Operating Systems Laboratory

Lab 6

Table of Content

Transformations

Grayscale Transformation

The first of the two transformations is the Grayscale Transformation. With this, we convert the coloured image to a grayscale image. The transformation is done on every pixel with the help of the following formula.

$$(R, G, B) = 0.299 * R + 0.587 * G + 0.114 * B$$

Shubh Agarwal && Saksham Chhimwal

Original

Grayscaled

Invert Transformation

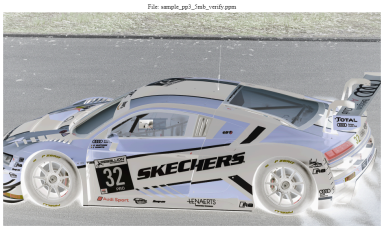
The second transformation inverts all the pixels of the image; it is done with the help of the following formulas.

$$R = 255 - \frac{(G+B)}{2}$$

$$G = 255 - \frac{(R+B)}{2}$$

$$B = 255 - \frac{(R+G)}{2}$$

Original



Inverted

Correctness

We take all the outputs we get after running part 1 as ground truth. We then compare the output produced after running part 2 with the ground truth.

Command:

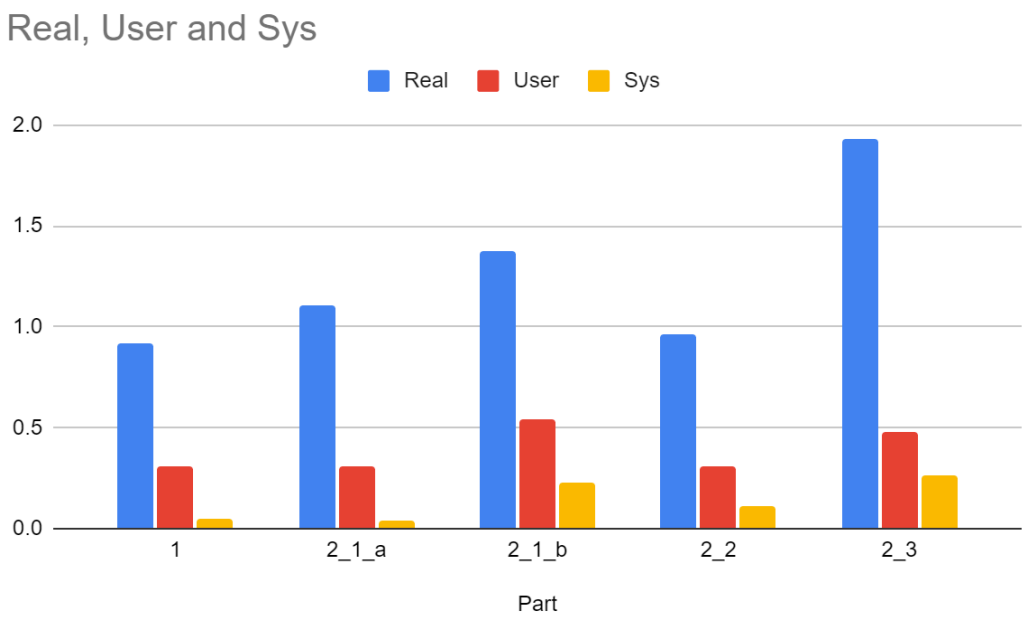
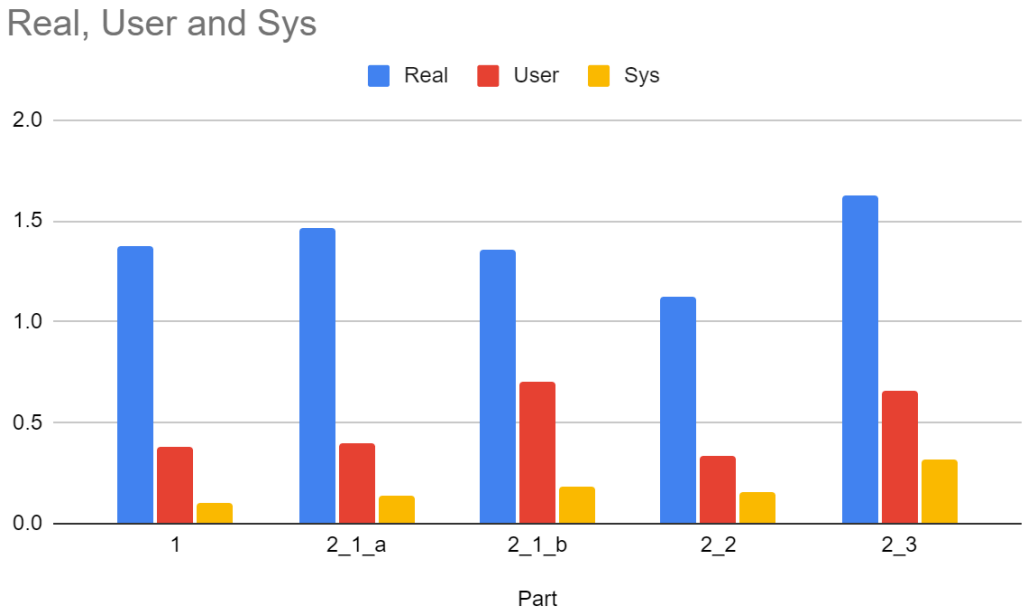
```
diff <output_of_part_1> <output_of_part2>
```

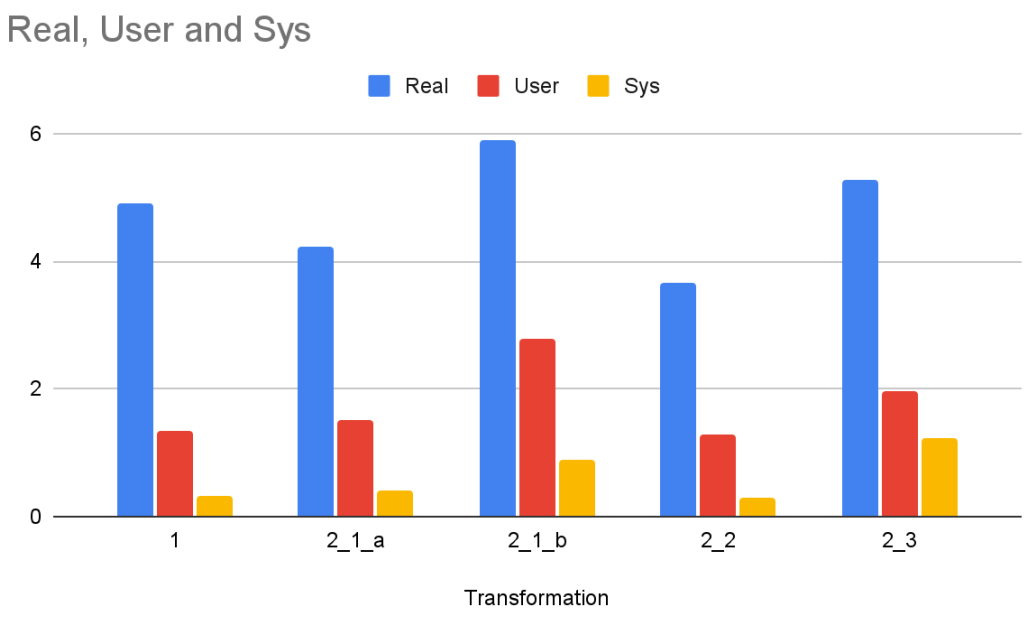
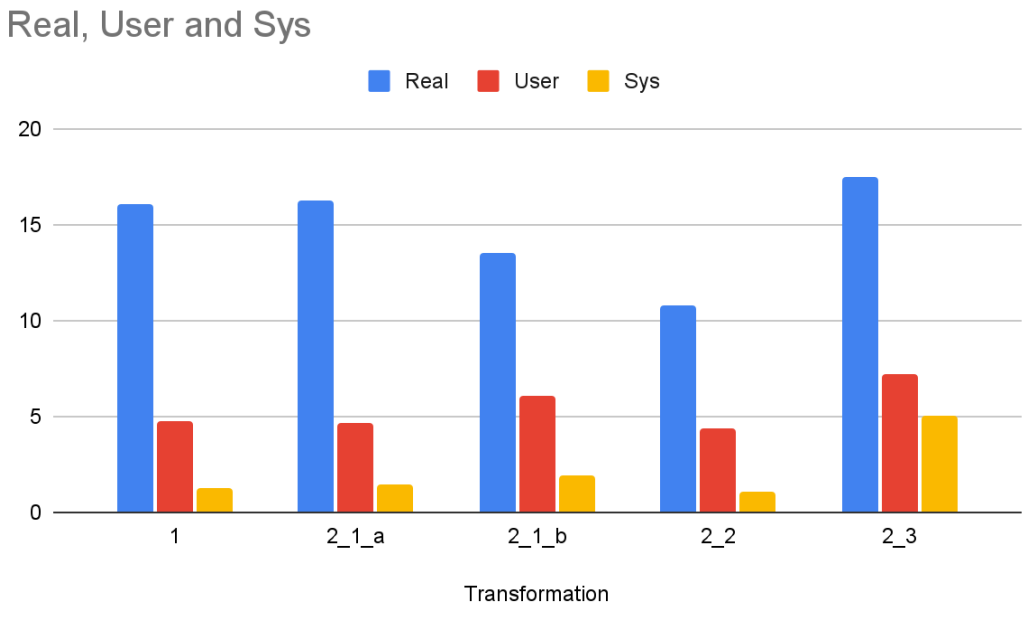
If both transformations are performed on all the input image pixels properly, then the output produced in part 1 must match the output produced in part 2.

Observations

We ran all the provided input files with the verifier program and all the program implementations asked in the assignment. Here are the results

Implementation /Time	5MB	8MB	25MB	73MB
1	real 0.915s user 0.309s sys 0.051s	real 1.376s user 0.378s sys 0.099s	real 4.904s user 1.336s sys 0.313s	real 16.071s user 4.772s sys 1.073s
2a	real 1.106s user 0.308s sys 0.042s	real 1.462s user 0.401s sys 0.137s	real 4.213s user 1.513s sys 0.414s	real 16.265s user 4.674s sys 1.441s
2b	real 1.377s user 0.540s sys 0.224s	real 1.355s user 0.706s sys 0.182s	real 5.895s user 2.797s sys 0.886s	real 13.495s user 6.093s sys 1.968s
3	real 0.962s user 0.309s sys 0.114s	real 1.122s user 0.336s sys 0.153s	real 3.650s user 1.276s sys 0.295s	real 10.750s user 4.361s sys 1.092s
4	real 1.931s user 0.476s sys 0.264s	real 1.626s user 0.660s sys 0.320s	real 5.280s user 1.956s sys 1.224s	real 17.444s user 7.191s sys 5.028s

Size	**Observation**
5MB	
8MB	

25MB	 Real, User and Sys Real User Sys 6 4 2 0 1 2_1_a 2_1_b 2_2 2_3 Transformation
73MB	 Real, User and Sys Real User Sys 20 15 10 5 0 1 2_1_a 2_1_b 2_2 2_3 Transformation

Implementation

Part 1

We created a custom image object to load the image data. And on it, we performed the transformations. Here is the Image class

Image Class

```
class Image
{
public:
    string ppmVersion;
    int height = -1, width = -1, maxColorValue = -1, valsInLine = -1;
    vector<vector<int>> colorSpace = {{/*RED*/},
                                      {/*GREEN*/},
                                      {/*BLUE*/}};
};
```

Grayscale Transformation

```
void toGrayscale(Image *image)
{
    int n = image->colorSpace[0].size();
    for (int i = 0; i < n; i++)
    {
        double newValue = 0.0;
        for (int j = 0; j < 3; j++)
            newValue += (double)image->colorSpace[j][i] * GRAYSCALE[j];
        for (int j = 0; j < 3; j++)
            image->colorSpace[j][i] = newValue;
    }
}
```

Invert Transformation

```
void toInvert(Image *image)
{
    int n = image->colorSpace[0].size();
    for (int i = 0; i < n; i++)
    {
        int R, G, B;
        R = image->colorSpace[0][i];
        G = image->colorSpace[1][i];
```

```
        B = image->colorSpace[2][i];
        image->colorSpace[0][i] = 255 - (G + B) / 2;
        image->colorSpace[1][i] = 255 - (R + B) / 2;
        image->colorSpace[2][i] = 255 - (G + R) / 2;
    }
}
```

Threads

Atomic Operations

- It was simple to implement the atomic operations so that all the pixels undergo both transformations.
- We added simple mutex locks to ensure the atomicity of the critical sections.
- Making it work took some time, but it all worked out.

Semaphores

- Implementing the semaphore operations to synchronise both transformations. It was done so that all the pixels in the image would undergo both transformations.
- Implementing it didn't take much time.

Process

Shared Memory

- This implementation took some work because of the custom Image class.
- So, we added two translation functions in the program that would convert the pixel value in the Image to arrays for three colours each.
- We also changed the transformations to work on array rather than class.
- To ensure that every pixel undergoes both transformations, the parent process was made to wait for the child process to finish first.

Pipe

- This was also simple to implement. At start, we took the naive approach and put the entire arrays in the pipe. Then we realised that the pipe could not handle such a huge amount of data. So we reverted to the original code and made some changes in the transformation functions.

- First, after grayscale, all the values of all the pixels are the same. So we piped the value from the grayscale function to the invert function and performed the transformations as usual.
- Since the pipe is blocking in nature, we did not need to insert waits to ensure the correctness of the program.