

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

```
1 Farmer A is awake.
2 Farmer B is awake.
3 Farmer D is awake.
4 Farmer C is awake.
5 [Northbound] Farmer A is passing.
6 [Northbound] Farmer B is passing.
7 [Northbound] Farmer B has passed.
8 [Waiting from south] Farmer D is waiting.
9 [Northbound] Farmer A has passed.
10 [Southbound] Farmer D is passing.
11 [Waiting from north] Farmer B is waiting.
12 [Southbound] Farmer C is passing.
13 [Southbound] Farmer C has passed.
14 [Southbound] Farmer D has passed.
15 [Northbound] Farmer B is passing.
16 [Northbound] Farmer A is passing.
17 [Northbound] Farmer A has passed.
18 [Waiting from south] Farmer D is waiting.
19 [Northbound] Farmer B has passed.
20 [Southbound] Farmer D is passing.
21 [Southbound] Farmer C is passing.
22 [Waiting from north] Farmer A is waiting.
23 [Southbound] Farmer D has passed.
24 [Waiting from north] Farmer B is waiting.
25 [Southbound] Farmer C has passed.
26 [Northbound] Farmer B is passing.
27 [Northbound] Farmer A is passing.
28 [Northbound] Farmer B has passed.
29 [Waiting from south] Farmer D is waiting.
30 [Waiting from south] Farmer C is waiting.
31 [Northbound] Farmer A has passed.
32 [Southbound] Farmer C is passing.
33 [Southbound] Farmer D is passing.
34 [Waiting from north] Farmer A is waiting.
35 [Southbound] Farmer D has passed.
36 [Southbound] Farmer C has passed.
37 [Northbound] Farmer A is passing.
38 [Northbound] Farmer B is passing.
39 [Waiting from south] Farmer D is waiting.
40 [Northbound] Farmer A has passed.
41 [Northbound] Farmer B has passed.
42 [Southbound] Farmer D is passing.
43 [Southbound] Farmer D has passed.
44 [Northbound] Farmer B is passing.
45 [Waiting from south] Farmer C is waiting.
46 [Northbound] Farmer A is passing.
47 [Northbound] Farmer B has passed.
48 [Northbound] Farmer A has passed.
49 [Southbound] Farmer C is passing.
50 [Southbound] Farmer D is passing.
51 [Waiting from north] Farmer A is waiting.
52 [Waiting from north] Farmer B is waiting.
53 [Southbound] Farmer C has passed.
54 [Southbound] Farmer C is passing.
55 [Southbound] Farmer D has passed.
56 [Southbound] Farmer D is passing.
57 [Southbound] Farmer D has passed.
58 [Southbound] Farmer C has passed.
59 [Northbound] Farmer B is passing.
60 [Northbound] Farmer A is passing.
61 [Northbound] Farmer A has passed.
62 [Northbound] Farmer B has passed.
```

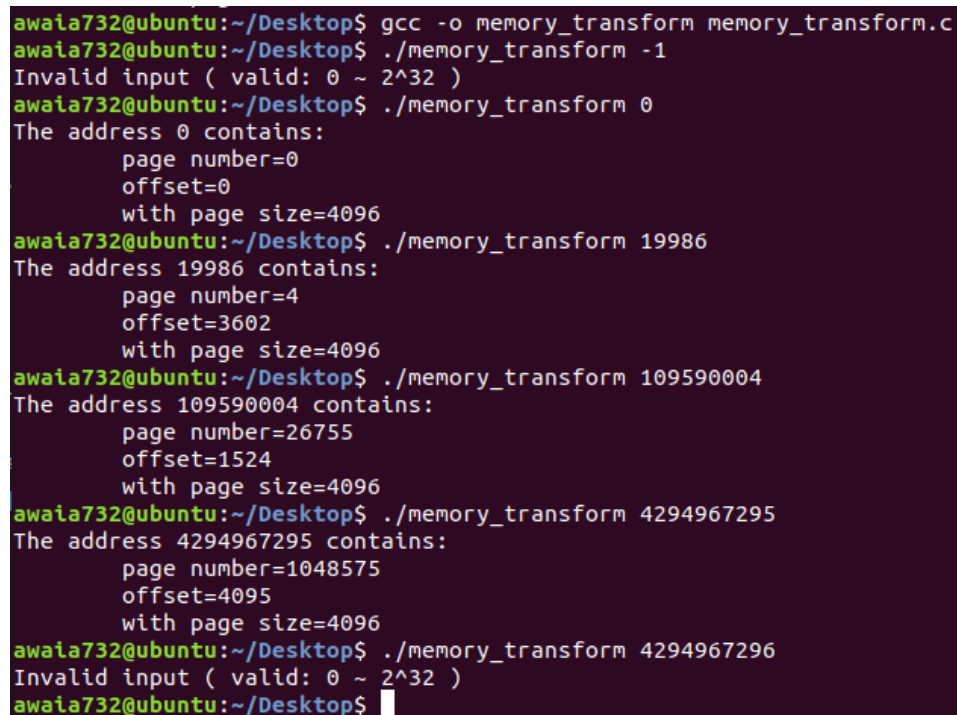
## ● Problem 8.25

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 = 4096，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



```
awaia732@ubuntu:~/Desktop$ gcc -o memory_transform memory_transform.c
awaia732@ubuntu:~/Desktop$ ./memory_transform -1
Invalid input ( valid: 0 ~ 2^32 )
awaia732@ubuntu:~/Desktop$ ./memory_transform 0
The address 0 contains:
    page number=0
    offset=0
    with page size=4096
awaia732@ubuntu:~/Desktop$ ./memory_transform 19986
The address 19986 contains:
    page number=4
    offset=3602
    with page size=4096
awaia732@ubuntu:~/Desktop$ ./memory_transform 109590004
The address 109590004 contains:
    page number=26755
    offset=1524
    with page size=4096
awaia732@ubuntu:~/Desktop$ ./memory_transform 4294967295
The address 4294967295 contains:
    page number=1048575
    offset=4095
    with page size=4096
awaia732@ubuntu:~/Desktop$ ./memory_transform 4294967296
Invalid input ( valid: 0 ~ 2^32 )
awaia732@ubuntu:~/Desktop$
```

## ● 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.
```