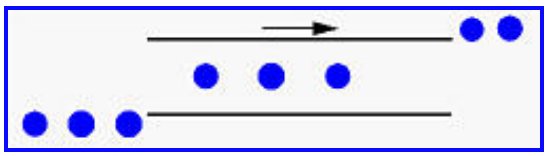
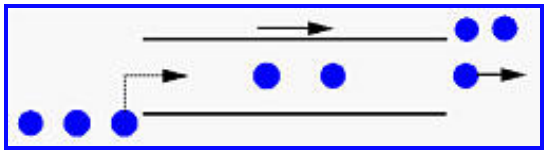
Bridge Crossing

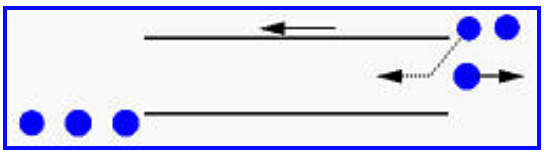
Consider a narrow bridge that can only allow three vehicles in the same direction to cross at the same time. If there are three vehicles on the bridge, any incoming vehicle must wait as shown in the following figure.



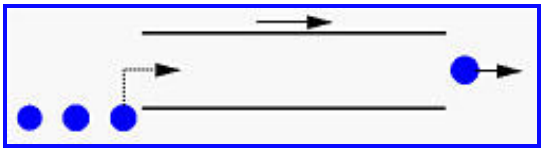
When a vehicle exits the bridge, we have two cases to consider. **Case 1**, there are other vehicles on the bridge; and **Case 2** the exiting vehicle is the last one on bridge. **Case 1** is shown below. In this case, one vehicle in the same direction should be allowed to proceed.



Case 2 is more complicated and has two subcases. In this case, the exiting vehicle is the last vehicle on the bridge. If there are vehicles waiting in the opposite direction, one of them can proceed. This is illustrated below:



Or, if there is no vehicle waiting in the opposite direction, then let the waiting vehicle in the same direction to proceed:



Problem Statement

Given the problem above, design a monitor to ``control'' this bridge. More precisely, design a monitor with procedures **ArriveBridge()** and **ExitBridge()**. When a vehicle arrives at the bridge, function **ArriveBridge()** is called and when it exits the bridge, function **ExitBridge()** is called. As always, an initialization function is required.

Implementation

Our implementation consists of two files, a header file and an implementation file. The following is the header file **bridge-m.h**. Both **ArriveBridge()** and **ExitBridge()** require one argument giving the direction of the vehicle.

Click **here** to download a copy of this file.

```
void  BridgeInit(void);
void  ArriveBridge(int Direction);
void  ExitBridge(int Direction);
```

The following is the implementation file **bridge-m.c**.

Click **here** to download a copy of this file.

```
#include  <thread.h>

#include  "bridge-m.h"

#define   MAX_VEHICLE  3               /* max vehicle on bridge    */
#define   TRUE         1
#define   FALSE        0
#define   EAST         0               /* east bound               */
#define   WEST         1               /* west bound               */

static int  CurrentDirection;          /* current direction of cars*/
static int  VehicleCount;              /* # of vehicle on bridge   */
static int  Waiting[2];                /* # east/west bound waiting*/

static mutex_t  MonitorLock;           /* monitor lock             */
static cond_t   EastWest[2];           /* blocking east/west cars  */

void  BridgeInit(void)
{
     VehicleCount = 0;                 /* no vehicle on bridge     */
     Waiting[0] = Waiting[1] = 0;      /* no vehicle waiting       */

     mutex_init(&MonitorLock, USYNC_THREAD, (void *) NULL);
     cond_init(&EastWest[0]), USYNC_THREAD, (void *) NULL);
     cond_init(&EastWest[1]), USYNC_THREAD, (void *) NULL);
}

static int  isSafe(int Direction)
{
     if (VehicleCount == 0)            /* if no vehicle on bridge  */
          return  TRUE;                /* safe to cross            */
     else if ((VehicleCount < MAX_VEHICLE) && (CurrentDirection == Direction))
          return  TRUE;                /* if < 3 in same direction */
     else
          return  FALSE;               /* otherwise, do not procee */
}

void  ArriveBridge(int Direction)
{
     mutex_lock(&MonitorLock);         /* lock the monitor         */
          if (!isSafe(Direction)) {
               Waiting[Direction]++;   /* no, wait at the bridge   */
               while (!isSafe(Direction))    /* safe to cross?      */
                    cond_wait(&EastWest[Direction]), &MonitorLock);
               Waiting[Direction]--;   /* go back to test again    */
          }
          VehicleCount++;              /* can proceed              */
          CurrentDirection = Direction; /* set direction           */
     mutex_unlock(&MonitorLock);        /* release monitor          */
}

void  ExitBridge(int Direction)
{
     mutex_lock(&MonitorLock);         /* lock the monitor         */
          VehicleCount--;              /* one vehicle exits        */
          if (VehicleCount > 0)        /* have vehicles on bridge? */
               cond_signal(&EastWest[Direction]));/* yes,same dir*/
```

```
            else {                          /* bridge is empty          */
                 if (Waiting[1-Direction] != 0)  /* any opposite wait?*/
                      cond_signal(&EastWest[1-Direction]); /* yes */
                 else                       /* no, release the same dir */
                      cond_signal(&EastWest[Direction]));
            }
       mutex_unlock(&MonitorLock);          /* release the monitor      */
}
```

- In this implementation, the directions are named **EAST** and **WEST**. Mutex lock **MonitorLock** is for locking the monitor, and condition variables **EastWest[0]** and **EastWest[1]** are used to block the east bound and west bound vehicles, respectively.
- The monitor must keep track the current direction with **CurrentDirection**, the number of vehicles on bridge with **VehicleCount**, and the number of east bound and west bound waiting vehicles with **Waiting[0]** and **Waiting[1]**. Since all of these variables are only used *within* this monitor, they are declared with **static**.
- This monitor has a function **isSafe()** which is only used *within* the monitor. Thus, it is also declared with **static**. It receives a direction. If there is no vehicle on the bridge, it is safe to let the calling vehicle proceed. Otherwise, if there are less than three vehicles on the bridge and the calling vehicle has a direction the same as the current direction of the vehicles on the bridge, it is also safe to let this vehicle proceed. Otherwise, it is not in a safe situation.
- Function **ArriveBridge()** takes a direction argument. If the direction is not safe, the number of waiting vehicles in the same direction is increased by one and the calling vehicle is put into the queue of the condition variable corresponding to the direction. After a vehicle is awakened and if its direction is safe, the number of waiting vehicles in that direction is decreased by 1, the number of vehicles on the bridge is increased by 1 and exits the monitor. Thus, this vehicle is allowed to be on the bridge.
- As discussed in the beginning, the exit part is more complicated. When a vehicle exits, it calls **ExitBridge()** with its direction. The first thing for **ExitBridge()** to do is subtracting 1 from **VehicleCount**. If the number of vehicles on the bridge is greater than zero, then a vehicle waiting in the same direction is awakened by signaling the corresponding condition variable. If there is no vehicle on the bridge, the number of vehicle waiting in the opposite direction is checked. If it is not zero, let one of the vehicle waiting there proceed; otherwise, let one vehicle waiting in the same direction proceed. What if there is no vehicle in both ends? **ExitBridge()** just returns and the next incoming vehicle will set the direction.

The Main Program

The following program generates a number of vehicles, each of which crosses the bridge several times. When a vehicle thread is created, a number (*i.e.*, the vehicle number) is assigned to it. Function **OneVehicle()** takes this number and starts to simulate bridge crossing. Each vehicle crosses the bridge **Max_Run** times. For each time, a random number in the range of 0 and 1 is generated. If the random number is less than or equal to 0.5, it is east bound; otherwise, it is west bound. Then, it calls **ArriveBridge()** asking permission to cross the bridge. After this function returns, the vehicle crosses the bridge. Then, it rests for some time and calls **ExitBridge()** to exit the bridge. This completes one iteration. Note that the screen and random number generator are protected by mutex locks **Screen** and **RandomNumber**.

Click **here** to download a copy of this file.

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <time.h>
#include  <thread.h>

#include  "bridge-m.h"
```

```
#define   MAX_CROSSING  20
#define   BIG           ((double) 0x7FFF)

#define   MAX_THREADS   20

mutex_t   Screen;
mutex_t   RandomNumber;

int       Max_Run;

void  Filler(char x[], int n)
{
     int  i;

     for (i = 0; i < n*2; i++)
          x[i] = ' ';
     x[i] = '\0';
}

void  *OneVehicle(void *voidPTR)
{
     int     *intPTR = (int *) voidPTR;
     int     ID = *intPTR;
     int     Direction;
     char    *Dir[2] = { "<--", "-->" };
     char    space[200];
     int     i;
     double  D;

     Filler(space, ID);
     mutex_lock(&Screen);
          printf("%sVehicle %d started ...\n", space, ID);
     mutex_unlock(&Screen);
     for (i = 1; i <= Max_Run; i++) {   /* for each crossing   */
          thr_yield();                  /* rest for a while         */
          mutex_lock(&RandomNumber);    /* lock random # generator  */
               D = rand() / BIG;        /* generate a random number */
          mutex_unlock(&RandomNumber);  /* release random # gen.    */
          Direction = (D <= 0.5) ? 0 : 1;    /* which direction?    */
          mutex_lock(&Screen);
               printf("%sVehicle %d (%d) arrives at the bridge in "
                      "direction %s\n", space, ID, i, Dir[Direction]);
          mutex_unlock(&Screen);
          ArriveBridge(Direction);      /* arrive at the bridge     */
          mutex_lock(&Screen);
               printf("%sVehicle %d (%d) crosses the bridge\n", space, ID, i);
          mutex_unlock(&Screen);
          thr_yield();                  /* crossing the bridge      */
          ExitBridge(Direction);        /* exit the bridge          */
          mutex_lock(&Screen);
               printf("%sVehicle %d (%d) is done\n", space, ID, i);
          mutex_unlock(&Screen);
     }
     mutex_lock(&Screen);
          printf("%sVehicle %d is gone forever ...\n", space, ID);
     mutex_unlock(&Screen);
     thr_exit(0);
}

void  main(int argc, char *argv[])
{
     thread_t   ID[MAX_THREADS];         /* vehicle ID               */
     size_t     Status[MAX_THREADS];     /* vehicle status           */
     int        Arg[MAX_THREADS];        /* vehicle argument         */
     int        Threads;                 /* # of vehicles            */
```

```
        int        i;

        if (argc != 3)  {
             printf("Use %s #-of-iterations #-of-vehicles\n", argv[0]);
             exit(0);
        }
        Max_Run = abs(atoi(argv[1]));
        Threads = abs(atoi(argv[2]));
        if (Threads > MAX_THREADS) {
             printf("The no. of vehicles is too large.  Reset to %d\n",
                    MAX_THREADS);
             Threads = MAX_THREADS;
        }

        printf("Parent started ...\n");

        mutex_init(&Screen, USYNC_THREAD, (void *) NULL);
        BridgeInit();
        srand((unsigned) time(NULL));

        for (i = 0; i < Threads; i++) {     /* start vehicles          */
             Arg[i] = i+1;
             thr_create(NULL, 0, OneVehicle, (void *) &(Arg[i]),
                        0, (void *) &(ID[i]));
        }
        for (i = 0; i < Threads; i++)       /* wait for vehicles       */
             thr_join(ID[i], 0, (void *) &(Status[i]));

        printf("Parent exits ...\n");
}
```