CS314: Operating systems lab

Assignment-6

Hamsini (210010038)

Sakeena (210010062)

Part-1:

The two transformations used are

1) Scale grey:
    - The image was colored in RGB format, we are converting a color image represented by a 2D vector of RGB into a grayscale image

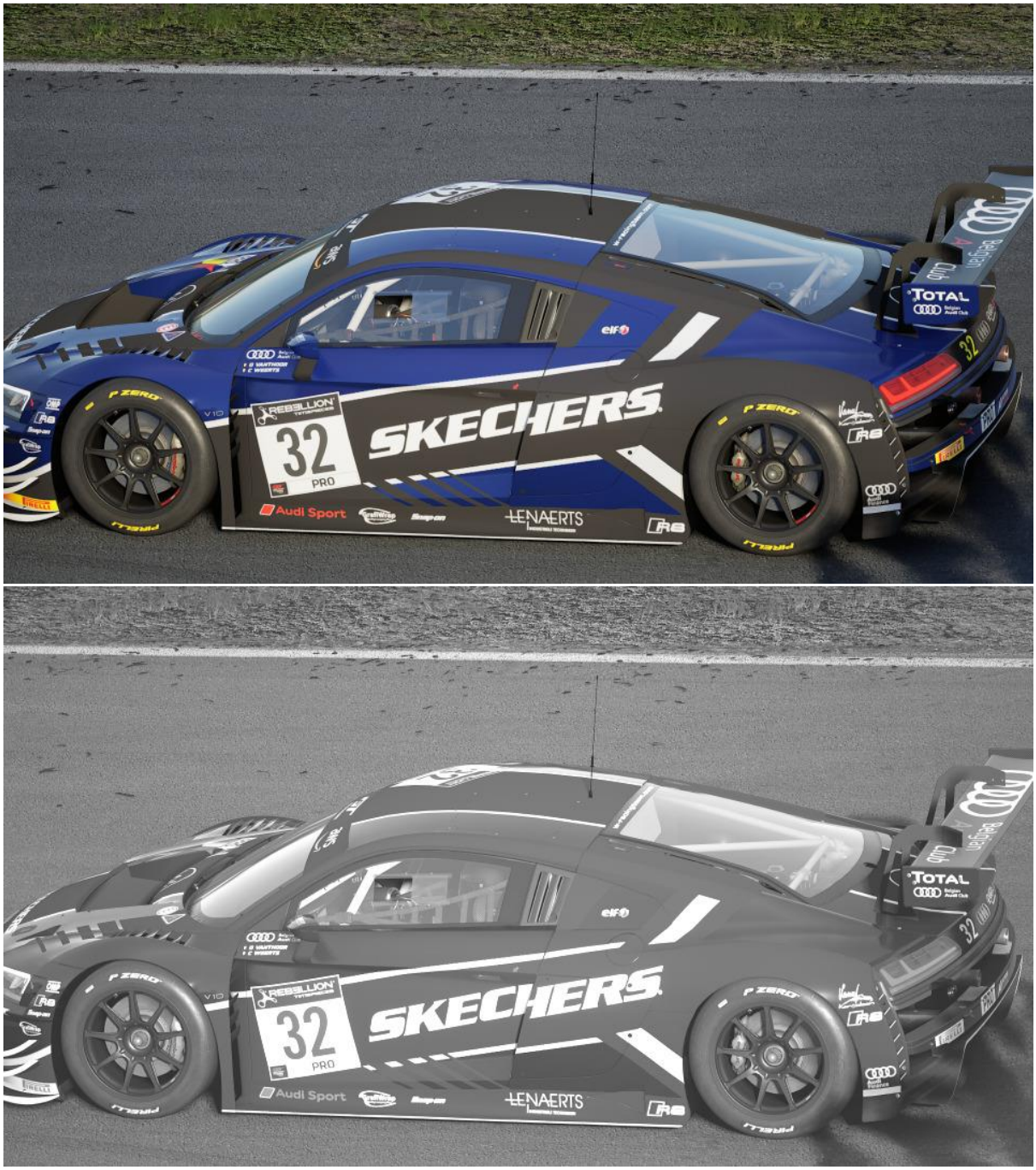
With grey = `(r*0.2899) + (g*0.5840) + (b*0.1261)`

    - After computing the grey scale value, it sets the red, green, and blue components of each pixel to this calculated grayscale value.
    - The result is that each pixel in the image is represented solely by its grayscale intensity, removing color information.
2) Brightness_change:
    - The image after the above transformation is now in black and white and greyscale we are now increasing the brightness of the changed image by some factor

```
int transformation(int value,int factor,int max)
{
    int sumofvaluefactor = value+factor;
    if (sumofvaluefactor >= max){
        return max;
    }else if (sumofvaluefactor <=0){
        return 0;
    }else{
        return sumofvaluefactor;
    }
}
image[h][w].r=transformation(image[h][w].r,factor,maxColor);

```

Here value is the deciding factor



Part-2: It is given that a processor is embedded with two cores. It is to be done by having the file read and T1 done on the first core, passing the transformed pixels to the other core, where T2 is performed on them, and then written to the output image file.

In the following subsections, different implementations using Synchronization

Primitives are explained. This a sequential processing one after another.

Part 2_1a:

Two distinct threads of the same process execute tasks T1 and T2.

Using the address space of the process itself, they converse. Atomic operations had to be used to synchronize in this situation. After using the above logic and appropriate flags, we got same image.

```
atomic_flag atomicFlag2False= ATOMIC_FLAG_INIT;
```

using threads and spinlock this is parallel processing with atomic operations

Part 2_1 b : Two distinct threads of the same process execute tasks T1 and T2.

Using the address space of the process itself, they converse. Atomic operations had to be used to synchronize in this situation.using

```
sem_t use_lock;
sem_wait(&use_lock);
            image[h][w].r=transformation(image[h][w].r,factor,maxColor);
            image[h][w].g=transformation(image[h][w].g,factor,maxColor);
            image[h][w].b=transformation(image[h][w].b,factor,maxColor);
            sem_post(&use_lock);
```

parallel processing with semapores

Part2_2:

T1 and T2 are carried out by 2 distinct processes that interact with one another through shared memory.

```
    key_t key_semaphore = 0x12345;
    int *finished;
    int shmid_semaphore = shmget(key_semaphore, sizeof(int) *2, 0666 |
IPC_CREAT);
    finished = (int*)shmat(shmid_semaphore, NULL, 0);
    finished[0]=-1;
    finished[1]=-1;
```

process-based parallelism with shared memory

Part2_3:

T1 and T2 are performed by 2 different processes that communicate via pipes. Process-based parallelism with pipe

Relative difficulties of implementation:

• The sequential approach was the easiest to implement; it's the most straightforward task, and doesn't use any synchronization primitives.

• The atomic operations and semaphores were also relatively easy to implement since they involved threads directly.

• The Shared Memory and Pipes were relatively harder to implement due to the need to check if values were properly passed between the different processes.

Analysis and Results:

• Generally, as the size of the image increases, the time taken for processing also increases across all parts.

• Process-Based Parallelism with Pipe takes more to execute than compared to others. This is because of the we read and write across two processes. Small critical session .

	A	B	C	D	E	F
1	**Image size**	**part-1(sequential)**	**Part2.1a(threads-atomic ops**	**Part2.1b(Threads -Semaphores)**	**Part2.2(Process-Shared memory**	**Part2.3(process-pipe)**
2	75mb	real 1m2.934s user 0m4.933s sys 0m8.609s	real 1m8.005s user 0m10.615s sys 0m8.738s	real 1m6.640s user 0m6.603s sys 0m9.696s	real 1m12.129s user 0m3.472s sys 0m6.981s	real 1m10.575s user 0m5.148s sys 0m9.756s
3	~25mb	real 0m23.186s user 0m1.889s sys 0m3.400s	real 0m25.152s user 0m3.142s sys 0m3.443s	real 0m24.841s user 0m1.796s sys 0m3.649s	real 0m21.110s user 0m1.773s sys 0m3.643s	real 0m21.861s user 0m1.787s sys 0m4.037s
4	~8mb	real 0m6.201s user 0m0.463s sys 0m1.122s	real 0m8.050s user 0m0.988s sys 0m1.044s	real 0m8.566s user 0m0.958s sys 0m1.154s	real 0m8.545s user 0m0.551s sys 0m1.237s	real 0m7.869s user 0m0.556s sys 0m1.494s
5	~5mb	real 0m5.494s user 0m0.285s sys 0m0.809s	real 0m5.800s user 0m0.558s sys 0m0.924s	real 0m5.613s user 0m0.645s sys 0m0.753s	real 0m5.258s user 0m0.561s sys 0m0.764s	real 0m6.558s user 0m0.433s sys 0m1.225s
6						