CS 481: Operating Systems
Lab 3: C and pthreads

Jacob Hurst

April 29, 2019

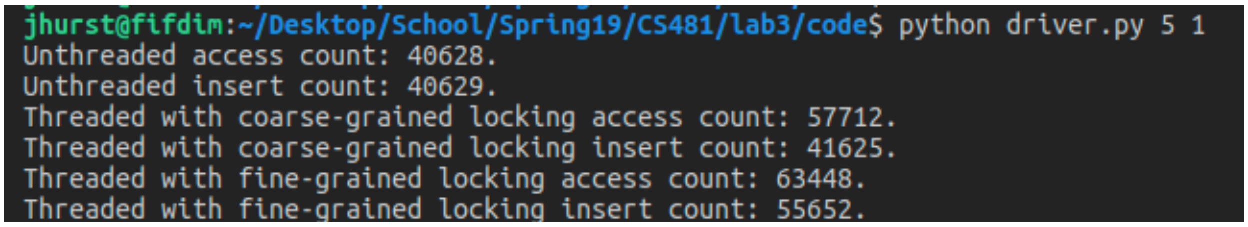
1 Hypothesis

Using C's pthread library can provide performance improvements to linked list implementations. The performance improvements are proportional to the granularity of your locking method, with fine-grained locking methods offering the most significant performance improvements.

2 Methods

Python program 'driver.py' drives 3 smaller C programs, 'unthreaded.c', 'coarse-grained.c', and 'fine-grained.c'.

The python driver works by first cleaning the directory and then compiling, running, and tracing a total of n runs of m seconds of the C programs, where n and m are provided via command-line. Each of the C programs implement the same linked list with minor differences in locking granularity. The programs are assessed on the number of accesses and inserts they are able to perform on their variant of the linked list.

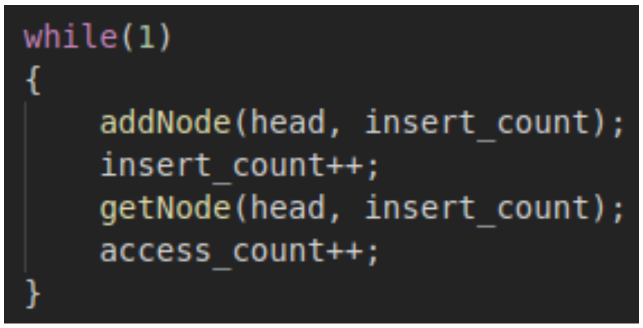


```
jhurst@fifdim:~/Desktop/School/Spring19/CS481/lab3/code$ python driver.py 5 1
Unthreaded access count: 40628.
Unthreaded insert count: 40629.
Threaded with coarse-grained locking access count: 57712.
Threaded with coarse-grained locking insert count: 41625.
Threaded with fine-grained locking access count: 63448.
Threaded with fine-grained locking insert count: 55652.
```

Figure 1: A single sample run of 5s.

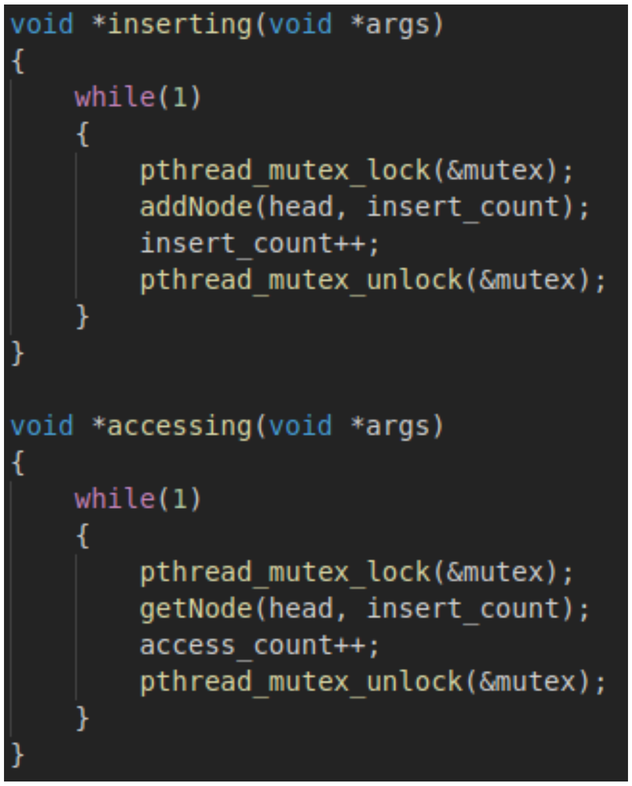
2.1 Factors

1. Lock granularity: In unthreaded, no lock is necessary, the program simply alternates between inserts and accesses for it's duration. In coarse-grained, lock is engaged prior to any function call and released after returning from the function call. In fine-grained, lock is engaged only in critical insert sections, as needed and released immediately after critical section. (See code samples below or full code in 'code' directory of zipped experiment.)
2. Sample Count: The number of runs completed for the 3 programs, 100 for this experiment.
3. Sample Duration: The length of time each program is allotted to run, 5s for this experiment.



```c
while(1)
{
    addNode(head, insert_count);
    insert_count++;
    getNode(head, insert_count);
    access_count++;
}
```

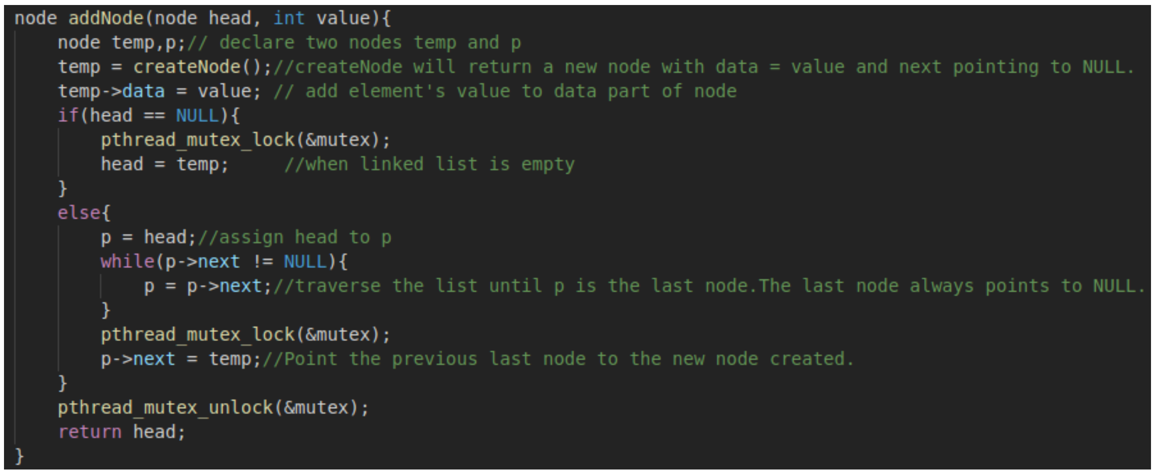
Figure 2: Code sample detailing unthreaded task.



```c
void *inserting(void *args)
{
    while(1)
    {
        pthread_mutex_lock(&mutex);
        addNode(head, insert_count);
        insert_count++;
        pthread_mutex_unlock(&mutex);
    }
}

void *accessing(void *args)
{
    while(1)
    {
        pthread_mutex_lock(&mutex);
        getNode(head, insert_count);
        access_count++;
        pthread_mutex_unlock(&mutex);
    }
}
```

Figure 3: Code sample detailing coarse-grained task



```c
node addNode(node head, int value){
    node temp,p;// declare two nodes temp and p
    temp = createNode();//createNode will return a new node with data = value and next pointing to NULL.
    temp->data = value; // add element's value to data part of node
    if(head == NULL){
        pthread_mutex_lock(&mutex);
        head = temp;     //when linked list is empty
    }
    else{
        p = head;//assign head to p
        while(p->next != NULL){
            p = p->next;//traverse the list until p is the last node.The last node always points to NULL.
        }
        pthread_mutex_lock(&mutex);
        p->next = temp;//Point the previous last node to the new node created.
    }
    pthread_mutex_unlock(&mutex);
    return head;
}
```

Figure 4: Code sample detailing fine-grained task, (no locks applied for accessing).

2.2 Response

Per Program, operations on linked list were enumerated and averaged.

1. Average Access Count: Over 100, 5s samples, how many accesses were achieved on average?
2. Average Insert Count: Over 100, 5s samples, how many inserts were achieved on average?

2.3 Treatments

Each treatment ran each C program for 5s. In unthreaded, the program alternates between inserting and accessing for the allotted time. In coarse-grained, the program has a thread for inserting and a thread for accessing but locks are placed over all linked list function calls. In fine-grained, the program has a thread for inserting and a thread for accessing and the locks are placed only in critical sections.

2.4 Repetitions

A total of 100 repetitions of the above treatment were ran in order to ensure consistency in results

3 Results

Below are figures generated from averaging the 100 samples on each process. For more information, see 'figures' directory of zipped experiment.

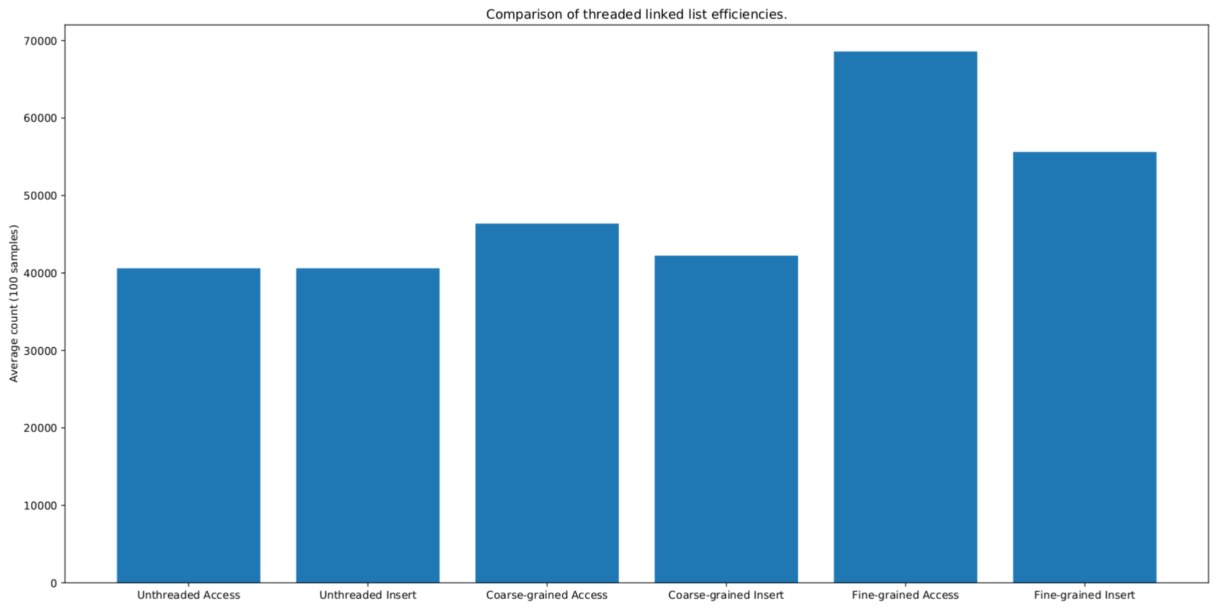


Figure 5: From left to right labels; unthreaded access, unthreaded insert, coarse-grained access, coarse-grained insert, fine-grained-access, fine-grained insert.

As shown in the figure above, the unthreaded program achieved 40,000 accesses on the linked list and 40,000 inserts. The threaded program with coarse-grained locking achieved 46,000 accesses and 43,000 inserts on the linked list. The threaded program with fine-grained locking achieved 68,000 accesses and 55,000 inserts on the linked list.

Performance improvements to the variant linked list implementations were proportional to the granularity of the threaded implementations locking method, with fine-grained locking methods offering the most significant performance improvements.