Question

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.
- As an example, many commercial software packages provide a given number of licenses, indicating the number of applications that may run concurrently.
- When the application is started, the license count is decremented.
- When the application is terminated, the license count is incremented.
- If all licenses are in use, requests to start the application are denied.
- Such a request will be granted only when an existing license holder terminates the application and a license is returned.
- The preceding program segment produces a race condition. Do the following: (a) Identify the data involved in the race condition. (b) Identify the location (or locations) in the code where the race condition occurs. (c) Using a semaphore or mutex lock, fix the race condition. It is permissible to modify the decrease_count() function so that the calling process is blocked until sufficient resources are available.

Answer

(a) The data involved in the race condition is `available_resources`.

(b) The race condition occurs in the increment available_resources in `increase_count` function and decrement available_resources in `decrease_count` function.

(c) To address this issue, a mutex lock is used to ensure that only one thread can access the shared resource at a time. Modified code is as follows:

```
#define MAX_RESOUCRES 5
int available_resources = MAX_RESOURCES;
pthread_mutex_t mutex;

int decrease_count(int count) {
    if (available_resources < count)
        return -1;
    else {
        pthread_mutex_lock(&mutex);
        available_resources -= count;
        return 0;
    }
}

int increase_count(int count) {
    pthread_mutex_lock(&mutex);
    available_resources += count;
    return 0;
}
```