

# Dining Philosopher In Spin

---

김대희, 한기홍



01

이전 시간의 SPIN의 문제점

02

Dining Philosophers Problem

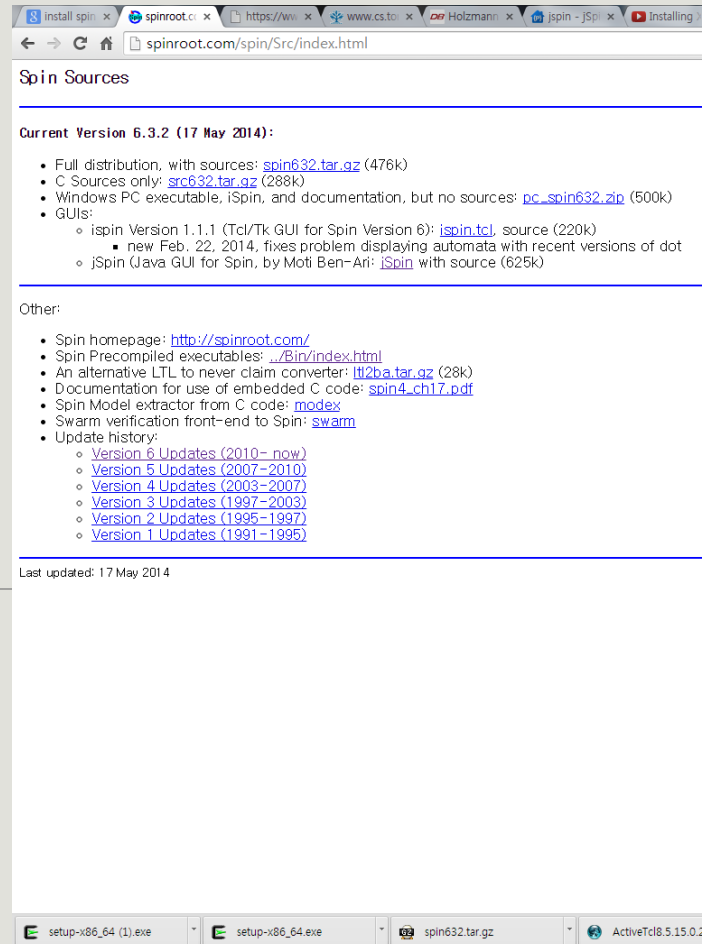
03

Dining Philosophers Problem Solution

---

# 1. 이전 시간의 SPIN의 문제점

2014.09.20 download v6.3.2



2014.09.20일 이후 download v6.4.0



# 1. 이전 시간의 SPIN의 문제점

---

결론

- 6.4.0 버전 spin에서 문제 발생. 곧바로 6.4.2 버전 나옴. (한 달에 2번씩 업데이트 됨.)
  - 6.3.2 버전 spin은 잘 작동함.
  - 우연의 일치!!
- 



## 2. Dining Philosopher Problem

---

properties of the system to verify

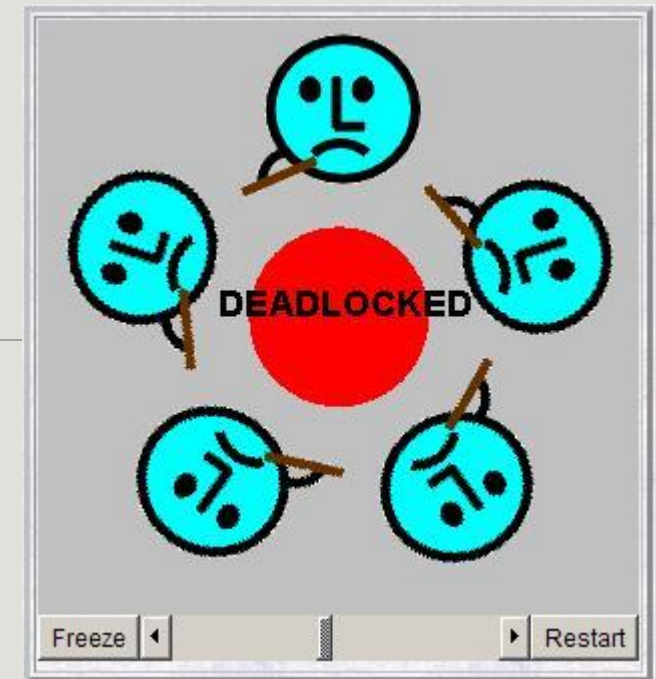
1. 왼쪽 포크를 집은 후에 오른쪽 포크를 집는다.
  2. 양쪽 포크를 집을 시에 식사가 가능.
  3. 식사 후에는 양 포크를 내려 놓는다.
  4. 5명의 철학자 5개의 포크가 있다.
- 



## 2. Dining Philosopher Problem

```
관리자: C:\Windows\system32\cmd.exe
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>spin DiningPhilosHoper.pml
                mypid is "4" waiting.!!
        mypid is "1" waiting.!!
mypid is "0" waiting.!!
                mypid is "2" waiting.!!
        mypid is "1" waiting.!!
mypid is "0" waiting.!!
5 processes created
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>
```

모두 대기상태



# 2. Dining Philosopher Problem

wait ?

The screenshot displays the Spin 6.3.2 IDE interface. On the left, the source code for the Dining Philosopher Problem is shown. The code defines 5 philosophers and 4 channels, and implements a process for each philosopher that can eat or wait based on the state of the channels. A red line highlights the condition `:: ( eatAndWaiting[_pid] == 0 ) ->` in line 15. On the right, the Automata View shows a state transition graph with nodes labeled S0 through S17. The graph illustrates the various states of the system, including states where philosophers are eating, waiting, or holding forks. The initial state is S11, and the graph shows transitions between states based on the actions of the philosophers. At the bottom, the command history shows the execution of the `select` command and the results of syntax and spin checks.

```
1 #define NUM_PHILOSOPHER 5
2 #define CHANNEL_SIZE 4
3
4 chan wwwwwwww = [CHANNEL_SIZE] of { byte, int, short };
5
6 bool fork[NUM_PHILOSOPHER];
7 bool eatAndWaiting[NUM_PHILOSOPHER]
8
9 active [NUM_PHILOSOPHER] proctype philosopher()
10 {
11     eatAndWaiting[_pid] = 0;
12     fork[_pid] = 0;
13
14     do
15     :: ( eatAndWaiting[_pid] == 0 ) ->
16         printf("mypid is \"%d\" waiting!!\n", _pid);
17         if
18         :: ( fork[_pid] == 0 ) -> fork[_pid] = 1
19         :: ( fork[NUM_PHILOSOPHER-1-_pid] == 0 ) ->
20             fork[NUM_PHILOSOPHER-1-_pid] = 1
21         :: ( fork[_pid]==1 && fork[NUM_PHILOSOPHER-1-_pid]==1 ) ->
22             eatAndWaiting[_pid] = 1
23         fi
24     :: else ->
25         printf("mypid is \"%d\" eating!!\n", _pid);
26         fork[_pid] = 0;
27         fork[NUM_PHILOSOPHER-1-_pid] = 0;
28         eatAndWaiting[_pid] = 0
29     fi;
30     :: break;
31     od
32 }
33
34
35
```

o select p\_philosopher
9 <saved DiningPhilosHoper.pml>
10 DiningPhilosHoper.pml: 1
11 syntax check
spin: nothing to report
12 syntax check
spin: nothing to report
13 syntax check
spin: nothing to report
14 syntax check
spin: nothing to report
15 syntax check
spin: nothing to report
16 <saved DiningPhilosHoper.pml>
17 DiningPhilosHoper.pml: 1
18 syntax check
spin: nothing to report



# 3. Dining Philosopher Problem Solution

---

properties of the system to verify

1. 웨이터에게 식사가 가능한지 물어본다. 가능하다면 양옆의 포크를 사용하여 식사한다.
  2. 식사 후에는 양 포크를 내려 놓는다.
  3. 5명의 철학자 5개의 포크가 있다.
- 

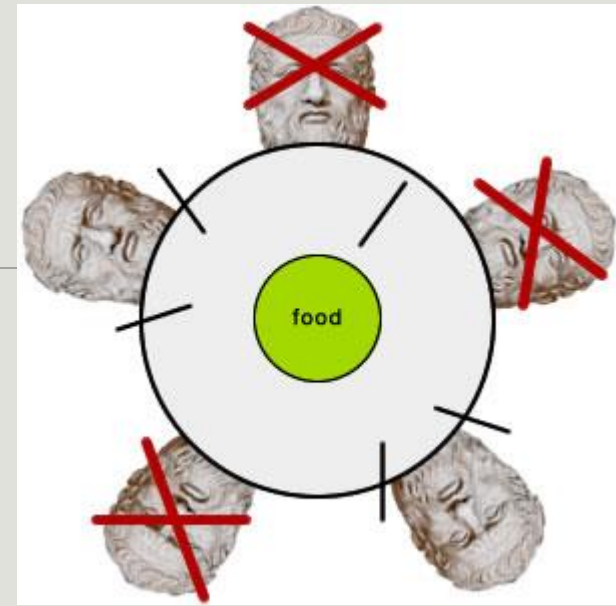




# 3. Dining Philosopher Problem Solution

```
관리자: C:\Windows\system32\cmd.exe
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>spin DiningPhilosHoper.pml
  mypid is "0" eating.!!
        mypid is "3" waiting.!!
          mypid is "2" waiting.!!
            mypid is "1" waiting.!!
              mypid is "1" waiting.!!
                mypid is "1" eat.!!
9 processes created
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>
C:\Users\daehee\Desktop>
```

'0'과 '1'이 먹음



# 3. Dining Philosopher Problem Solution

Send!!

```
2 #define NUM_PHILOSOPHER 5
3 #define CHANNEL_SIZE 100
4
5 chan eatingPhilosopher = [CHANNEL_SIZE] of { int };
6
7
8
9
10
11
12
13 bool fork[NUM_PHILOSOPHER];
14 bool eatAndWaiting[NUM_PHILOSOPHER]
15
16
17
18
19 proctype waiter(int id)
20 {
21
22     if
23     :: ( eatAndWaiting[id] == 0 )-> printf("mypid is \"%d\" waiting!!\n", id);
24     if
25     :: ( fork[id] == 0 )-> fork[id] = 1
26     :: ( fork[NUM_PHILOSOPHER-1-id] == 0 )-> fork[NUM_PHILOSOPHER-1-id] = 1
27     :: ( fork[id]==1 && fork[NUM_PHILOSOPHER-1-id]==1 )->
28         eatAndWaiting[id] = 1;
29         printf("mypid is \"%d\" eat!!\n", id)
30         eatAndWaiting[id] = 0;
31         eatingPhilosopher[id]
32     :: else-> printf("mypid is \"%d\" eating!!\n", id);
33         fork[id] = 0;
34         fork[NUM_PHILOSOPHER-1-id] = 0;
35         eatAndWaiting[id] = 0;
36     fi
37 }
38
39
40
41
42
43
44
45
46 active [NUM_PHILOSOPHER] proctype philosopher()
47 {
48     int ppid;
49
50     eatAndWaiting[_pid] = 0;
51     fork[_pid] = 0;
52
53 do
54 ::   run waiter(_pid);
55 ::   eatingPhilosopher ?ppid;
56 ::   if
57 ::   :: ( ppid == _pid )-> printf("mypid is \"%d\" eating!!\n", _pid);
58 ::   fi
59 ::   if
60 ::   :: ( eatAndWaiting[ppid] == 0 )-> break;
61 ::   fi
62 od
63 }
64
65
66 spin -o3 -a DiningPhilosHoper.pml
67 C:/MinGW/bin/gcc -o pan pan.c
68 ./pan -D > dot.tmp
69 select p_philosopher
```

The Automata View on the right shows a state transition graph with states S0 through S16. Transitions are labeled with actions like `eatAndWaiting[_pid] = 0`, `fork[_pid] = 0`, and `run waiter(_pid)`. State S13 is highlighted with a red circle, and a red arrow points from the `Send!!` callout to the `eatAndWaiting[id] = 0;` line in the code.



# 3. Dining Philosopher Problem Solution

Recv.!!

```
2 #define NUM_PHILOSOPHER 5
3 #define CHANNEL_SIZE 100
4
5 chan eatingPhilosopher = [CHANNEL_SIZE] of { int };
6
7
8
9
10
11
12
13 bool fork[NUM_PHILOSOPHER];
14 bool eatAndWaiting[NUM_PHILOSOPHER]
15
16
17
18
19 proctype waiter(int id)
20 {
21
22     if
23     :: ( eatAndWaiting[id] == 0 ) -> printf("mypid is \"%d\" waiting.!!\n", id);
24     if
25     :: ( fork[id] == 0 ) -> fork[id] = 1
26     :: ( fork[NUM_PHILOSOPHER-1-id] == 0 ) -> fork[NUM_PHILOSOPHER-1-id] = 1
27     :: ( fork[id]==1 && fork[NUM_PHILOSOPHER-1-id]==1 ) ->
28         eatAndWaiting[id] = 1;
29         printf("mypid is \"%d\" eat.!!\n", id)
30         eatingPhilosopher !id
31     fi
32     :: else -> printf("mypid is \"%d\" eating.!!\n", id);
33     fork[id] = 0;
34     fork[NUM_PHILOSOPHER-1-id] = 0;
35     eatAndWaiting[id] = 0;
36     fi
37 }
38
39 active [NUM_PHILOSOPHER] proctype philosopher()
40 {
41     int ppid;
42
43     eatAndWaiting[_pid] = 0;
44     fork[_pid] = 0;
45
46     do
47     :: run waiter(_pid);
48     :: eatingPhilosopher ?ppid;
49     :: if
50     :: ( ppid == _pid ) -> printf("mypid is \"%d\" eating.!!\n", _pid);
51     fi
52     :: if
53     :: ( eatAndWaiting[ppid] == 0 ) -> break;
54     fi
55     od
56 }
57
58
59
60
61
62
63
64
```

70 spin -o3 -a DiningPhilosHoper.pml  
71 C:/MinGW/bin/gcc -o pan pan.c  
72 ./pan -D > dot.tmp  
73 select p\_waiter

The automata view shows states S0 through S20. S18 is the initial state for the waiter. Transitions are labeled with Promela expressions such as `!eatAndWaiting[id]`, `!fork[id]`, and `!fork[5-1-id]`. The final state is S0, labeled as `-end-`.



# Q & A

