作業系統程式作業 1

編譯方式：~~“g++ -o BSS BSS3.cpp -lrt -std=c++17”~~ -> “g++ -o BSS BSS4.cpp –lrt”

不要開 –O2

程式執行：“./BSS #(總船艦數量-2)”

總船艦數量至少為 2，最多為 100。

```
s1061506@s1061506-VirtualBox:~/桌面$ ls
BSS  BSS3.cpp
s1061506@s1061506-VirtualBox:~/桌面$ g++ -o BSS BSS3.cpp -lrt -std=c++17
s1061506@s1061506-VirtualBox:~/桌面$ ./BSS
run this program with "./BSS #num", and #num should be less than 98: Success
s1061506@s1061506-VirtualBox:~/桌面$ ./BSS 28
[3189 Parent]: The gunboat: (3,0)(3,1)
[3190 Child]: The gunboat: (3,0)(3,1)
[3191 Child]: The gunboat: (2,0)(1,0)
[3192 Child]: The gunboat: (0,0)(0,1)
[3193 Child]: The gunboat: (0,3)(0,2)
[3194 Child]: The gunboat: (3,0)(3,1)
[3195 Child]: The gunboat: (1,0)(0,0)
[3196 Child]: The gunboat: (3,3)(3,2)
[3197 Child]: The gunboat: (0,3)(1,3)
[3198 Child]: The gunboat: (3,1)(3,0)
[3199 Child]: The gunboat: (2,1)(1,1)
[3200 Child]: The gunboat: (2,0)(2,1)
[3201 Child]: The gunboat: (2,0)(3,0)
[3202 Child]: The gunboat: (1,1)(2,1)
```

……

```
[3218 Child]: bombing (0,0)
[3217 Child]: missed
[3202 Child]: missed
[3199 Child]: missed
[3199 Child]: bombing (0,1)
[3202 Child]: missed
[3218 Child]: missed
[3217 Child]: missed
[3202 Child]: bombing (1,1)
[3218 Child]: hit and sinking
[3217 Child]: hit and sinking
[3199 Child]: hit and sinking

[3189 Parent]: 3203 makes 9 hits!!
[3189 Parent]: 3202 makes 8 hits!!
[3189 Parent]: 3198 makes 7 hits!!
[3189 Parent]: 3193 makes 7 hits!!
[3189 Parent]: 3190 makes 6 hits!!

[3189 Parent]: 3202 wins with 8 bombs
```

程式在有人勝出後，會先列出前五名命中數最多的船艦，然後才印出最終贏家。

程式中除了 function error_and_die() 以及 shared memory 的建立是直接使用或修改自老師給的範例程式中的程式碼，其餘部分皆為獨立完成。

關於座標：座標由兩個範圍是[0, 4)的整數組成，共有 16 個座標點。

關於同步機制：使用了 Peterson 算法。

line 27~63: SHM_

抽象化並分離 shared memory 建立以及歸還的輔助 template struct，能夠在實例構造出來時自動取得一塊相應大小的 shared memory 並初始化記憶體，且只有在 parent process 中的實例被銷毀(析構, destruct)時自動將 shared memory 歸還給系統。

line 117~125: init_game

初始化 shared memory 中基本變數的 function。

line 98~115: creat_n_battleship

將建立所需的 n 個 process 的動作抽離出來的 function。

line 127~142: init_self

每個 process 各自初始化自身的戰艦位置，攻擊位置順序。

- 戰艦位置決定：先隨機決定一個合法位置，接著隨機決定上下左右四個方向的順序。先從起始位置往其中一個方向延伸一格，若延伸座標在範圍外則繼續挑選下個方向，直到延伸座標在範圍內。此時起始座標及延伸座標就是戰艦所在位置的兩個座標。存入 my_pos。
- 攻擊位置順序決定：先在一個陣列內填入所有可以攻擊的 16 個座標點，並使用洗牌演算法將其打亂。此時陣列內的座標依序為戰艦的攻

擊位置順序。存入 attack_stack。

line 144~257: main function

- line 145~156: 檢查傳入的參數是否合法
- line 158~173: 進行初始化並建立 child process
- line 175~183: 所有 process 在此進行 busy waiting。由 parent process 進行檢查，只有在確定所有的 process 都已經初始化完畢，或是在 fork()時出現問題時，才可以離開此區塊。
- line 185~240: 戰艦互相轟炸的核心部分。
  - 存活者留在此區塊，沉沒者離開。
  - line 186~219: 當前的攻擊者決定攻擊座標(187~189)，確認當前的所有存活者皆已離開 line 237 的迴圈後(190~191)，開始詢問存活者攻擊結果(193~196)。等待當前所有存活的 process 回報結果後(198)，開始統計各項數據(199~209)。若發現僅剩自己存活，則自己成為贏家，並離開此區塊(210~214)；否則輪下一個存活的 process 進行攻擊(215~217)。
  - line 220~238: 受擊者確認攻擊者已經選定好攻擊座標後(220)，開始檢查自身是否被攻擊命中/擊沉並回報結果(222~236)。並在選定下一個攻擊者前在 line 237 進行 busy waiting
- line 242~245: 除了最後一個 process 以外，其他的 process 只要先被

炸沉了就先在此處等待自己 fork()出來的 child process 結束。

- line 247: 只要不是 parent process 就先結束。
- line 249~254: 由 parent process 分別公布命中數最高的船艦以及最後的勝利者。