다음과 같은 Task Set을 고려

```
/* Code /  Name  / Run / Period / Core / Priority / Subnum / my_ptr / splitted_ptr / Dependency / Heavy */

{vTask0, "TASK 0",  3, 30,  Core0, 7,1, NULL, NULL, false,false},
{vTask1, "TASK 1",  2, 30, Core1, 6,1, NULL, NULL, false,false},
{vTask2, "TASK 2",  3, 30, Core1, 5,1, NULL, NULL, false,false},
{vTask3, "TASK 3",  8, 35, Core0, 4,1, NULL, NULL, false,false},
{vTask4, "TASK 4",  36,  50, Core0, 3,1, NULL, NULL,  false,false},
{vTask5, "TASK 5",  12, 100,  0, 2,1, NULL, NULL,  false,false},
{vTask6, "TASK 6",  12, 100,  0, 1,1, NULL, NULL,  false,false},

// * Extra Tasks
{vTask7, "Extra T",   0,   0,  0, 1,1, NULL, NULL,  false,false},
```

- **SPA1** 알고리즘

```
na_osdc@DESKTOP-3RRT7B1:~/osdc/lab/Task_split_Scheduling/SPA1$ ./main
For 7 tasks Utilization Bound is :  0.728627
For 7 tasks Light_Bound is :  0.421506
-------------------------------------------------------------
Normal_assign -> P1 (0.000000) : Gonna assign Task(6,1) 0.120000  After :  0.120000
Normal_assign -> P0 (0.000000) : Gonna assign Task(5,1) 0.120000  After :  0.120000
Split -> P1 (0.120000) : Gonna assign Task(4,1) 0.608627  After :  0.728627
Normal_assign -> P0 (0.120000) : Gonna assign Task(4,2) 0.111373  After :  0.231373
Normal_assign -> P0 (0.231373) : Gonna assign Task(3,1) 0.228571  After :  0.459945
Normal_assign -> P0 (0.459945) : Gonna assign Task(2,1) 0.100000  After :  0.559945
Normal_assign -> P0 (0.559945) : Gonna assign Task(1,1) 0.066667  After :  0.626612
Normal_assign -> P0 (0.626612) : Gonna assign Task(0,1) 0.100000  After :  0.726612
-------------------------------------------------------------
P1 Utilization : 0.728627  Wait Q -> Task (4,1) : 0.608627  / Task (6,1) : 0.120000  /
P0 Utilization : 0.726612  Wait Q -> Task (0,1) : 0.100000  / Task (1,1) : 0.066667  / Task (2,1) : 0.100000  / Task (3,1) : 0.228571  / Task (4,2) : 0.111373  / Task (5,1) : 0.120000  /
(0.405238 * 50.000000) / 19.568670 + 0.284571 = 1.540956
IN Tail task(4,2) in Processor 0  and Im Not Schedulable.
```

SPA1을 통해 파티셔닝하는 경우 Lemma5 식을 만족하지 못하는 것을 확인

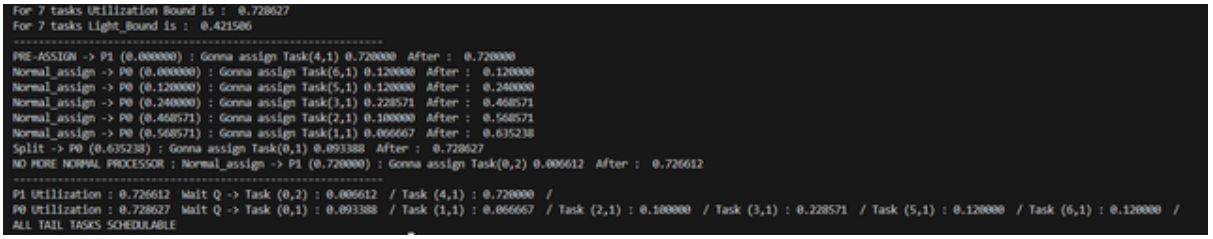
```
For 7 tasks Utilization Bound is :  0.7286
Core 0 Utilization : 0.7266, Core 1 Utilization : 0.7286
MONITOR ACTIVATED LCM: 2100
TASK 0(1)  (3 , 30) Utilization : 0.100 priority: 7 at : CORE 0
TASK 4(1)  (30 , 50) Utilization : 0.609 priority: 3 at : CORE 1
TASK 1(1)  (2 , 30) Utilization : 0.067 priority: 6 at : CORE 0
TASK 6(1)  (12 , 100) Utilization : 0.120 priority: 1 at : CORE 1
TASK 2(1)  (3 , 30) Utilization : 0.100 priority: 5 at : CORE 0
TASK 3(1)  (8 , 35) Utilization : 0.229 priority: 4 at : CORE 0
TASK 4(2)  (6 , 50) Utilization : 0.111 priority: 3 at : CORE 0
TASK 5(1)  (12 , 100) Utilization : 0.120 priority: 2 at : CORE 0
```

이를 보드 위에서 SPA1으로 파티셔닝 한 후

```
30 : TASK 1(1) execute on Core 0 Deadline : 60
31: Complete TASK 4(1)  (< 50)
31 : TASK 0(1) execute on Core 0 Deadline : 60
31 : TASK 6(1) execute on Core 1 Deadline : 100
32 : TASK 2(1) execute on Core 0 Deadline : 60
36: Complete TASK 1(1)  (< 60)
36 : TASK 3(1) execute on Core 0 Deadline : 70
40: Complete TASK 0(1)  (< 60)
40: Complete TASK 2(1)  (< 60)
43: Complete TASK 6(1)  (< 100)
46: Complete TASK 3(1)  (< 70)
46 : TASK 4(2) execute on Core 0 Deadline : 50
46: OVERFLOW TASK 4(2)  at Core 0
```

실제로 스케줄링하는 경우 tick 46에서 오버플로우 발생

- **SPA2** 알고리즘



SPA2를 통해 파티셔닝하는 경우 Lemma5식을 만족

```
For 7 tasks Utilization Bound is :  0.7286
Core 0 Utilization : 0.7286, Core 1 Utilization : 0.7266
MONITOR ACTIVATED LCM: 2100
TASK 0(1) (2 , 30) Utilization : 0.093 priority: 7 at : CORE 0
TASK 0(2) (1 , 30) Utilization : 0.007 priority: 7 at : CORE 1
TASK 1(1) (2 , 30) Utilization : 0.067 priority: 6 at : CORE 0
TASK 4(1) (36 , 50) Utilization : 0.720 priority: 3 at : CORE 1
TASK 2(1) (3 , 30) Utilization : 0.100 priority: 5 at : CORE 0
TASK 3(1) (8 , 35) Utilization : 0.229 priority: 4 at : CORE 0
TASK 5(1) (12 , 100) Utilization : 0.120 priority: 2 at : CORE 0
TASK 6(1) (12 , 100) Utilization : 0.120 priority: 1 at : CORE 0
```

이를 보드 위에서 SPA2로 파티셔닝 한 결과

```
2046: Complete TASK 0(1) (< 2070)
2046 : TASK 0(2) execute on Core 1 Deadline : 2070
2046: Complete TASK 1(1) (< 2070)
2047: Complete TASK 0(2) (< 2070)
2047: Complete TASK 2(1) (< 2070)
2049: Complete TASK 6(1) (< 2100)
2050 : TASK 4(1) execute on Core 1 Deadline : 2100
2065 : TASK 3(1) execute on Core 0 Deadline : 2100
2070: TASK 0(2) Suspended
2070 : TASK 0(1) execute on Core 0 Deadline : 2100
2071 : TASK 1(1) execute on Core 0 Deadline : 2100
2072 : TASK 2(1) execute on Core 0 Deadline : 2100
2078: Complete TASK 0(1) (< 2100)
2078 : TASK 0(2) execute on Core 1 Deadline : 2100
2078: Complete TASK 1(1) (< 2100)
2079: Complete TASK 0(2) (< 2100)
2080: Complete TASK 2(1) (< 2100)
2080: Complete TASK 3(1) (< 2100)
2086: Complete TASK 4(1) (< 2100)
2100: TASK 0(2) Suspended
2100 : TASK 0(1) execute on Core 0 Deadline : 2130
2100 : TASK 4(1) execute on Core 1 Deadline : 2150
ALL TASKS SCHEDULABLE
```

이를 모든 주기들의 최소 공배수까지 스케줄링했을 때 오버플로우가 발생하지 않았음을 확인할 수 있음

- 결론

SPA2의 Simple test의 필요성

- 모든 Heavy Task를 Pre-assign했을 때 Light Task의 우선 순위를 밀리게 하여 Scheduling을 보장할 수 없게 만들 수 있음
- 따라서 Heavy Task 중 다른 Tail Subtask의 Deadline miss를 야기하지 않을 Task만 Pre-assign 하는 것이 중요
- 다음과 같은 Simple test를 통해 위에서 언급한 Heavy Task를 선별

$$\textbf{if } \tau_i \text{ is heavy } \wedge \sum_{j>i} U_j \leq (|PQ| - 1) \cdot \Theta(N) \textbf{ then}$$

1차 구현을 통해서는 위의 Simple test를 사용하여 파티셔닝 했을 때 쪼개진 Task가 Lemma5의 식을 만족함을 확인해 봄

Lemma 5. *Suppose a tail subtask τ_i^t is assigned to processor P_t. If τ_i^t satisfies*

$$Y^t \cdot T_i / \triangle_i^t + V_i^t \leq \Theta(N), \qquad (7)$$

then τ_i^t can meet its deadline.

위의 구현을 통해서는 Lemma5의 식을 만족할 때 실제 스케줄링이 가능함을 확인해보았음.