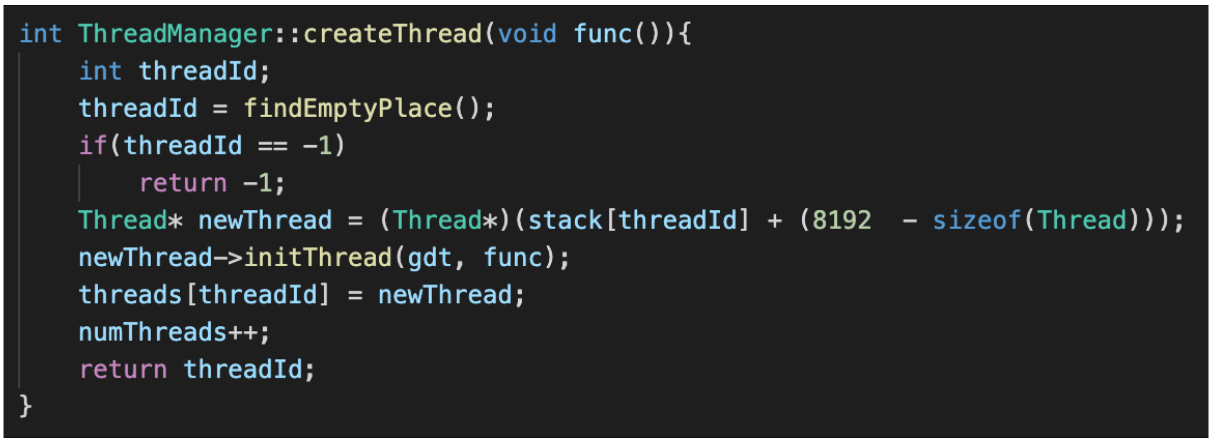
Part 1 - Multithread Library

Create Thread:

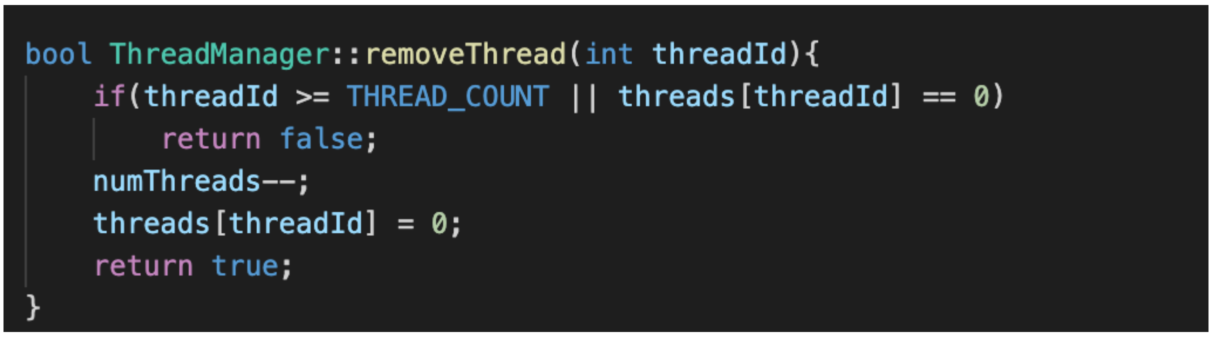


```
int ThreadManager::createThread(void func()){
    int threadId;
    threadId = findEmptyPlace();
    if(threadId == -1)
        return -1;
    Thread* newThread = (Thread*)(stack[threadId] + (8192  - sizeof(Thread)));
    newThread->initThread(gdt, func);
    threads[threadId] = newThread;
    numThreads++;
    return threadId;
}
```

I used thread table method to save the thread instead of using the method in multitasking library which is already defined.

I saved the thread information into the stacks in thread manager and then initialize it. It returns the thread id which will be static in thread table until thread is removed.

Remove Thread:



```
bool ThreadManager::removeThread(int threadId){
    if(threadId >= THREAD_COUNT || threads[threadId] == 0)
        return false;
    numThreads--;
    threads[threadId] = 0;
    return true;
}
```

It removes the thread address from thread table and then decrease numthreads.

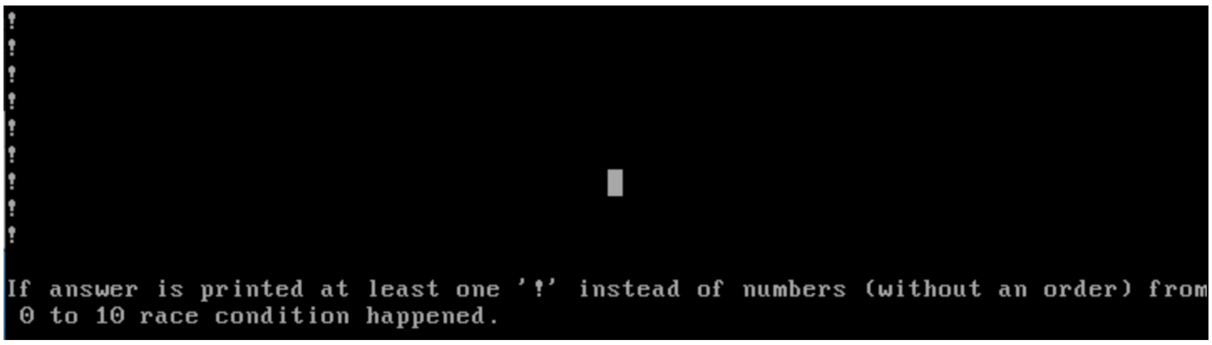
Join Thread:

It waits the caller thread until given thread id is finished.

Yield Thread:

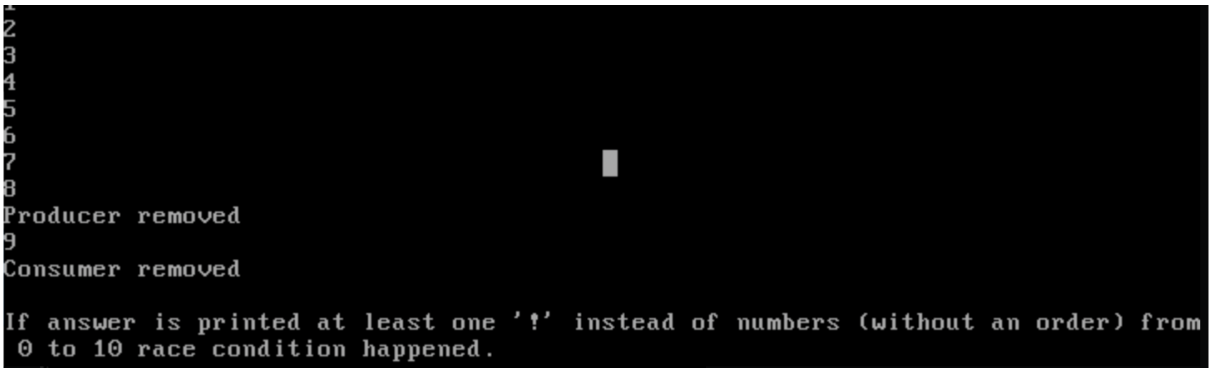
It adds yield amount to the field in the thread and each time schedule is called it decrease. With this way given thread gives advantage to other threads.

Part 2 - Producer Consumer, Peterson Algorithm, Race Conditions



```
!
!
!
!
!
!
!
!

If answer is printed at least one '!' instead of numbers (without an order) from
 0 to 10 race condition happened.
```



```
2
3
4
5
6
7
8
Producer removed
9
Consumer removed

If answer is printed at least one '!' instead of numbers (without an order) from
 0 to 10 race condition happened.
```