Lab 9 - Locks and Conditions

Nov 6, 2023

CSCI 4061
Introduction to Operating Systems

UNIVERSITY OF MINNESOTA
Driven to Discover℠

Overview

1. PA3 Discussion
2. Producer Consumer Problem (Activity)
3. Locking APIs
4. Condition Variable APIs
5. Expected Output
6. Deliverables

PA3 - Parallel Handling of Image Rotation

Idea: Given a an input folder with a set of png images. Rotate the images by either 180 or 270 degrees and store the rotated images in an output folder.

Objectives:

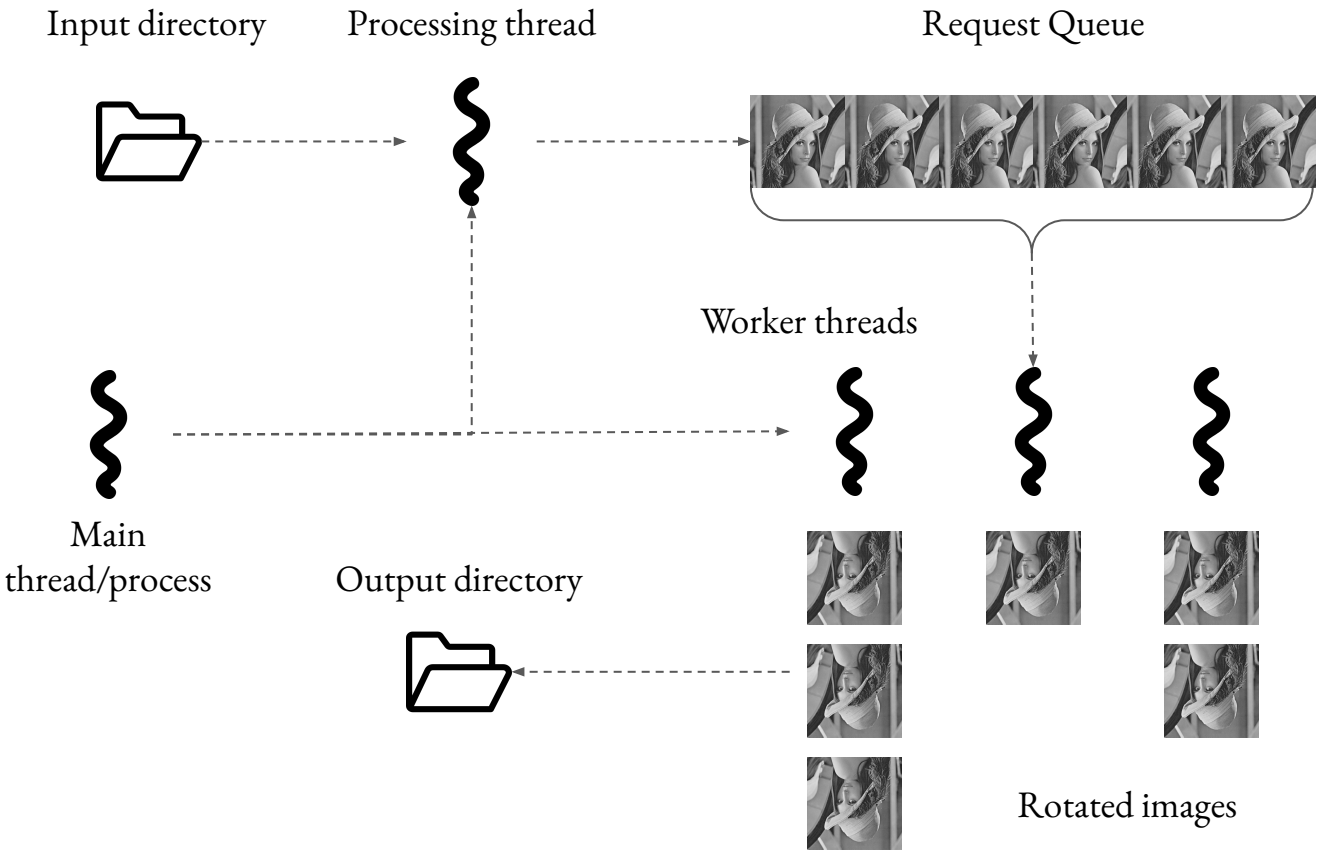
1. The processing thread will traverse the given directory (argv[1]) to populate a request queue with image filepath and rotation angle (argv[4]) (use structure).
2. The worker threads (argv[3])will pick requests from the request queue, carry out the rotation and write the rotated image to a output folder (argv[2]) with the same name as original image.

PA3 - Parallel Handling of Image Rotation

To think about:

1. The processing thread and the worker threads are all spawned at the same time
2. The worker threads will be monitoring the request queue and as soon as a request is added by processing thread, one of the worker threads will take it
3. A worker thread should only be able to pick a request only after the processing thread has stored one.
4. Once the processing thread has finished uploading to the request queue, it will let all the worker threads know that there will be no more request added to the queue
5. The worker threads will acknowledge the processing thread once all the images in the queue are processed
6. The processing thread will verifying if the total number of requests uploaded to the queue is same as the total number of requests processed by the workers, send a termination signal to all workers and exit

PA3 Overview



APIs required for PA3

1. pthread_create
2. pthread_join
3. pthread_mutex_lock
4. pthread_mutex_unlock
5. pthread_cond_signal
6. pthread_cond_wait
7. pthread_cond_broadcast
8. pthread_exit

You may need all of them.

Intermediate submission

1. The processing thread and worker threads are spawned simultaneously
2. The processing thread will traverse input directory and populate the request queue
3. The worker threads will print threadID and exit

Deadline: Nov 8, 11:59 pm, 2023

Final Submission

Complete Project Objective

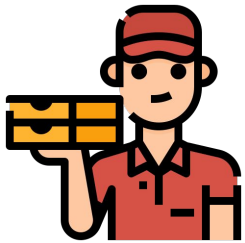
Check the Rubric, testcases have less weightage this time

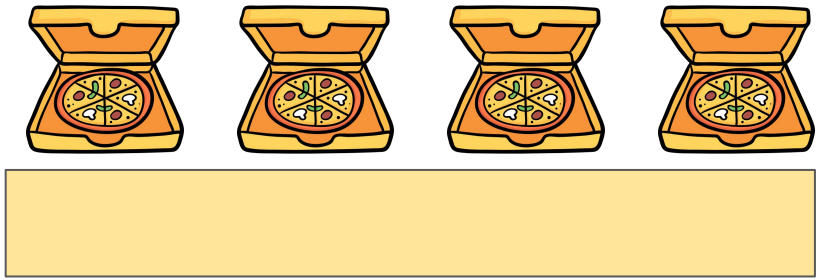
Refer to Pinned FAQ in Piazza

Deadline: Nov 15, 11:59 pm, 2023

Activity: Producer-Consumer Problem

Dominos Pizza shop (Producer) has a stand on which they place ordered Pizzas for customers to be picked by Delivery boy (Consumer).





Key Points

1. The producer can place a maximum number of pizzas on the stand
2. When the producer place a pizza, it will signal the consumer about its availability
3. A consumer can pick a pizza from the stand only if there is at least one available, otherwise it will wait for a signal from producer
4. Once all the orders are complete, the produce will signal the consumer to exit and it will also exit.
5. There is only one producer and one consumer.
6. In this activity, the producer will create 20 pizzas. The stand can only hold 4 pizzas at a time. The consumer should pick all the 20 pizzas.

Mutex (Mutual Exclusion) locks - samples/p1.c

Used for locking critical sections (shared resource access) to ensure only one thread can access the section after acquiring the lock

```
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // mutex variable

pthread_mutex_lock(&mutex); // enter critical section
```

Critical Section

```
pthread_mutex_unlock(&mutex); // leave critical section
```

Locking APIs

```
pthread_mutex_t // mutex variable type

PTHREAD_MUTEX_INITIALIZER // default mutex behavior, no error checking

int pthread_mutex_lock(pthread_mutex_t *mutex); // locking

int pthread_mutex_unlock(pthread_mutex_t *mutex); // unlocking
```

Condition Variables - samples/p2.c

Say T1 and T2 are two threads. T2 is waiting for T1 to set shared variable x to 1. It keeps on checking if x is 1 in a while loop.

```
// T1

...

...

x = 1
```

```
// T2

while(1) {

        if(x == 1) break

}
```

CPU cycles wasted

Condition Variables

Instead of unnecessarily consuming CPU cycles, the thread can go to sleep and wake up when T1 sets x to 1. This is achieved using condition variables.

```
// T1

...

...

x = 1

signal(T2)
```

```
// T2

while(1) {

        if(x == 1) break

        wait(T1)

}
```

T2 sleeps and waits for signal from T1

Condition Variables

```
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // mutex variable

pthread_cond_t cond = PTHREAD_COND_INITIALIZER; // condition variable

 T1(){                                          T2(){

 pthread_mutex_lock(&mutex);                    pthread_mutex_lock(&mutex);

 x = 1;                                         while(x != 1)

 pthread_cond_signal(&cond);                          pthread_cond_wait(&cond, &mutex);

 pthread_mutex_unlock(&mutex);                  pthread_mutex_unlock(&mutex);

 }                                              }
```

Condition variables

pthread_cond_wait(&cond, &mutex)

Release the mutex lock and waits for the signal from another thread on cond

On receiving signal, the mutex is locked automatically

pthread_cond_signal(&cond)

Signals any one thread waiting on cond (consider it as a queue)

pthread_cond_broadcast(&cond)

Signals all threads waiting on cond

Condition Variable APIs

```
pthread_cond_t // condition variable type

PTHREAD_COND_INITIALIZER // default cond behavior, no error checking

int pthread_cond_signal(pthread_cond_t *cond); // signal cond variable

int pthread_cond_wait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex);

        // wait on cond variable

int pthread_cond_broadcast(pthread_cond_t *cond); // broadcast signal to multiple threads
```

APIs for Activity

pthread_create: Create producer and consumer threads

pthread_join: Join the producer and consumer threads after execution

pthread_mutex_lock: Lock the stand when a producer adds pizza to it and when a consumer picks pizza from it

pthread_mutex_unlock: Unlock the stand after the pizza is added or picked

pthread_cond_signal: Producer will signal consumer when its add a pizza and consumer will signal producer if it consumed a pizza. It will be also used to signal consumer to exit by producer.

pthread_cond_wait: Producer will wait for signal from consumer if the stand is full before adding a pizza and the consumer will wait for signal from producer if the stand is empty

Expected output

```
$ make
Producer added Pizza 1 to stand
Consumer picked Pizza 1 from stand
Producer added Pizza 2 to stand
Consumer picked Pizza 2 from stand
.....
Producer added Pizza 20 to stand
Consumer picked Pizza 20 from stand
Producer completed all orders, exiting...
Consumer completed all orders, exiting...
```

Order of output may change

Deliverables

Most of the code is provided to you, fill the locations with “?”. Submit the deliverables to Gradescope as a zip by Nov 7, 11:59 pm.

- pizza.c
- output.txt - copy output on the screen