Problem 2:

將整個 board 的位置參數(i,j)hash 成一維向量傳進 pthread_create 的 start function，平分成兩半給兩條 thread 處理，把新的 board 狀態存在一個新的 map 裡，再由 main thread_join2 thread 後，把 map 更新成新的 map。

Problem 3:

和 2 處裡方式相同，只是將 board 切成 20 分由 20 條 thread 處裡。

Problem 4:

-t 2

```
b08902043@linux1 [~/SP_practice/hw/hw4] time ./main -t 2 ./largeCase.txt ./output.txt

real    0m2.167s
user    0m3.554s
sys     0m0.072s
```

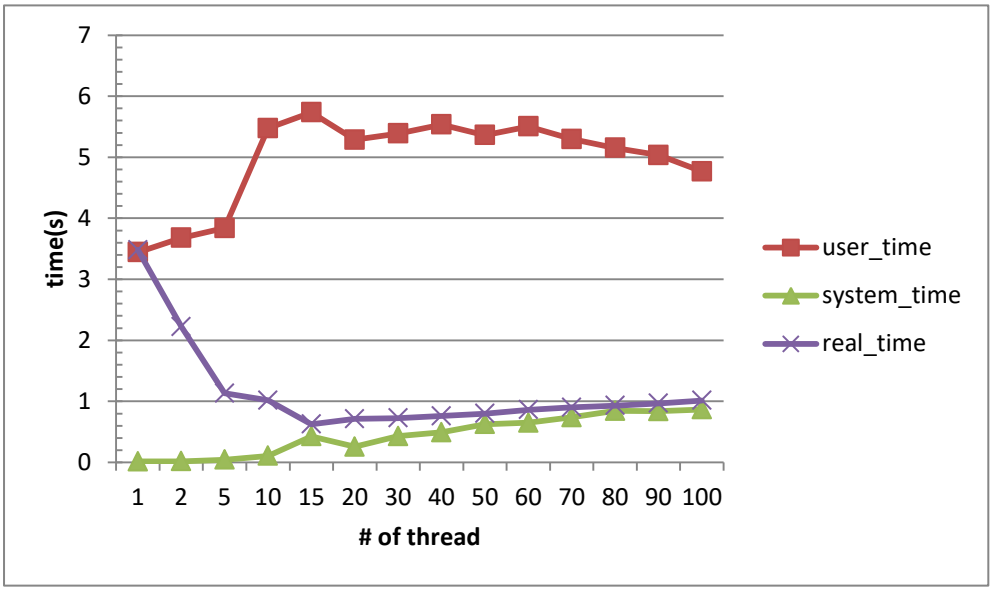
-t 20

```
b08902043@linux1 [~/SP_practice/hw/hw4] time ./main -t 20 ./largeCase.txt ./output.txt

real    0m0.704s
user    0m5.646s
sys     0m0.206s
```

當有多個計算單元時，有越多條 thread 越能同時處裡平行工作，在 largeCase 中 20 條 thread 可以將 task 拆得更小平行處裡，速度較快。

Problem 5



由此圖可以看出幾個結果:

1. 從# of thread = 1 ~ 15 的區間中，可以看到 real time 顯著下降，在 15 為最低點，推斷工作站的可用運算單元可能在 15~20 之間，因此才能平行處裡各條 thread(在本機測試 20 條 thread 會比 2 條慢，可能因為我電腦的運算單元< 20，需很多 overhead 去對 thread 做 thread 的 context switch)
2. Real time 在 # of thread > 15 遞增，和上面用本機測試的結果原因相似，可能可用的運算單元 < 15，在 15 條 thread 後無法藉由平行處裡增加效率，反而多了很多 context switch 和 system call 的 overhead
3. System time 在# of thread > 20 後遞增，因為 pthread_create 和 pthread_join 都為 system call，thread 數量增加，call system call 的次數增加，system time 也會增加
4. User time 在 0~20 之間遞增，後來維持平衡，0~20 之間時，可平行處裡的 thread 較多，user time

可能會> real time，且 thread 越多，user time 越高，而到 > 20 之後，因為可平行處裡的 thread 數量固定了，所以 user time 部會在上升，應該會大致維持定值(但不知為何實驗結果會有些許遞減)

Problem 6:

這個部分的重點應該是要把整個 board 做成 mmap，因為 fork 完以後 child 不像 thread 一樣共享 variable，mmap 可以讓 parent(不更新 board，只生成 process)，child1，child2 共享 board 並更新，其他部分像切 task 等等都和 thread 差不多。

和 problem2 的比較，測出來結果是沒有差很多

```
b08902043@linux15 [~/SP_practice/hw/hw4] time ./main -p 2
./largeCase.txt ./output.txt

real    0m2.192s
user    0m3.903s
sys     0m0.059s
```

但有觀察到 process 開到 100 的時候 system time 會比 thread 開 100 高很多，應該是 fork 的 cost 比較高。