

Operating Systems – Exercise 2

Processes, Threads, Synchronization

General Guidelines:

Submission deadline is **Wednesday, April 15th, 23:55 (Moodle Server Time)**.

Pack your files in a ZIP file named `Ex2-YourID.zip`, for example: `Ex2-012345678.zip`

Submit the ZIP file in the submission page in the course website

- No late submission will be accepted!
- You should work on your exercise by yourself. Misconducts will be punished harshly.
- Please give concise answers, but make sure to explain any answer.
- Document your code. Place your name and ID at the top of every source file, as well as in the printed theoretical part.
- Do not submit handwritten answers.

Part 1 - Threads (30 points)

A. (20 points)

Write a Multithreaded Program:

In this part you will implement a multithreaded matrix multiplication tool. The program will be able to split the work to multiple threads, to enhance performance.

This program will be able to handle square matrices only.

Write a class named `MatrixMultThread` implements [Runnable](#), with the following methods:

This class should include the following public static method:

```
public static float[][] mult(float[][] a, float[][] b, int threadCount)
```

Parameters:

a – Left hand matrix

b – Right hand matrix

threadCount – Number of threads

Return value:

The result of $a*b$

Also, you should implement a main method that generates two 1024x1024 matrices filled with random values. Then, it should multiply them using the `mult` method described above using at least two threads. Also, it should measure the time in milliseconds it took to perform the multiplication and print this time to the screen.

- **Both matrices are of the same size, and square**
- Use `System.currentTimeMillis()` to retrieve current system time
- See `java.lang.Runnable` and `java.lang.Thread` to understand more on threads in Java
- It is your responsibility to manage the workload between threads
- **You may have an upper limit on the number of threads that depends on matrix size. This number should be explicitly stated in the documentation, in a noticeable location.**
- Example for a good implementation: divide the job by rows – each thread handles *number_of_rows* divided by *number_of_threads* rows. Maximal number of threads is *number_of_rows*. Of course, better implementations are possible.
- Only use static variables for constants, not as a synchronization or data exchange tool.

B. (5 points)

Create a line graph of several runs of your implementation of MatrixMultThread on 1024x1024 matrices. Run the program with 1, 2, 3, 4, 5, 6, 7, 8 threads, for each thread count run it 5 times and then use the median result for the graph. The graph X axis will have the number of threads, the Y axis will have the average time it took to execute.

What number of threads gave the best times? _____

How many cores do you have in your computer? _____

(Find it via Task Manager [Windows] / Top [Linux/Mac] / System Monitor [Mac])

Try to explain the differences / graph trend you got in the experiment: _____

C. (5 points)

Note: this part is theoretical. No code is required to be written or submitted!

Let's assume you implemented the multiplication as suggested, divide the job by rows – each thread handles *number_of_rows* divided by *number_of_threads* rows. Maximal number of threads is *number_of_rows*.

Now we will change the implementation to multiply each cell of the matrix in its own thread.

How will that affect the performance?

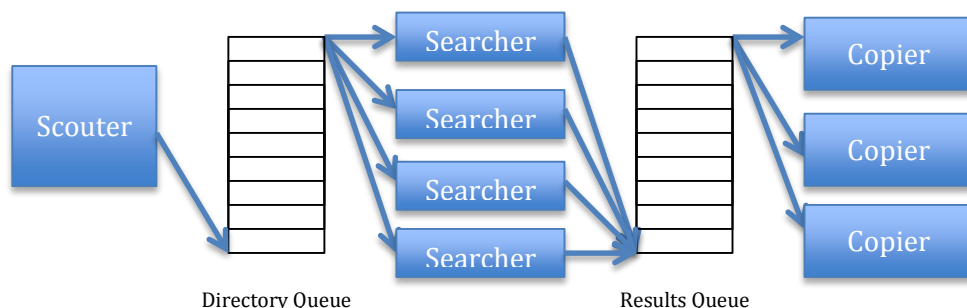
(The hardware does not change, you run the same tests as before)

It will run faster / slower because _____

Part 2 - Synchronization (40 points)

In this part we will create a multithreaded search utility. The utility will allow searching for files that contain some given file extension, under some given root directory. Files that have this extension will be copied to some specified directory.

Our application consists of two queues and three groups of threads:



The attached JavaDoc contains detailed explanation for each class in the application. Please read it carefully and follow the APIs as defined in it.

(To open the attached JavaDoc open the file [index.html](#) inside the directory [doc](#))

A. Write the class `SynchronizedQueue`.

This class should allow multithreaded enqueue/dequeue operations.

The basis for this class is already supplied with this exercise. You have to complete the empty methods according to the documented API and also follow **TODO** comments.

For synchronization you may either use monitors or semaphores, as learned in recitation.

This class uses Java generics. If you are not familiar with this concept you may read the first few pages of this tutorial: <http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>

- B.** Write the class `Scouter` implements `Runnable`.
This class is responsible to list all directories that exist under the given root directory. It enqueues all directories into the directory queue.
There is always only one scouter thread in the system.
- C.** Write the class `Searcher` implements `Runnable`.
This class reads a directory from the directory queue and lists all files in this directory. Then, it checks each file to see if the extension matches the one given. Files that has the right extension are enqueued to the results queue (to be copied).
- D.** Write the class `Copier` implements `Runnable`.
This class reads a file from the results queue (the queue of files that has the right extension), and copies it into the specified destination directory.
- E.** Write the class `DiskSearcher`.
This is the main class of the application. This class contains a `main` method that starts the search process according to the given command lines.
Usage of the main method from command line goes as follows:

```
> java DiskSearcher <extension> <root directory> <destination directory>
    <# of searchers> <# of copiers>
```


For example:

```
> java DiskSearcher txt C:\OS_Exercises C:\temp 10 5
```

This will run the search application to look for files with the extension 'txt', in the directory C:\OS_Exercises and all of its subdirectories. Any matched file will be copied to C:\temp. The application will use 10 searcher threads and 5 copier threads.
Specifically, it should:
 - Start a single scouter thread
 - Start a group of searcher threads (number of searchers as specified in arguments)
 - Start a group of copier threads (number of copiers as specified in arguments)
 - Wait for scouter to finish
 - Wait for searcher and copier threads to finish

Specific guidelines:

1. Read the attached JavaDoc. It contains a lot of information and tips.
You must follow the public APIs as defined in the attached JavaDoc!
2. Use the attached code as a basis for your exercise. Do not change already-written code. Just add your code.
3. To list files or directories under a given directory, use the `File` class and its methods `listFiles()` and `listfiles(FilenameFilter)`.
Note that if for some reason these methods fail, they return null. You may ignore such failures and skip them (they usually occur because insufficient privileges).
4. If you have a problem reading the content of a file, skip it.

Part 3 - Theoretical (30 points)**A.** (10 Points)

Two threads execute the following code:

```
for i = 6 to 22
    x += 2
```

(x is a shared variable, initially set to zero. i is a local (unshared) variable)

What is/are the possible value(s) for x at the end? Prove formally!

B. (10 points)

Suppose two threads execute the following C code concurrently, accessing shared variables a, b, and c:

Initialization:

```
int a = 5;
int b = 0;
int c = 0;
```

Thread 1:

```
1. if (a < 0) {
2.     c = b - a;
3. } else {
4.     c = b + a;
5. }
```

Thread 2:

```
1. b = 10;
2. a = -4;
```

* the numbers to the left are the line numbers

What are the possible values for c after both threads complete? Explain!

You can assume that reads and writes of the variables are atomic, and that the order of statements within each thread is preserved in the code generated by the C compiler.

C. (10 points)

What would be the output of the following pseudo C/UNIX code? Explain!

Notes:

Be careful not to explain what the code does! Do not tell a story about x, that it will be multiplied by 2 and then something else will happen. Give a concise answer about what the output to the screen will be.

i.

```
int x; // x is a global variable
int main() {
    int i;
    x = 1;
    for (i = 0; i < 75; i++) {
        x = i;
        fork();
        printf("X=%d\n", x);
    }
    return 0;
}
```

ii.

```
int k; // global variable
void main() {
    k = 1;
    createThread(, , printSomething, ...);
}

void *printSomething() {
    if (k < 7) {
        atomic_add(&k, 2); // atomically adds 2 to k
        createThread(, , printSomething, ...);
        printf("%d", k);
    }
}
```
