ADVANCE OPERATING SYSTEM

Name: M. Hassan Sattar

Roll No: 22F-3773

Submitted to: Dr. Bilal Khan

National University of Computer and Emerging Sciences

1

What a 'C' Program that finds minimum in an array of size 1000 of integers. The elements of array shall be randomly generated in the range of 1 to 100,000. You shall repeat this exercise for different number of threads, e.g., 2, 4 and 8. Your answer shall also include the full code as well as the screenshot of the terminal output. Your program shall use the following for the solution:

i. Pthread library

```
#include <iostream>
#include <pthread.h>
#include <stdlib.h>

using namespace std;

const int size = 1000;
const int total_threads = 4;

int array_values[size];
int min_num[total_threads];
int pid = 0;

void* min(void*) {
    int id = pid;
    cout << "process id #" << id << " started..." << endl;
    pid++;
    int min = 100000;
    int block = size/total_threads;
    int sid = id * block;
    int eid = sid + block;
    cout << "block size " << block << endl;
    for(int i = sid; i < eid; i++) {
        if( min > array_values[i] )
            min = array_values[i];
    }
    min_num[id] = min;  //  assign minimum value into min_num[id]
    cout << "process id #" << id << " ended..." << endl;
    return NULL;
```

```
}

int main() {
    cout << "starting process " << endl;
    pthread_t thread[total_threads];
    srand(time(NULL));
    for(int i = 0; i < size; i++ ) {
        array_values[i] = (rand()%100000)+1;
        cout << array_values[i] << ", ";
    }
    cout << endl;

    for(int i = 0; i < total_threads; i++ ) {
        pthread_create(&thread[i], NULL, min, NULL);
    }

    for(int i = 0; i < total_threads; i++ ) {
        pthread_join(thread[i], NULL);
    }

    int min_value = 100000;
    for(int i = 0; i < total_threads; i++ ) {
        if( min_value > min_num[i] )
            min_value = min_num[i];
    }

    cout << "minimum value is " << min_value << endl;

    return 0;
}
```

main.cpp

```cpp
#include <iostream>
#include <pthread.h>
#include <stdlib.h>

using namespace std;

const int size = 1000;
const int total_threads = 2;

int array_values[size];
int min_num[total_threads];
int pid = 0;

void* min(void*) {
    int id = pid;
    cout << "process id #" << id << " started..." << endl;
    pid++;
    int min = 100000;
```

input

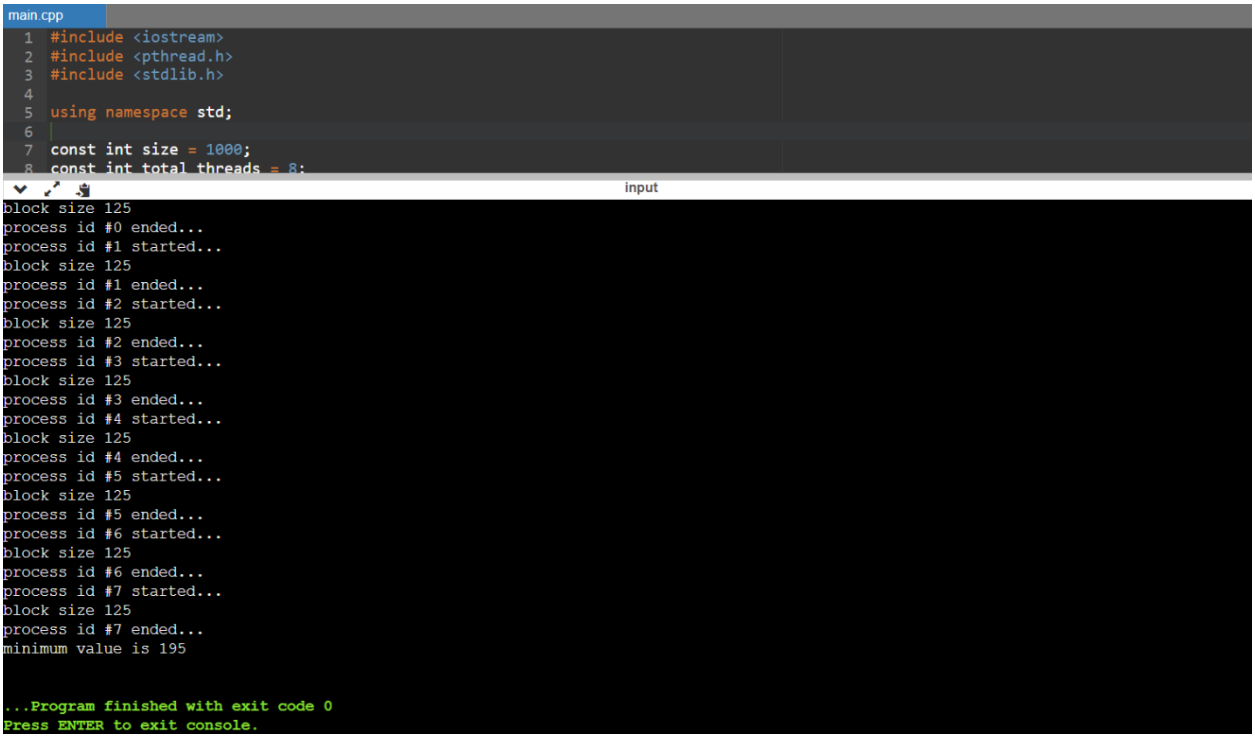
```
 55120, 9112, 64636, 20283, 77931, 14850, 43399, 75468, 67803, 9184, 27300, 53489, 17060, 25380, 62999, 83808, 55943, 6169, 26361, 7417
1, 77987, 22812, 93830, 68702, 26622, 34319, 64927, 26638, 97351, 60695, 51896, 68822, 69807, 16532, 89104, 64089, 47733, 48854, 39556,
 15535, 58037, 66855, 85375, 91448, 8586, 48374, 91607, 80881, 54542, 17967, 71403, 48880, 40779, 65232, 17582, 83752, 99550, 82508, 26
741, 13252, 59554, 78637, 82073, 29360, 11520, 71176, 93449, 59252, 36381, 49356, 74786, 94418, 32563, 60160, 85865, 41148, 24885, 7747
2, 22028, 79426, 11790, 93430, 44658, 68920, 75014, 62239, 69023, 74563, 61098, 95764, 87815, 20651, 90752, 69887, 66363, 18623, 57415,
 76163, 77874, 93795, 25518, 69011, 4564, 58080, 45522, 90429, 15580, 70407, 84252, 53959, 66184, 12393, 63741, 10841, 81313, 38754, 73
079,
process id #0 started...
block size 500
process id #0 ended...
process id #1 started...
block size 500
process id #1 ended...
minimum value is 6


...Program finished with exit code 0
Press ENTER to exit console.
```

main.cpp

```cpp
#include <iostream>
#include <pthread.h>
#include <stdlib.h>

using namespace std;

const int size = 1000;
const int total_threads = 4;

int array_values[size];
int min_num[total_threads];
int pid = 0;

void* min(void*) {
    int id = pid;
    cout << "process id #" << id << " started..." << endl;
    pid++;
    int min = 100000;
```

input

```
 71280, 7612, 18761, 88449, 98544, 49528, 79648, 98137, 15790, 61387, 3118, 63320, 65052, 5105, 79884, 97164, 28593, 57305, 11029,
process id #0 started...
block size 250
process id #0 ended...
process id #1 started...
block size 250
process id #1 ended...
process id #2 started...
block size 250
process id #2 ended...
process id #3 started...
block size 250
process id #3 ended...
minimum value is 21


...Program finished with exit code 0
Press ENTER to exit console.
```



ii. openMP

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

using namespace std;

int main() {
    const int size = 1000;
    int array_values[size];

    srand(time(0));
    for(int i = 0; i < size; i++) {
        array_values[i] = (rand() % 100000) + 1;
    }

    int min_value = 100000;

 cout << "number of threads running in openmp library " << omp_get_max_threads();
    cout << endl;
```

```
#pragma omp parallel for
    for(int i = 0; i < size; i++) {
        if( min_value > array_values[i] )   //  Check if the min_value is greater
        {
            #pragma omp critical            //  Enter critical section
            if(min_value > array_values[i]) //  Compare again if the min_value is
greater
            {
                min_value = array_values[i];
            }
        }
    }

    cout << "minimum value using openmp library " << min_value;
    cout << endl;

    return 0;
}
```

```
[root@localhost ~]# export OMP_NUM_THREADS=2
[root@localhost ~]# g++ q1.cpp -o q1 -fopenmp
[root@localhost ~]# ./q1
number of threads running in openmp library 2
minimum value using openmp library 275
[root@localhost ~]#
```

```
[root@localhost ~]# export OMP_NUM_THREADS=4
[root@localhost ~]# g++ q1.cpp -o q1 -fopenmp
[root@localhost ~]# ./q1
number of threads running in openmp library 4
minimum value using openmp library 33
[root@localhost ~]#
```

```
[root@localhost ~]# export OMP_NUM_THREADS=8
[root@localhost ~]# g++ q1.cpp -o q1 -fopenmp
[root@localhost ~]# ./q1
number of threads running in openmp library 8
minimum value using openmp library 70
[root@localhost ~]#
```

2

Show how a binary semaphore can be used to implement mutual exclusion among 3 processes?

We are aware that a multitasking operating system's semaphore variable may be utilized to control access to a common resource using a few different methods. According to the thought experiment, if there are three approaches, each of which typically shares a semaphore, system allocation is prepared using the following algorithm..

```
do {
wait(mutex);
signal(mutex);
} while(true);

Wait(Semaphore s)
{  s = s - 1;
   if (s < 0) {
  block(p);
   }
}

Signal(Semaphore s)
{
```

```
  s = s + 1;
  if (s >= 0) {
    wakeup(p);
  }
}
```

Suppose we have 3 processes P0, P1, and P2. Using Binary Semaphore, only 1 process can enter critical section at a time. Let's say P0 enters the critical section. Others processes P1, and P2 will wait for signal () call from the P0 process. Once P0 process executes completely it will call signal () and let other process enter critical section. The next process from the queue will get high priority. Let's say the P1 is the process in queue. Now P1 enters in critical section. And let P0, and P2 wait for signal () call from P1. After P1 executes completely. It will call signal () and P2 in the waiting queue will get the priority. Now P2 enter critical section P1, and P2 waits for the execution of the process P2. Once P2 executes, the same thing happen we discuss from the start. This is the example of mutual exclusion using Binary semaphore.

3

The implementation of the basic mutex locks causes busy waiting. Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue until the lock became available.

Busy waiting affects the mutex lock implementation. Describe the modifications that would be required to allow a process waiting to acquire a mutex lock to be blocked and placed into a waiting queue until the lock became available.

Spinlocks may be used in certain circumstances to prevent operating systems from becoming less effective. a) A spinlock is a lock that causes process threads to enter the wait condition as they seek for the availability of a desired lock. b) Spinlocks are used for brief periods of time; therefore, they could not have a significant influence on the computer's resources. c) Additionally, the addition of spinlocks could enable threads to actively partition processors based on their requirements. where other threads can operate simultaneously over different

processors without being interrupted while the thread waiting for a spinlock may run on one processor.

4

Discuss the tradeoff between fairness and throughput of operations in the readers–writers' problem. Propose a method for solving the readers–writers' problem without causing starvation.

The readers-writers problem is explained throughout by preferring a few readers rather than allowing a single writer to have exclusive access to the shared values. Again, favoring readers may cause writers to go hungry. By keeping track of the timestamps associated with waiting techniques, the hunger in the readers'/writers' difficulty may be avoided. after a creator has finished their creation. The approach that has been anticipating for the greatest amount of time may be awakened. While another reader is accessing the database, a reader comes and observes this. If no writers are available, it would then most effectively go onto the crucial section. These boundaries would ensure equality.