How to implement thread safe queues

Asked 12 years, 7 months ago Modified 12 years, 6 months ago Viewed 35k times

10

I have used multithreading library before in Python, but this is the first time I am trying threading in C. I want to create pool of workers. In turn, these workers supposed to push to or pop from queue.Following code is not quite there yet, but is what I have done so far:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define NUMTHREADS 20 /* number of threads to create */

typedef struct node node;
typedef struct queue queue;

struct node {
    char *name;
    node *next;
};

struct queue {
    node *head;
    node *tail;
};

/* pop: remove and return first name from a queue */
char *pop(queue *q)
{
    if (q->head == NULL)
        return NULL;
    char *name = q->head->name;
    node *tmp = q->head;
    q->head = q->head->next;
    free(tmp);
    return name;
}

/* push: add name to the end of the queue */
int push(queue *q, char *name)
{
    node *new = malloc(sizeof(node));
    if (new == NULL)
        return -1;
    new->name = name;
    new->next = NULL;
    if (q->tail != NULL)
        q->tail->next = new;

    q->tail = new;
    if (q->head == NULL) /* first value */
        q->head = new;
    return 0;
}
```

```
/* printname: get a name from the queue, and print it. */
void *printname(void *sharedQ)
{
    queue *q = (queue *) sharedQ;
    char *name = pop(q);
    if (name == NULL)
        pthread_exit(NULL);
    printf("%s\n",name);
    pthread_exit(NULL);
}

int main()
{
    size_t i;
    int rc;
    pthread_t threads[NUMTHREADS];
    char *names[] = {
        "yasar",
        "arabaci",
        "osman",
        "ahmet",
        "mehmet",
        "zeliha"
    };

    queue *q = malloc(sizeof(queue));
    q->head = NULL;
    q->tail = NULL;

    /* number of elements in the array */
    size_t numelems = sizeof(names) / sizeof(char *);

    for (i = 0; i < numelems; i++) /* push each name */
        push(q, names[i]);

    for (i = 0; i < NUMTHREADS; i++) { /* fire up threads */
        rc = pthread_create(&threads[i], NULL, printname,
                (void *)q);
        if (rc) {
            printf("Error, return code from pthread is %d\n", rc);
            exit(-1);
        }
    }

    pthread_exit(NULL);
}
```

I tried above code, and it always printed each name exactly once. It didn't skip any names, or printed same name twice. On the other hand, I am not sure how thread safe this queue implementation is. So my question is, Is this a threadsafe queue? If not, why not? And how to make it thread safe?

c multithreading pthreads queue posix

Share

Improve this question

Follow

edited Jun 5, 2012 at 21:36

asked May 23, 2012 at 13:37

yasar
13.7k ● 30 ● 97 ● 170

The structs don't need typedefs; they have a type already. – user82238 May 23, 2012 at 13:54

2 Answers

Sorted by: Highest score (default)

8

The code is not thread safe.

The push and pop functions are not thread safe. In the code, the push is only being executed by a single thread, so it doesn't matter, but the pops are being executed by multiple threads.

```
1. char *name = q->head->name;
2. node *tmp = q->head;
3. q->head = q->head->next;
4. free(tmp);
```

Imagine thread A executes up to and including line 2. Thread B then executes up to and including line 4. Thread A resumes execution. It finds that q->head has already been free()ed.

Now, this so far discusses logical issues.

However, there are physical issues to consider.

Imagine we had a locking mechanism whereby threads could synchronize their behaviour, such that only one thread at a time could execute the code in the lines 1 to 4, e.g. a mutex, which is an object only one thread can 'hold' at a time, and where attempting to get the mutex blocks the thread until the holding thread releases.

```
0. get mutex
1. char *name = q->head->name;
2. node *tmp = q->head;
3. q->head = q->head->next;
4. free(tmp);
5. release mutex
```

We would still have a problem, in that the writes performed by any given CPU core (not thread) are visible immediately only to threads on that core; not to threads on other cores.

It is not enough merely to sychronize execution; at the same time, we must also ensure the writes performed by a core become visible to other cores.

(Un)fortunately, all modern sychronization methods also perform this write flushing (e.g. when you get a mutex, you also flush all writes to memory). I say unfortunately, because you don't -always- need this behaviour and it is harmful to performance.

Share
Improve this answer
Follow

edited May 23, 2012 at 15:06

answered May 23, 2012 at 13:58
user82238

3

It is not thread-safe since multiple threads may modify the pointers in the linked list at the same time, potentially corrupting it.

Here you have an answer for a very similar question: Multiple-writer thread-safe queue in C

There you can see how to make the queue thread-safe.

Share
Improve this answer
Follow

edited May 23, 2017 at 11:53
Community Bot
1 ● 1

answered May 23, 2012 at 14:00
betabandido
19.7k ● 11 ● 64 ● 79