執行結果

```
cs4108033007@cs4108033007-VirtualBox:~$ ./code
Producer: Temporary max=2143124030 and min=19485054
Producer: Put 2143124030 into max_buffer at 0
Producer: Put 19485054 into min_buffer at 0
Consumer: Updated! minimum=19485054
Consumer: Updated! maximum=2143124030
Producer: Temporary max=2118421993 and min=2416949
Producer: Put 2118421993 into max_buffer at 1
Producer: Put 2416949 into min_buffer at 1
Consumer: Updated! minimum=2416949
Producer: Temporary max=2135019593 and min=6072641
Producer: Put 2135019593 into max_buffer at 2
Producer: Put 6072641 into min_buffer at 2
Producer: Temporary max=2147469841 and min=6939507
Producer: Put 2147469841 into max_buffer at 3
Producer: Put 6939507 into min_buffer at 3
Consumer: Updated! maximum=2147469841
Success! maximum= 2147469841 and minimum= 2416949
```

程式碼說明：

```
// semaphore declare
// 1. mutual exclusion
sem_t mutex_maxbuffer;
sem_t mutex_minbuffer;

// 2. Synchronous
sem_t sem_maxbuffer;
sem_t sem_minbuffer;

// big_buffer
int big_buffer[SIZE];

// max_buffer, min_buffer and index for them
int max_buffer[4];
int min_buffer[4];
int index_max = 0;
int index_min = 0;
```

1. mutex_maxbuffer: 提供 max_buffer 互斥存取機制
2. mutex_minbuffer: 提供 min_buffer 互斥存取機制
3. sem_maxbuffer: 避免 consumer 在 max_buffer 為空時還去拿取的同步機制
4. sem_minbuffer: 避免 consumer 在 min_buffer 為空時還去拿取的同步機制

```c
void main()
{
        // random initialize big buffer
        for(int i = 0 ; i < SIZE ; i++)
        {
                big_buffer[i] = rand();
        }

        // create Producer and consu
        producer_info data1 = {0, SIZE/4};
        producer_info data2 = {SIZE/4, 2 * (SIZE / 4)};
        producer_info data3 = {2 * (SIZE / 4), 3 * (SIZE/4)};
        producer_info data4 = {3 * (SIZE / 4), SIZE};


        // sempahore initialize
        sem_init(&mutex_maxbuffer, 0, 1);
        sem_init(&mutex_minbuffer, 0, 1);
        sem_init(&sem_maxbuffer, 0, 0);
        sem_init(&sem_minbuffer, 0, 0);

        // init pthread id
        pthread_t p1, p2, p3, p4, c1, c2;
```

```c
        // pthread exec
        pthread_create(&p1, NULL, producer, (void*) &data1);
        pthread_create(&p4, NULL, producer, (void*) &data4);
        pthread_create(&p2, NULL, producer, (void*) &data2);
        pthread_create(&p3, NULL, producer, (void*) &data3);
        pthread_create(&c1, NULL, consumer1, (void*) NULL);
        pthread_create(&c2, NULL, consumer2, (void*) NULL);

        // wait for each thread
        pthread_join(p1, NULL);
        pthread_join(p2, NULL);
        pthread_join(p3, NULL);
        pthread_join(p4, NULL);
        pthread_join(c1, NULL);
        pthread_join(c2, NULL);

        printf("Success! maximum= %d and minimum= %d \n", maximum, minimum);
}
```

於 main 中進行 big_buffer 內數值的隨機初始化、Thread 生成與指派任務以及 Semaphore 的初始化

```
void* producer(void* arg)
{
        producer_info *data = (producer_info*)arg;
        int begin = data->begin_index;
        int end = data->end_index;
        int max = 0;
        int min = 2147483647;

        for(int i = begin ; i < end ; i++)
        {
                int item = big_buffer[i];
                if(item > max)
                {
                        max = item;
                }
                if(item < min)
                {
                        min = item;
                }
        }
        printf("Producer: Temporary max=%d and min=%d \n", max, min);
```

```
        //----------critial section for max_buffer-----------------------
        sem_wait(&mutex_maxbuffer);
        max_buffer[index_max] = max;
        printf("Producer: Put %d into max_buffer at %d \n", max, index_max);
        index_max++;
        sem_post(&mutex_maxbuffer);
        sem_post(&sem_maxbuffer);
        //----------------------------------------------------------------

        //----------critial section for min_buffer-----------------------
        sem_wait(&mutex_minbuffer);
        min_buffer[index_min] = min;
        printf("Producer: Put %d into min_buffer at %d \n", min, index_min);
        index_min++;
        sem_post(&mutex_minbuffer);
        sem_post(&sem_minbuffer);
        //----------------------------------------------------------------
        pthread_exit(NULL);
}
```

但由於四個 producer、兩個 consumer 都會去使用到這個共享的 buffer，因此必須提互斥存取機制，這裡使用 mutex_maxbuffer, mutex_minbuffer 來提供。

```c
void* consumer1(void* arg)
{
        for(int i = 0 ; i < 4 ; i++)
        {
                //----------critial section for max_buffer---------
                sem_wait(&sem_maxbuffer);
                sem_wait(&mutex_maxbuffer);
                int temp = max_buffer[i];
                if(temp > maximum)
                {
                        maximum = temp;
                        printf("Consumer: Updated! maximum=%d \n", maximum);
                }
                sem_post(&mutex_maxbuffer);
                //-----------------------------------------------------------
        }
        pthread_exit(NULL);
}
```

```c
void* consumer2(void* arg)
{
        for(int i = 0; i < 4; i++)
        {
                //----------critial section for min_buffer-----------------
                sem_wait(&sem_minbuffer);
                sem_wait(&mutex_minbuffer);
                int temp = min_buffer[i];
                if(temp < minimum)
                {
                        minimum = temp;
                        printf("Consumer: Updated! minimum=%d \n", minimum);
                }
                sem_post(&mutex_minbuffer);
                //-----------------------------------------------------------
        }
        pthread_exit(NULL);
}
```

consumer (1 負責更新 maximum, 2 負責更新 minimum)

Consumer 目的為歷遍過整個 max_buffer/min_buffer 後找出其中最大值當作最後的 maximum 答案，因此這邊先用 for 迴圈讓其能夠去檢查 max_buffer/min_buffer 上的四個位置。

這邊 consumer 如果先執行可能會遇到 max_buffer/min_buffer 內還沒有東西的情境，因此要使用 sem_maxbuffer/sem_minbuffer 來進行同步控制，確保 buffer 內是有東西的，consumer 才能進入 critical section 來進行取物。這邊要小心的是要先測同步機制再測互斥機制不然有可能會發生 Deadlock。