系統環境：VMware, Linux, Ubuntu 18.04

- Problem 7.17

  創建兩種進程函式（北方與南方），使用 mutex lock 保護共享變數，以及 condition variable 判斷是否等待，因為要避免雙向的農夫在橋上碰面，而同方向的農夫同時在橋上是允許的，所以等待的判斷式為：只有當沒有任何對向農夫在橋上才可以過橋，否則需等待直到所有對向的農夫經過橋，農夫過橋後發送廣播給所有對向的農夫，通知他們可以不需要等待，但如果對向農夫判斷還有農夫往他們這個方向過橋，則繼續等待，周而復始。

  程式碼：Source code\Chap7.17\farmers.c

  檔案編譯方式：gcc -o farmers farmers.c -lpthread

  輸出：Source code\Chap7.17\farmers_output.txt

```
Farmer A is awake.
Farmer B is awake.
Farmer D is awake.
Farmer C is awake.
[Northbound] Farmer A is passing.
[Northbound] Farmer B is passing.
[Northbound] Farmer B has passed.
[Waiting from south] Farmer D is waiting.
[Northbound] Farmer A has passed.
[Southbound] Farmer D is passing.
[Waiting from north] Farmer B is waiting.
[Southbound] Farmer C is passing.
[Southbound] Farmer C has passed.
[Southbound] Farmer D has passed.
[Northbound] Farmer B is passing.
[Northbound] Farmer A is passing.
[Northbound] Farmer A has passed.
[Waiting from south] Farmer D is waiting.
[Northbound] Farmer B has passed.
[Southbound] Farmer D is passing.
[Southbound] Farmer C is passing.
[Waiting from north] Farmer A is waiting.
[Southbound] Farmer D has passed.
[Waiting from north] Farmer B is waiting.
[Southbound] Farmer C has passed.
[Northbound] Farmer B is passing.
[Northbound] Farmer A is passing.
[Northbound] Farmer B has passed.
[Waiting from south] Farmer D is waiting.
[Waiting from south] Farmer C is waiting.
[Northbound] Farmer A has passed.
[Southbound] Farmer C is passing.
[Southbound] Farmer D is passing.
[Waiting from north] Farmer A is waiting.
[Southbound] Farmer D has passed.
[Southbound] Farmer C has passed.
[Northbound] Farmer A is passing.
[Northbound] Farmer B is passing.
[Waiting from south] Farmer D is waiting.
[Northbound] Farmer A has passed.
[Northbound] Farmer B has passed.
[Southbound] Farmer D is passing.
[Southbound] Farmer D has passed.
[Northbound] Farmer B is passing.
[Waiting from south] Farmer C is waiting.
[Northbound] Farmer A is passing.
[Northbound] Farmer B has passed.
[Northbound] Farmer A has passed.
[Southbound] Farmer C is passing.
[Southbound] Farmer D is passing.
[Waiting from north] Farmer A is waiting.
[Waiting from north] Farmer B is waiting.
[Southbound] Farmer C has passed.
[Southbound] Farmer C is passing.
[Southbound] Farmer D has passed.
[Southbound] Farmer D is passing.
[Southbound] Farmer D has passed.
[Southbound] Farmer C has passed.
[Northbound] Farmer B is passing.
[Northbound] Farmer A is passing.
[Northbound] Farmer A has passed.
[Northbound] Farmer B has passed.
```

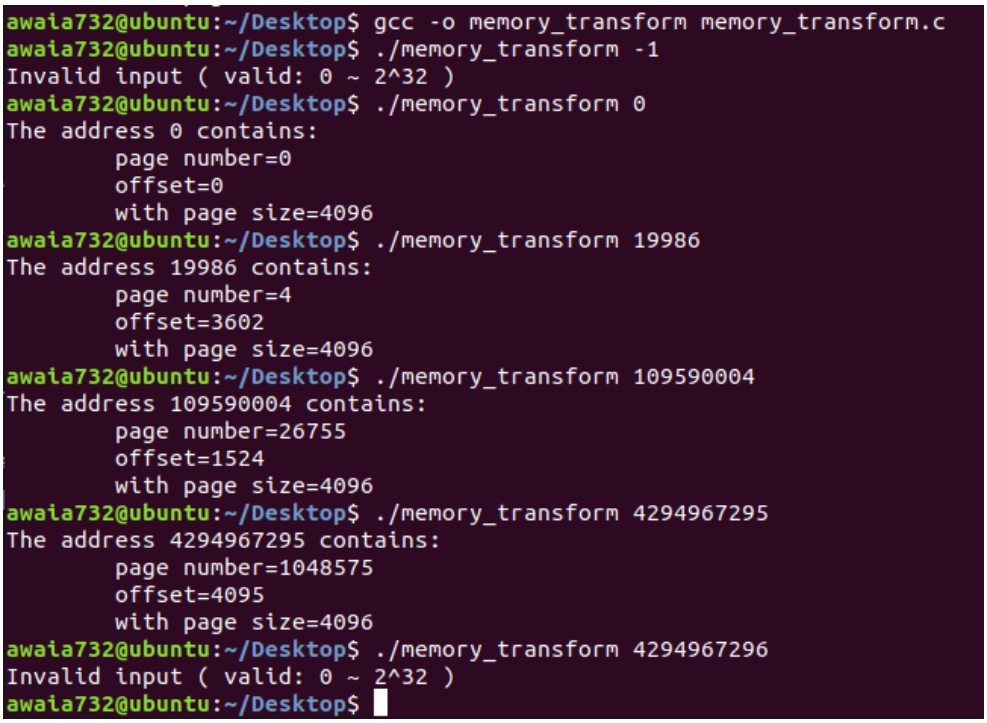
- Problem 8.25

  輸入一個 32-bits 的十進位數字，設輸入的資料型別為 unsigned int，確定執行程式有輸入參數，且用 unsigned long 來判斷輸入的數字範圍是否在 32-bits 中，設輸入為 virtual address，若 page size = 4086，則 page number = virtual address / page size（取商）、offset = virtual % page size（取餘）。

  程式碼：Source code\Chap8.25\memory_transform.c

  檔案編譯方式：gcc -o memory_transform memory_transform.c

  輸出： Source code\Chap8.25\memory_transform_output.png



- Problem 9.26

  先創建代表頁面參考的隨機字串，並依照以下三種方法進行頁面替換

  FIFO：用一個 Queue 來決定替換的順序，使用一個陣列當作 queue，並用 queue index 來當作 queue 對列的開頭，以此來決定哪一個頁面應該被替換掉。

  Optimal：尋找在 frame 中最久才會被再次呼叫的頁面做替換，這能得到最少的 page fault，但在實際應用時很難實現。

  Less-Recently-Used：尋找在 frame 中閒置最久的資源做替換，是最常被實際應用的方法。

  程式碼：Source code\Chap9.26\page_replace.c

  檔案編譯方式：gcc -o page_replace page_replace.c

  輸出：Source code\Chap9.26\page_replace_output.png

```
yuwei@ubuntu:~/Desktop$ ./page_replace
Page reference string : 6 4 0 1 4 3 6 0 0 8 0 4 8 4 1 9 9 2 0 3

[1 frames] Page fault 18 with FIFO algorithm.
[1 frames] Page fault 18 with Optimal algorithm.
[1 frames] Page fault 18 with LRU algorithm.

[2 frames] Page fault 15 with FIFO algorithm.
[2 frames] Page fault 14 with Optimal algorithm.
[2 frames] Page fault 16 with LRU algorithm.

[3 frames] Page fault 14 with FIFO algorithm.
[3 frames] Page fault 11 with Optimal algorithm.
[3 frames] Page fault 14 with LRU algorithm.

[4 frames] Page fault 14 with FIFO algorithm.
[4 frames] Page fault 9 with Optimal algorithm.
[4 frames] Page fault 14 with LRU algorithm.

[5 frames] Page fault 9 with FIFO algorithm.
[5 frames] Page fault 8 with Optimal algorithm.
[5 frames] Page fault 11 with LRU algorithm.

[6 frames] Page fault 8 with FIFO algorithm.
[6 frames] Page fault 8 with Optimal algorithm.
[6 frames] Page fault 9 with LRU algorithm.

[7 frames] Page fault 8 with FIFO algorithm.
[7 frames] Page fault 8 with Optimal algorithm.
[7 frames] Page fault 9 with LRU algorithm.
```