobrieABordo;
}

void *Ubriaco( void *arg ) {

    char Ulabel[128];
    sprintf( Ulabel, "Ubriaco U%" PRIiPTR "", ( intptr_t )arg );

    while( 1 ) {
    DBGpthread_mutex_lock( &mutex, Ulabel );
    while( funiviaInTransito || contaPersoneInFunivia() > 0 ) {
        printf( "%s: funivia in transito -> %d, personeInFunivia -> %d\n", Ulabel, funiviaInTransito, contaPersoneInFunivia() );
        DBGpthread_cond_wait( &condPasseggeriInAttesa, &mutex, Ulabel );
    }
    /* dev'essere il primo a salire -> indice per forza 0 */
    personeSuFunivia[0] = 0;
    personeUbriacheABordo++;
    /* appena sale un ubriaco fa la signal alla funivia */
    DBGpthread_cond_signal( &condFunivia, Ulabel );
    DBGpthread_cond_wait( &condPasseggeriInFunivia, &mutex, Ulabel );
    personeSuFunivia[0] = -1;
    personeUbriacheABordo--;
    printf( "%s: evviva!\n", Ulabel );
    DBGpthread_cond_signal( &condFunivia, Ulabel );
    DBGpthread_mutex_unlock( &mutex, Ulabel );
    }

    pthread_exit( NULL );
```

```
}

int cercaPostoInFunivia() {
    int i;
    for( i=0; i<CAPIENZA_FUNIVIA; i++ ){
        if( personeSuFunivia[i]==-1 ){
            return i;
        }
    }
    return -1;
}

void *Sobrio( void *arg ) {

    char Slabel[128];
    int mioPosto;
    sprintf( Slabel, "Sobrio S%" PRIiPTR "", ( intptr_t )arg );

    while( 1 ){
        DBGpthread_mutex_lock( &mutex, Slabel );
        while( funiviaInTransito || contaPersoneInFunivia() >= CAPIENZA_FUNIVIA ){
            DBGpthread_cond_wait( &condPasseggeriInAttesa, &mutex, Slabel );
        }
        personeSobrieABordo++;
        mioPosto = cercaPostoInFunivia();
        personeSuFunivia[mioPosto] = 1;
        if( contaPersoneInFunivia() == CAPIENZA_FUNIVIA ) {
            DBGpthread_cond_signal( &condFunivia, Slabel );
        }
        DBGpthread_cond_wait( &condPasseggeriInFunivia, &mutex, Slabel );
        personeSobrieABordo--;
        personeSuFunivia[mioPosto] = -1;
        if( contaPersoneInFunivia() <= 0 ) {
            DBGpthread_cond_signal( &condFunivia, Slabel );
        }
        printf( "%s: evviva!\n", Slabel );
        DBGpthread_mutex_unlock( &mutex, Slabel );
    }

    pthread_exit( NULL );

}

void stampaInfoPersoneABordo() {
    int i;
    char infoPersona[128];
    for( i=0; i<CAPIENZA_FUNIVIA; i++ ) {
        if( personeSuFunivia[i]==1 ) {
            sprintf( infoPersona, "SOBRIO " );
        } else if ( personeSuFunivia[i] == 0 ) {
            sprintf( infoPersona, "UBRIACO " );
        } else {
            sprintf( infoPersona, "<VUOTO> " );
        }
        printf( "Funivia: %s\n", infoPersona );
    }
}

void *Funivia( void *arg ){

    char Flabel[128];
    sprintf( Flabel, "Funivia" );

    while( 1 ){
        DBGpthread_mutex_lock( &mutex, Flabel );
        while( ( personeSobrieABordo < CAPIENZA_FUNIVIA && personeUbriacheABordo <= 0 )
            || ( personeSobrieABordo <= 0 && personeUbriacheABordo < 1 ) ) {
            DBGpthread_cond_wait( &condFunivia, &mutex, Flabel );
```

```
        }
        stampaInfoPersoneABordo();
        funiviaInTransito = 1;
        DBGpthread_mutex_unlock( &mutex, Flabel );

        printf( "%s: parte!\n", Flabel );
        sleep( 1 );
        printf( "%s: arrivato in cima\n", Flabel );
        sleep( 1 );
        printf( "%s: arrivato a valle\n", Flabel );

        DBGpthread_mutex_lock( &mutex, Flabel );
        DBGpthread_cond_broadcast( &condPasseggeriInFunivia, Flabel );
        DBGpthread_cond_wait( &condFunivia, &mutex, Flabel );
        funiviaInTransito = 0;
        DBGpthread_cond_broadcast( &condPasseggeriInAttesa, Flabel );
        DBGpthread_mutex_unlock( &mutex, Flabel );
    }

    pthread_exit( NULL );

}

int main( void ) {

    int rc;
    pthread_t th;
    intptr_t i;

    DBGpthread_mutex_init( &mutex, NULL, "main" );
    DBGpthread_cond_init( &condPasseggeriInAttesa, NULL, "main" );
    DBGpthread_cond_init( &condPasseggeriInFunivia, NULL, "main" );
    DBGpthread_cond_init( &condFunivia, NULL, "main" );

    for( i=0; i<CAPIENZA_FUNIVIA; i++ ) {
        personeSuFunivia[i] = -1;
    }

    rc = pthread_create( &th, NULL, Funivia, NULL );
    if( rc ) {
        PrintERROR_andExit( rc, "Creazione funivia" );
    }

    for( i=0; i<SOBRI; i++ ) {
        rc = pthread_create( &th, NULL, Sobrio, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione funivia" );
        }
    }

    for( i=0; i<UBRIACHI; i++ ) {
        rc = pthread_create( &th, NULL, Ubriaco, ( void *)i );
        if( rc ) {
            PrintERROR_andExit( rc, "Creazione funivia" );
        }
    }

    pthread_exit( NULL );
    return 0;
}
```

14 PONTE PERICOLANTE SEMPLICE

Nell'ameno paesino di Villa Inferno, tra Cesena e Cervia, c'è una unica strada a due corsie fatta come un anello che gira tutto attorno al paese. Le auto girano continuamente in tondo, senza smettere mai. In un tratto della strada c'è un ponte pericolante e con una sola corsia, così solo un' auto alla volta può attraversare il ponte. Ciascuna auto, quindi, può attraversare solo quando nessuna altra auto è sul ponte. Ma bisogna anche rispettare un turno tra le auto del proprio lato. A tal scopo, su ciascun lato del ponte c'è un distributore di biglietti numerati crescenti che determinano l'ordine di attraversamento del ponte per le macchine che partono da quel lato. Ciascuna auto che vuole attraversare prende un biglietto sul proprio lato e attende il proprio turno tra quelle del proprio lato. Per smaltire le code, la regola di precedenza stabilisce che attraversa il ponte l'auto col biglietto più piccolo tra quelle che stanno sul lato in cui ci sono attualmente più auto in attesa di attraversare. In caso di parità, attraversa l'auto che gira in senso orario. Ciascuna auto impiega 1 secondo a percorrere il ponte e altri 5 secondi per percorrere il resto dell'anello. Ci sono 4 auto che viaggiano in senso orario e altre 4 in senso antiorario. All'inizio le auto si trovano all'ingresso del ponte (ciascuna sul lato dipendente dal proprio verso di percorrenza).

```
#define NUMAUTOORARIO 4
#define NUMAUTOANTIORARIO 4

#define INDICESENSOORARIO 0
#define INDICESENSOANTIORARIO 1

int bigliettoDistributore[2];
int autoInCoda[2];
int biglietto[2];

int ponteOccupato = 0;

pthread_mutex_t  mutexDistributori;
pthread_mutex_t  mutex;
pthread_cond_t   condPonteLibero;

void *Auto (void *arg)
{
        char Plabel[128];
        intptr_t indice;
        int myBiglietto, indiceSenso;
        char senso; /* O orario  A antiorario */

        indice=(intptr_t)arg;
        if( indice<NUMAUTOORARIO )
        {       senso='O'; indiceSenso=INDICESENSOORARIO; }
        else
        {       senso='A'; indiceSenso=INDICESENSOANTIORARIO;

        sprintf(Plabel,"%cA%" PRIiPTR "", senso, indice);

        while(1) {
                DBGpthread_mutex_lock(&mutexDistributori,Plabel);
                myBiglietto=bigliettoDistributore[indiceSenso];
                bigliettoDistributore[indiceSenso]++;
                DBGpthread_mutex_unlock(&mutexDistributori,Plabel);

                printf("auto %s ha preso biglietto %i \n", Plabel, myBiglietto);
                fflush(stdout);

                DBGpthread_mutex_lock( &mutex, Plabel );
                autoInCoda[indiceSenso]++;
                while( ponteOccupato
```

```
                || autoInCoda[indiceSenso] < autoInCoda[!indiceSenso]
                || myBiglietto != biglietto[indiceSenso]
                || ( autoInCoda[indiceSenso] == autoInCoda[!indiceSenso] && indiceSenso == INDICESENSOANTIORARIO ) ) {
                        DBGpthread_cond_wait( &condPonteLibero, &mutex, Plabel );
                }
                autoInCoda[indiceSenso]--;
                ponteOccupato = 1;
                DBGpthread_mutex_unlock( &mutex, Plabel );

                printf("auto %s inizia attraversamento ponte con biglietto %i \n", Plabel,
                        myBiglietto );
                fflush(stdout);

                /* Auto comincia ad attraversare il ponte */

                sleep(1); /* auto attraversa il ponte */

                /* auto finisce attraversamento ponte
                   ed avvisa di avere finito l'attraversamento */

                DBGpthread_mutex_lock( &mutex, Plabel );
                ponteOccupato = 0;
                biglietto[indiceSenso]++;
                DBGpthread_cond_broadcast( &condPonteLibero, Plabel );
                DBGpthread_mutex_unlock( &mutex, Plabel );

                /* auto fa un giro intorno */
                printf("auto %s gira intorno \n", Plabel );
                fflush(stdout);
                sleep(5); /* faccio il giro attorno a Villa Inferno */
        }

        pthread_exit(NULL);
}


int main ( int argc, char* argv[] )
{
        pthread_t    th;
        int  rc;
        intptr_t i;

        rc = pthread_cond_init(&condPonteLibero, NULL);
        if( rc ) PrintERROR_andExit(rc,"pthread_cond_init failed");

        rc = pthread_mutex_init(&mutex, NULL);
        if( rc ) PrintERROR_andExit(rc,"pthread_mutex_init failed");

        rc = pthread_mutex_init(&mutexDistributori, NULL);
        if( rc ) PrintERROR_andExit(rc,"pthread_mutex_init failed");

        /* configuro le variabili */
        for( i=0; i<2; i++ ) {
                bigliettoDistributore[i]=0;
                biglietto[i]=0;
                autoInCoda[i]=0;
        }

        for(i=0;i<NUMAUTOORARIO+NUMAUTOANTIORARIO;i++) {
                rc=pthread_create(&th,NULL,Auto,(void*)i);
                if(rc) PrintERROR_andExit(rc,"pthread_create failed");
        }

        pthread_exit(NULL);
        return(0);
}
```

15 PONTE PERICOLANTE COMPLICATO

Nell'ameno paesino di Villa Inferno, tra Cesena e Cervia, c'è una unica strada a due corsie fatta come un anello che gira tutto attorno al paese. Le auto girano continuamente in tondo, senza smettere mai. In un tratto della strada c'è un ponte pericolante con una strettoia, così le auto possono attraversare solo in un senso di marcia per volta. Un'auto può attraversare solo se è vera una delle due seguenti condizioni.

1. Nessuna altra auto è sul ponte,
2. sul ponte c'è ancora un auto, ma questa ha già percorso almeno metà del ponte e va nello stesso verso dell'auto che vuole attraversare.

Ma bisogna anche rispettare un turno tra le auto del proprio lato. A tal scopo, su ciascun lato del ponte c'è un distributore di biglietti numerati crescenti che determinano l'ordine di attraversamento del ponte per le macchine che partono da quel lato. Ciascuna auto che vuole attraversare prende un biglietto sul proprio lato e attende il proprio turno tra quelle del proprio lato. Quando nessuna auto è sul ponte, la regola di precedenza stabilisce che può attraversare il ponte l'auto col biglietto più piccolo tra quelle che stanno sul lato in cui ci sono attualmente più auto in attesa di attraversare. In caso di parità, attraversa l'auto che gira in senso orario. Ciascuna auto impiega 1 secondo a percorrere ciascuna delle metà del ponte e altri 10 secondi per percorrere il resto dell'anello. Ci sono 4 auto che viaggiano in senso orario e altre 4 in senso antiorario. All'inizio le auto sono all'ingresso del ponte (secondo il loro verso di percorrenza).

```
#define NUMAUTOORARIO 4
#define NUMAUTOANTIORARIO 4
#define INDICESENSOORARIO 0
#define INDICESENSOANTIORARIO 1

/* variabili condivise da proteggere */
int bigliettoDistributore[2];
int biglietto[2];
int autoInCoda[2];
/* 0 se ponte è vuoto, 1 se ponte pieno, 2 se prima auto è passata a metà */
int ponteOccupato = 0;
int sensoAutoCheAttraversa = 0;

pthread_mutex_t  mutexDistributori;
pthread_mutex_t mutex;
pthread_cond_t condAttesaAuto;

void *Auto (void *arg)
{
        char Plabel[128];
        intptr_t indiceAuto;
        int myBiglietto, indiceSenso;
        char senso; /* O orario  A antiorario */

        /* aggiungete le vostre variabili locali */

        indiceAuto=(intptr_t)arg;
        if( indiceAuto<NUMAUTOORARIO )
        {       senso='O'; indiceSenso=INDICESENSOORARIO; }
        else
        {       senso='A'; indiceSenso=INDICESENSOANTIORARIO; }

        sprintf(Plabel,"%c%" PRIiPTR "", senso, indiceAuto);

        while(1) {
                /* auto prende il biglietto */
                DBGpthread_mutex_lock(&mutexDistributori,Plabel);
```

```
                myBiglietto=bigliettoDistributore[indiceSenso];
                bigliettoDistributore[indiceSenso]++;
                DBGpthread_mutex_unlock(&mutexDistributori,Plabel);

                printf("auto %s ha preso biglietto %i \n", Plabel, myBiglietto);
                fflush(stdout);

                DBGpthread_mutex_lock( &mutex, Plabel );
                autoInCoda[indiceSenso]++;
                while( ponteOccupato == 1
                        || ( ponteOccupato == 2 && sensoAutoCheAttraversa != indiceSenso )
                        || myBiglietto != biglietto[indiceSenso]
                        || autoInCoda[indiceSenso] < autoInCoda[!indiceSenso]
                        || ( autoInCoda[indiceSenso] == autoInCoda[!indiceSenso] && indiceSenso == INDICESENSOANTIORARIO ) )
                        {
                        DBGpthread_cond_wait( &condAttesaAuto, &mutex, Plabel );
                }
                autoInCoda[indiceSenso]--;
                biglietto[indiceSenso]++;
                ponteOccupato = 1;
                sensoAutoCheAttraversa = indiceSenso;
                DBGpthread_mutex_unlock( &mutex, Plabel );

                printf("auto %s attraversa ponte biglietto %i \n", Plabel,
                        myBiglietto );
                fflush(stdout);

                sleep(1); /* attraverso la prima meta' del ponte */

                printf("auto %s supera meta\' ponte \n", Plabel );
                fflush(stdout);

                DBGpthread_mutex_lock( &mutex, Plabel );
                ponteOccupato = 2;
                DBGpthread_cond_broadcast( &condAttesaAuto, Plabel );
                DBGpthread_mutex_unlock( &mutex, Plabel );

                sleep(1); /*attraverso seconda meta' del ponte */

                DBGpthread_mutex_lock( &mutex, Plabel );
                ponteOccupato = 0;
                DBGpthread_cond_broadcast( &condAttesaAuto, Plabel );
                DBGpthread_mutex_unlock( &mutex, Plabel );

                printf("auto %s gira intorno\n", Plabel );
                fflush(stdout);
                sleep(10); /* faccio il giro attorno a Villa Inferno */
        }

        pthread_exit(NULL);
}


int main ( int argc, char* argv[] )
{
        pthread_t    th;
        int  rc;
        intptr_t i;

        rc = pthread_mutex_init(&mutexDistributori, NULL);
        if( rc ) PrintERROR_andExit(rc,"pthread_mutex_init failed");

        /* configuro le variabili */
        for( i=0; i<2; i++ ) {
                bigliettoDistributore[i]=0;
                biglietto[i]=0;
                autoInCoda[i] = 0;
        }
```

```
        DBGpthread_mutex_init( &mutexDistributori, NULL, "main" );
        DBGpthread_mutex_init( &mutex, NULL, "main" );
        DBGpthread_cond_init( &condAttesaAuto, NULL, "main" );

        for(i=0;i<NUMAUTOORARIO+NUMAUTOANTIORARIO;i++) {
                rc=pthread_create(&th,NULL,Auto,(void*)i);
                if(rc) PrintERROR_andExit(rc,"pthread_create failed");
        }

        pthread_exit(NULL);
        return(0);
}
```

16 MAKEFILE

```
CFLAGS=-ansi -Wpedantic -Wall -Werror -D_THREAD_SAFE -D_REENTRANT -D_POSIX_C_SOURCE=200112L
LIBRARIES=-lpthread
LFLAGS=

all: sorgente.exe

sorgente.exe: sorgente.o DBGpthread.o
	gcc ${LFLAGS} -o sorgente.exe sorgente.o sorgente.o ${LIBRARIES}

sorgente.o: sorgente.c DBGpthread.h
	gcc -c ${CFLAGS} sorgente.c

DBGpthread.o: DBGpthread.c printerror.h
	gcc -c ${CFLAGS} DBGpthread.c

.PHONY: clean run

clean:
	rm -f *.exe *.o *~ core
```