Hw3 Report

Q1-6.2

- $Mutual\ exclusion$:

  使用兩個變數$flag$跟$turn$, 即使兩個$process$的$flag$都為$true$, 但因為$turn$的值只可能是i或者j其中一個, 不能既是i也是j, 這樣保證只有一個$process$可以成功進入$critical\ section$. 而正在等待的$process$必須等到另一個$process$將$turn$的值更新後才能夠進入自己的$critical\ section$.

- $Progress$:

  使用兩個變數$flag$跟$turn$, 如果其中一個$process$進入$critical\ section$, 他會將$turn$設成另一個$process$後離開$critical\ section$. 如果同個$process$想要再次進入$critical\ section$, 他會不斷重複進入$critical\ section$然後將$turn$設成另一個$process$後離開, 直到另一個$process$進入$critical\ section$.

- $Bounded\ Waiting$:

  當兩個$Process$都準備好要進入$critical\ section$時, 只有輪到自己的$turn$的$process$可以進入. $Dekker's\ algorithm$中有將$turn$設置成另一個$process$的設定, 這個設定可以保證另一個$process$一定可以在下一輪進入$critical\ section$.

  如果沒有這個設定, 那很有可能發生某個$process$總是沒有辦法進入自己的$critical\ section$的狀況

Q2-6.20

```
Monitor boundedbuffer
{
  int buffer_size = 15;
  item buffer[buffer_size];
  int int, out, count;
  condition producer, consumer;

  void produce(item i)
  {
    if (count == buffer_size)  // the buffer is full
      producer.wait();
    buffer[in] = item;
    count += 1;
    in = (in+1) % buffer_size;
    consumer.signal();
  }

  void consume(item j)
  {
    if(count == 0) // the buffer is empty
```

```
      consumer.wait();

    j = buffer[out];
    count -= 1;
    out = (out+1) % buffer_size;
    producer.signal();
  }

  initialization code()
  {
    in = 0;
    out = 0;
    count = 0;
  }
}
```

Q3

- 先設定NUMBER_OF_THREAD = 5（除了main thread外再產生5個child threads）
- 先設定INTERVAL = 1000, 以供random generate points和計算point是否在圓內使用
- 設定變數circle_points = 0, total_points = INTERVAL*NUMBER_OF_THREAD作為後續計算公式所需要的變數
- 根據題目提供的公式，撰寫child thread要執行的功能的function：
  - 使用rand()隨機產生x, y：
    - x = (rand() % (INTERVAL + 1)) / (*double*)INTERVAL;
    - y = (rand() % (INTERVAL + 1)) / (*double*)INTERVAL;
    - 計算$(x^2 + y^2)$是否> 1
      - < 1代表(x, y)落在半徑$= 1$的圓內, 為circle point, 反之則為square point
      - 若為circle point, 則將in_circle ++
  - 利用迴圈更新完in_circle後, 為了確保在更新circle_points時不會出現race condition, 使用mutex lock：
    - 更新circle_points前將mutex lock鎖上
    - 更新circle_points += in_circle
    - 將mutex lock解開
  - child thread完成工作後結束
- 5個child threads都完成工作後, 將threads進行join, 然後回到main thread
- Main thread根據$\pi = 4$*circle points/total points計算π並印出計算結果