Student Name: Sumeet Kumar Sahoo
Student Regd. No – 11712534
Email id – sumeetsahoo7@gmail.com
GitHub link - https://github.com/sumu007/Operating-System-Project

---

/* You must implement the traffic flow mechanism by defining a structure struct bridge.

- ✓ **Traffic can flow in only a single direction on the bridge at a time.**
- ✓ **Any number of cars can be on the bridge at the same time, as long as they are all traveling in the same direction.**
- ✓ **To avoid starvation, you must implement the "five car rules": once 5 or more consecutive northbound cars have entered the bridge, if there are any southbound cars waiting then no more northbound cars may enter the bridge until some southbound cars have crossed. A similar rule also applies once 5 or more consecutive southbound cars have entered the bridge.**

You must write your solution using the functions for locks and condition variables:

- ✓ **lock_init (struct lock *lock)**
- ✓ **lock_acquire (struct lock *lock)**
- ✓ **lock_release (struct lock *lock)**
- ✓ **cond_init (struct condition *cond)**
- ✓ **cond_wait (struct condition *cond, struct lock *lock) cond_signal (struct condition *cond, struct lock *lock)**
- ✓ **cond_broadcast (struct condition *cond, struct lock *lock) */**

```
struct bridge {
int nwait;
int ncross;
int nconsec;
int swait;
int scross;
```

```
int sconsec;
struct lock *lock;
struct condition nboundover;
struct condition sboundover;
}
```

/*Write a declaration for struct bridge using function bridge_init, which will be invoked to initialize the bridge*/

```
void bridge_init (struct bridge *b)
{
b->nwait = 0;
b->ncross = 0;
b->nconsec = 0;
b->swait = 0;
b->scross = 0
b->sconsec = 0;
lock_init(&b->lock);
cond_init(&b-> nboundover);
cond_init(&b-> sboundover);
}
```

/*When a northbound car arrives at the bridge, it invokes the function:
bridge_arrive_north (struct bridge *b)

This function must not return until it is safe for the car to cross the bridge, according to the rules above*/

```
int bridge_arrive_north (struct bridge *b)
{
lock_acquire(&b->lock);
b->nwait++;
while ((b->scross> 0) || ((b->swait > 0) && (b->nconsec >= 5)) {
cond_wait (&b->sboundover, &b->lock);
}
```

```
b->nwait--;
b->ncross++;
b->nconsec++;
b->sconsec = 0;
lock_release(&b->lock);
}
```

/*Once a northbound car has finished crossing the bridge it will invoke the function*: bridge_leave_north (struct bridge *b) */

```
int bridge_leave_north (struct bridge *b)
{
lock_acquire(&b->lock);
b->ncross--;
if (b->ncross == 0) {
cond_broadcast(&b->nboundover, &b->lock);
}
lock_release(&b->lock);
}
```

/*(Southbound cars will invoke analogous functions bridge_arrive_south and bridge_leave_south) */

```
int bridge_arrive_south (struct bridge *b)
{
lock_acquire(&b->lock);
b->swait++;
while ((b->ncross > 0) || ((b->nwait > 0) && (b->sconsec >= 5)) {
cond_wait(&b->nboundover, &b->lock);
}
b->swait--;
b->scross++;
b->sconsec++;
```

```
b->nconsec = 0;
lock_release(&b->lock);
}


int bridge_leave_south (struct bridge *b)
{
lock_acquire(&b->lock);
b->scross--;
if (b->scross == 0) {
cond_broadcast(&b->sboundover, &b->lock);
}
lock_release(&b->lock);
}
```

Description

Designed a code which helps in the proper synchronization of cars on the bridge. In multithreading, when you want to sleep a thread, condition variable is used.

(1.) To wait or sleep these are used

In C:

```
pthread_cond_wait ();
```

(2.) To wake up sleeping or waiting thread

In C:

```
pthread_cond_signal ();
```

And whenever you want to change value of a variable, lock variable is used.

- ✓ A total of N (Number of cars passing the bridge) cars(threads) are created.
- ✓ Then a thread car enters the bridge via north which calls **bridge_arrive_north (struct bridge *b)**. **This** function must not return until it is safe for the car to cross the bridge, according to the rules.

- ✓ A lock variable is used for the same.
- ✓ Then, the function is conditioned to wait till the car has reached the end and then function **int bridge_leave_north (struct bridge *b**) is invoked**.**
- ✓ Similarly, all the cars entering from south side will invoke the function, **bridge_arrive_south (struct bridge *b)** and **bridge_leave_south (struct bridge *b).**
- ✓ This function will wait till cars cross the bridge and the signals all other condition variable so that they can come out of sleep and execute further instructions.

Algorithms

- ✓ Initialize bridge structure **(struct bridge)**
- ✓ Call bridge_arrive_north **(struct bridge *b)**
- ✓ Call bridge_leave_north **(struct bridge *b)**
- ✓ Call bridge_arrive_south **(struct bridge *b)**
- ✓ Call bridge_leave_south **(struct bridge *b)**

---

struct bridge

Step-1) Defining a bridge

Complexity – O(n)

void bridge_init

Step – 1) Initializing all values of entities of b

Complexity – O(n)

int bridge_arrive_north

Step -1) Initializing lock.

Step – 2) Incrementing all waiting cars.

Step - 3) Repeat Step 2 to while (south crossing > 0) or (south waiting car >0 && cars waiting for north side> = 5.

Step – 4) Condition bound.

Step – 5) Decrement count of north waiting and increment count of north crossing and cars going north consecutively.

Step – 6) Set value of south crossing cars to 0.

Step – 7) Release locks

Complexity – O(n*n*n) = O(n^3).

int bridge_leave_north

Step – 1) Acquire lock

Step – 2) Decrement count of all cars crossing bridge after entering from north side.

Step – 3) when cars of north crossing becomes zero, cond_broadcast that north bound cars are done.

Step – 4) lock is released

Complexity – O(n)

int bridge_arrive_south

Step – 1) Acquire lock

Step – 2) Incrementing all waiting cars.

Step - 3) Repeat Step 2 to while (north crossing > 0) or (north waiting car >0 && cars waiting for south side> = 5.

Step – 4) Condition wait for northbound car to be done.

Step – 5) Decrement count of south waiting and increment count of south crossing and cars going south consecutively.

Step – 6) Set value of south crossing cars to 0.

Step – 7) Release locks

Complexity – O(n*n*n) = O(n^3).

int bridge_leave_south

Step – 1) Acquire lock

Step – 2) Decrement count of all cars crossing bridge after entering from south side.

Step – 3) when cars of south crossing becomes zero, cond_broadcast that south bound cars are done.

Step – 4) lock is released.

Complexity – O(n).

Total Complexity – O(n)+O(n)+O(n^3) +O(n)+O(n^3) +O(n) = O(n^3)

CODE SNIPPET

```
struct bridge {
int nwait;
int ncross;
int nconsec;
int swait;
int scross;
int sconsec;
struct lock *lock;
struct condition nboundover;
struct condition sboundover;
}
```

Purpose of use: -

Initialize the bridge with nwait, ncross, nconsec, swait, scross, sconsec, 3 structures for lock and northbound, southbound condition.

```
void bridge_init (struct bridge *b)
{
b->nwait = 0;
b->ncross = 0;
b->nconsec = 0;
b->swait = 0;
b->scross = 0
b->sconsec = 0;
lock_init(&b->lock);
cond_init(&b-> nboundover);
cond_init(&b-> sboundover);
}
```

Purpose of Use: -

Defining the function bridge_init and initializing all values to zero so that we start from zero cars. Defining locks and cond_init ().

```
int bridge_arrive_north (struct bridge *b)
{
lock_acquire(&b->lock);
b->nwait++;
while ((b->scross> 0) || ((b->swait > 0) && (b->nconsec >= 5)) {
cond_wait (&b->sboundover, &b->lock);
}

b->nwait--;
b->ncross++;
b->nconsec++;
b->sconsec = 0;
lock_release(&b->lock);
}
```

Purpose of use: -

To define a function “bridge_arrive_north” when a car arrives at the north end of the bridge.

Does not return any value till it is safe for all cars to cross the bridge.

Acquires the lock and releases after all conditions are satisfied.

```
int bridge_leave_north (struct bridge *b)
{
lock_acquire(&b->lock);
b->ncross--;
if (b->ncross == 0) {
cond_broadcast(&b->nboundover, &b->lock);
}
lock_release(&b->lock);
}
```

Purpose of use: -

This function is invoked when a car which was northbound leaves the bridge. When the number of cars crossing the bridge is zero, then condition is broadcasted and then lock is released.

```
int bridge_arrive_south (struct bridge *b)
{
lock_acquire(&b->lock);
b->swait++;
while ((b->ncross > 0) || ((b->nwait > 0) && (b->sconsec >= 5)) {
cond_wait(&b->nboundover, &b->lock);
}
b->swait--;
b->scross++;
b->sconsec++;
b->nconsec = 0;
lock_release(&b->lock);
}
```

Purpose of use: -

Whenever a car is southbound, this function is invoked. All cars which are waiting are incremented one by one using b->swait++.

Similar to the bridge_arrive_north (), here also after all conditions are passed, the lock is released.

```
int bridge_leave_south (struct bridge *b)
{
lock_acquire(&b->lock);
b->scross--;
if (b->scross == 0) {
cond_broadcast(&b->sboundover, &b->lock);
}
lock_release(&b->lock);
}
```

Purpose of Use: -

When a southbound car leaves the bridge, this function is invoked. The number is decremented one by one. When the number of cars left is zero, the condition is fulfilled, (&b->sboundover). Hence Lock is released.

Test Cases

1) Number of north bound cars = 5; south bound cars = 3;
2) Number of north bound cars = 6; south bound cars = 0;
3) Number of north bound cars = 0; south bound cars = 5;
4) Number of north bound cars = 3; south bound cars = 3;
5) Number of north bound cars = 3; south bound cars = 5;

Since, traffic can flow in one direction only. The cars which are more in number must be crossing first. Again, if there are more than 5 or more cars entering from one side, no more car can enter from same side if some cars are waiting on other side.

GITHUB LINK - https://github.com/sumu007/Operating-System-Project