Operating System Homework 3 Report

Student ID:0410117

Name:吳沛璇

Detailed description of the implementation:

(Number of threads, the purpose of those threads, how do you use mutex lock and semaphore…etc.)

Programming assignment sets rule that both semaphore and mutexshould be used, so I used **semaphore** in Prob1 and put **mutex** in Prob2.

Number of thread：

(HW3-1)：RGBtoGrey-5 threads、Filter Convolve-10 threads

(HW3-2)：RGBtoGrey-1 threads、Filter Convolve-10 threads

Beacuse threads can share data, we don't need to use interprocess communication. Using threads makes **context switch faster**, comparing to calling fork() to create processes, that is the purpose of those threads.

The reason I implement more threads in Convolution is that its **run time complexity** is much higher, with RGBtoGrey function takes only O(N^2).

For Prob1, I put **binary semaphore** to make sure RGBtoGrey function will be done before Convolve function. Owing to different numbers of threads in these two function, sem_bin is signaled after RGBtoGrey function completes and waited before creating thread two.

```
Wait(S)
  {
   while S <= 0 do noop;  /* busy wait! */
   S = S - 1;             /* S >= 0 */
  }
Signal (S)
  {
   S = S + 1;
  }
```

Binary semaphore is **initialized to be 0**, so that thread two will **keep busy waiting** until thread one finish doing his jobs.

For Prob2, I used mutex lock to protect the critical section, preventing race condition to happen when we calculate image array's value.

Your speed:

HW3-1

```
> ./Speed.sh
Input a number of times to run './a.out' : 10

Run time:
   Finished once.
   Avg time: 856800 µs
>
```

HW3-2

```
> ./Speed.sh
Input a number of times to run './a.out' : 10

Run time:
   Finished once.
   Avg time: 828952 μs
>
```

Running on linux3.nctu.edu.tw.

Comparison	General version	Using threads	Speedup
HW3-1	1544042 us	856822 us	1.8 times
HW3-2	1430390 us	828952 us	1.7times

Problems encountered and solutions:

1.

Image(i, j) = sqrt(image_x(i, j)*image_x(i, j) + image_y(i, j)* image_y(i, j))

```
184         //extend the size form WxHx1 to WxHx3
185         for (int j = 0; j<imgHeight; j++) {
186             for (int i = 0; i<imgWidth; i++){
187             unsigned temp = sqrt((unsigned)pic_gx[j*imgWidth + i]*(unsigned)pic_gx[j*imgWidth + i]
188                                  + (unsigned)pic_gy[j*imgWidth + i]*(unsigned)pic_gy[j*imgWidth + i]);
189             //temp = sqrt(temp);
190             if (temp < 0) temp = 0;
191             if (temp > 255) temp = 255;
192             pic_final[3 * (j*imgWidth + i) + MYRED] = (unsigned char)temp;
193             pic_final[3 * (j*imgWidth + i) + MYGREEN] = (unsigned char)temp;
194             pic_final[3 * (j*imgWidth + i) + MYBLUE] = (unsigned char)temp;
195         }
196     }
```

我第一個遇到的問題是結合 image_x 和 image_y 的部分，平方後相加再開根號即為所求，然而在平方時可能會有 overflow 的情況，所以要**用 unsigned int 存起來**，以免 **overflow** 導致值變成負的，在判斷>255 或<0 時黑白對調。

2.

```
 99  void *thr_func_2(void *arg) {
100
101      int index = (long)arg;
102      int start = imgHeight*(index) / NUM_THREADS_2 ;
103      int end = imgHeight*(index+1) / NUM_THREADS_2 ;
104      //sem_wait(&bin_sem);
105      cout<<"start convolve"<<endl;
106      //apply the Gaussian filter to the image
107      for (int j = start; j<end; j++) {
108          for (int i = 0; i<imgWidth; i++){
109              pic_gx[j*imgWidth + i] = SobelFilter(i, j, filter_GX);
110              pic_gy[j*imgWidth + i] = SobelFilter(i, j, filter_GY);
111          }
112      }
113      pthread_exit(NULL);
114  }
```

我第二個遇到的問題比較低級一點，因為我將每個 thread 要做的事切割成等分，用 thread 的 index 去算出 start 和 end 位置，算法是*index/thread number，一開始我先除再乘 MAE 就會算出小誤差(約 0.5)，要先乘再除才行，才不會因為 type 是 int 而在做除法時有值一開始就被略去，後面*index 又把誤差再放大。