Description:

Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and will return them once finished.
The following program segment is used to manage a finite number of instances of an available resource.
When a process wishes to obtain a number of resources, it invokes the decrease count() function.
When a process wants to return a number of resources, it calls the increase count() function.

Algorithm:

```
#include<stdio.h>
#include<errno.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<pthread.h>
#include<time.h>

#define MAX_RESOURCES 5
int available_resources = MAX_RESOURCES;
pthread_mutex_t mtx;

int decrease_count(int count)
{
    pthread_mutex_lock(&mtx);
    if (available_resources < count)
    {
        pthread_mutex_unlock(&mtx);
        return -1;
    }
    else
    {
        available_resources -= count;
        printf("%s %d %s %d %s\n", "Got", count, "resources",
available_resources, "remaining");
```

```
            pthread_mutex_unlock(&mtx);
        }
        return 0;
}
int increase_count(int count)
{
        pthread_mutex_lock(&mtx);
        available_resources += count;
        printf("%s %d %s %d %s\n", "Released", count, "resources",
available_resources, "remaining");
        pthread_mutex_unlock(&mtx);
        return 0;
}

void* f(void* param)
{
        int* i = (int *)param;
        printf("%d\n", *i);
        if(decrease_count(2) != -1)
            increase_count(2);
        return param;
}

int main()
{
        pthread_mutex_init(&mtx, NULL);
        printf("%s %d\n", "MAX_RESOURCES =", MAX_RESOURCES);
        int thread_count = 5;
        pthread_t threads[thread_count];

        for(int i = 0; i < thread_count; i++)
        {
            int a = i;
            if(pthread_create(&threads[i], NULL, f, &a))
            {
                perror(NULL);
                return errno;
            }
        }

        for(int i = 0; i < thread_count; i++)
        {
            if(pthread_join(threads[i], NULL))
```

```
        {
            perror(NULL);
return errno;

 }
    }

    pthread_mutex_destroy(&mtx);
    return 0;
}
```

Description (purpose of use):

When shared data or a common data is access by number of processes that's alright, But race condition is achieved when more than one processes modifies or update that shared data or a common data, So multiple threads and multiple processes are in race to each other for execution with this common data(shared data) thus the method named as race condition.Additionally, the order in which the number of processes modifies it (common data) is unknown so this cause ambiguity and it can be avoided by using Mutex or Semaphores upon critical sections.

Code snippet:

**The maximum number of resources and the number of available resources are declared as follows:

```
//In the code block//
    #define MAX_RESOURCES 5
    int available_resources = MAX_RESOURCES;
```

**When a process wishes to obtain a number of resources, it invokes the decrease count() function:

```
//In the code block//
int decrease_count(int count)
{
    pthread_mutex_lock(&mtx);
    if (available_resources < count)
```

```
    {
        pthread_mutex_unlock(&mtx);
        return -1;
    }
    else
    {
        available_resources -= count;
        printf("%s %d %s %d %s\n", "Got", count, "resources",
available_resources, "remaining");
        pthread_mutex_unlock(&mtx);
    }
    return 0;
}
```

**When a process wants to return a number of resources, it calls the increase count() function:
//In the code block//

```
int increase_count(int count)
{
    pthread_mutex_lock(&mtx);
    available_resources += count;
    printf("%s %d %s %d %s\n", "Released", count, "resources",
available_resources, "remaining");
    pthread_mutex_unlock(&mtx);
    return 0;
}
```

Test cases:

The variable should not be incremented by both the threads. Hence,we need to lock with pthread_mutex_lock and release it afterwards.

```
int increase_count(int count)
{
    pthread_mutex_lock(&mtx);
    available_resources += count;
    printf("%s %d %s %d %s\n", "Released", count, "resources",
available_resources, "remaining");
    pthread_mutex_unlock(&mtx);
    return 0;
}
```

Changing the number of threads.Don’t effect the functioning of the program.

```
void* f(void* param)
{
    int* i = (int *)param;
    printf("%d\n", *i);
    if(decrease_count(2) != -1)
        increase_count(2);
    return param;
}

int main()
{
    pthread_mutex_init(&mtx, NULL);
    printf("%s %d\n", "MAX_RESOURCES =", MAX_RESOURCES);
    int thread_count = 3;
    pthread_t threads[thread_count];

    for(int i = 0; i < thread_count; i++)
    {
        int a = i;
        if(pthread_create(&threads[i], NULL, f, &a))
        {
```

```
File  Edit  View  Search  Terminal  Help
root@Ezio:/media/root/New Volume/osca# gcc -lpthread ca7.c -o ca7
root@Ezio:/media/root/New Volume/osca# ./ca7
MAX_RESOURCES = 5
1
Got 2 resources 3 remaining
Released 2 resources 5 remaining
2
Got 2 resources 3 remaining
Released 2 resources 5 remaining
2
Got 2 resources 3 remaining
Released 2 resources 5 remaining
root@Ezio:/media/root/New Volume/osca#
```

GitHubLink:
https://github.com/pradeepdante/11615922_B50_K1605