Assignment 4

CS330: Operating Systems

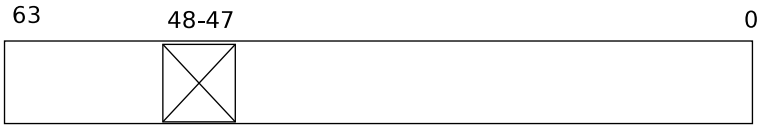
Task-1: Fix me! (20 Marks)

For this task, please refer to the code template and helper routines provided in part-1 folder and go through the `README` file carefully. In this task, a file is passed as input which is chucked into blocks of 64bytes and a hash is calculated for every block. An output array is used to store the calculated hash values of corresponding blocks. A serial implementation is provided in the provided code which does not work correctly when executed with more than one threads. Your task is to modify the `hashit` function in `thread_func.c` to make it work correctly with more than one threads. Note that, the `calculate_and_store_hash` function should be invoked without holding any locks. To test your program, you can use any existing file on your system. (see README)

Task-2: A miser's read-write lock! (35 Marks)

A read-write lock allows multiple readers but one writer to enter the critical section. Refer to the slide#9 - #15 in `https://www.cse.iitk.ac.in/users/deba/cs330/producer-consumer.pdf` for details and an example implementation using semaphores. For this task, please refer to the code template and helper routines provided in part-2 folder and read the `README` file carefully.

The proposed RWLOCK uses a single 64-bit long integer. One of the ways to implement a read-write lock using a single variable is to use the single variable in an innovative manner as explained below.



64-bit long integer based R/W lock implementation

Value at the bit position 47 plays a crucial role in determining the state of the lock as explained below,

```
Value   0x1000000000000     -->     unlocked
        0x0000000000000     -->     locked for writing
        0xFFFFFFFFFFFF      -->     one reader
```

```
0xFFFFFFFFFFFE      -->       two readers
                              .....
```

In the template code and helper files, you are provided with the implementation for `atomic_add` (see common.h). The `val` argument can be negative. This function returns 0, 1 and -1 if the result of `atomic_add` is zero, positive and negative, respectively. Implement the template functions in `rwlock.c`. A sample main program is also provided to test basic correctness (in main.c). However, you should use your own test programs to validate the lock implementation.

Task-3: Parallel hashing (45 marks)

For this task, please refer to the code template and helper routines provided in part-2 folder and read the `README` file carefully. Implement a parallel hash implementation (linear probing based). A serial implementation with the template and infrastructures are provided. Refer to the code and comments for details of the implementation. Note that, your implementation should not perform a global serialization of hash operation and try to achieve as much parallelism as possible. Implementations equivalent to serial implementations will be awarded zero marks.

Submission guidelines

- The assignment is to be done individually. You have to submit only three files (thread_func.c (part-1), rwlock.c (part-2) and parallel_hash.c (part-3)). Put these three files in a directory named with your roll number. Create a zip archive of the directory and upload in canvas.
- *Don't modify any other files.* We will not consider any file other than the above mentioned files for evaluation.
- *You should remove all printf debug statements from submission files.* We will taking diff of your output and expected output for evaluation.

In-case of any issues you should reach out to us at the earliest. All the best!