CS 314 - OS Laboratory Assignment 6

Cebajel Tanan (210010055)
Tanishq Trivedi (210010056)

March 1, 2024

Contents

1 Transformations

We have used the following two transformations:

1.1 Grayscale

This transformation converts a colour RGB image to a grayscale image. This simple transform can be achieved by taking the weighted average of the red, green and blue channels. The exact formula for the same is shown below:

$$gray = red * 0.114 + green * 0.299 + blue * 0.587$$

1.2 Horizontal Flip

Our second transformation is horizontal flip. When flipping an image horizontally, the pixels are rearranged from left to right, effectively reversing the order of columns.

2 Correctness

To ensure that the data transfer between the tasks in part 2 was successful and that the pixels were sent in order, we employed a verification method using the diff command. This command compares the output image generated by the code from part 1 with the output image generated by the code from part 2.

```
diff -s part1-output.ppm part2-output.ppm
```

By executing this command, we can determine whether the transfer was done correctly. If the data was transmitted in order, there should be no difference between the output images from part 1 and part 2. This method allowed us to validate our approach and ensure the accurate transmission of pixels.

3 Sample Outputs

The following two images show the sample output for the files: `sample_pp3_5mb.ppm` and `sample_pp3_8mb.ppm`. It is clearly visible that the transformations are working

(a) Original Image

(b) Transformed Image

Figure 1: Sample 1 (5mb)

(a) Original Image

(b) Transformed Image

Figure 2: Sample 2 (8mb)

(a) Original Image

(b) Transformed Image

Figure 3: Sample 3 (25mb)

(a) Original Image

(b) Transformed Image

Figure 4: Sample 3 (73mb)

Images Size	Sequential	Threads - Atomic operations	Threads - Semaphores	Shared Memory	Pipes
5mb	0.52	0.29	0.56	0.37	0.39
8mb	1.05	0.39	0.80	0.51	0.55
25mb	1.35	1.16	2.50	1.55	1.68
73mb	4.16	3.63	8.16	5.01	5.48

Table 1: Runtime for various implementations (in seconds)

4 Analysis

The run-times for the various methods are shown in the following table:

The results indicate that as the size of the images increases, all implementations experience an increase in runtime. While parallel processing techniques such as threads and pipes show improvements over sequential processing, they still face challenges with scaling as the computational load increases.

The observed increase in runtime for the multi-threaded and multi-process methods can be attributed to the substantial synchronization overhead inherent in these approaches. Despite utilizing atomic operations, semaphores for multi-threading, and shared memory or pipes for multi-processing, the overhead associated with coordinating and managing concurrent execution outweighs the benefits gained from parallelism.

5 Implementation

The following section shows the implementation using atomic operations, semaphores, shared memory and pipes

5.1 Threads synchronised using atomic operations

The implementation required incorporating a straightforward spin-lock mechanism to guarantee the atomicity of the critical section. Due to the simplicity of the code, debugging efforts remained minimal.

5.2 Threads synchronised using semaphores

This approach involved utilizing semaphores to maintain the atomicity of the critical section. While still manageable, the implementation was slightly more intricate than the previous method, necessitating a bit more debugging effort.

5.3 Process via shared memory

This implementation posed the greatest challenge as it required transferring processed pixels through shared memory to another process. Synchronizing the transfer and ensuring accurate pixel reading demanded meticulous attention. Debugging efforts were particularly intensive for this task.

5.4 Process via pipes

This implementation was considerably intricate, though not as demanding as the previous one. We needed to relay the data from the initial process executing the first transformation.