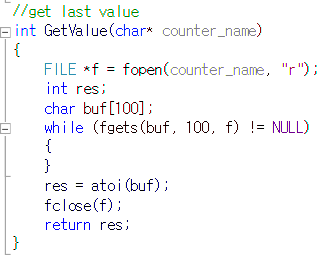
**[Homework](http://os.cs.kookmin.ac.kr/homework-1)**‎ > ‎

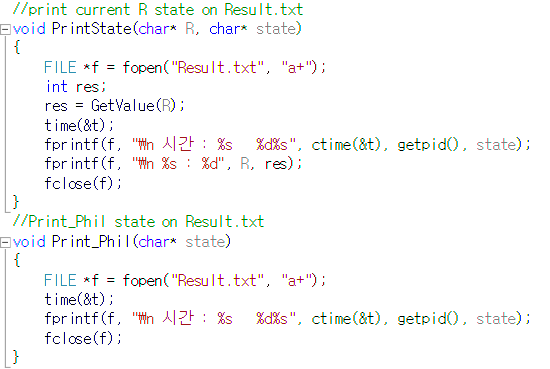
Dining Philosopher

**컴퓨터공학부 20123420 차경준**

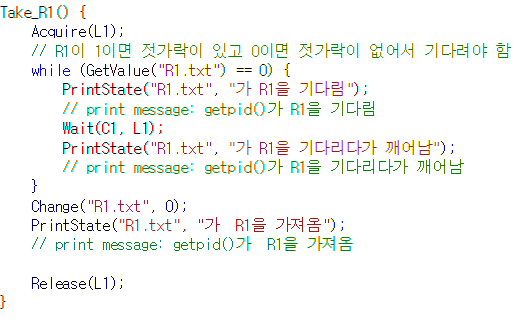
1. Dining Philosopher 문제를 세마포를 이용하여 구현

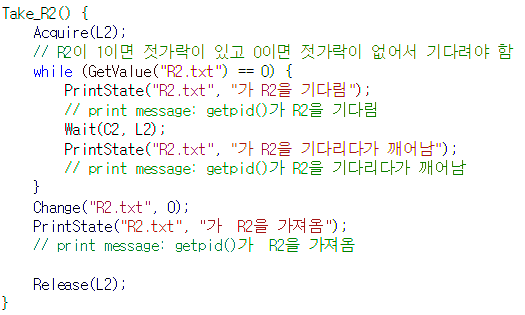
//결과값을 출력하는 함수

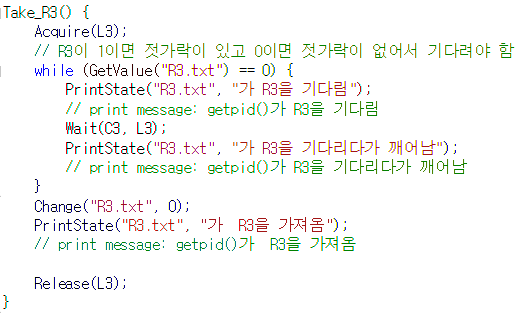




//Take R1, R2, R3



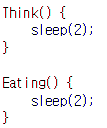




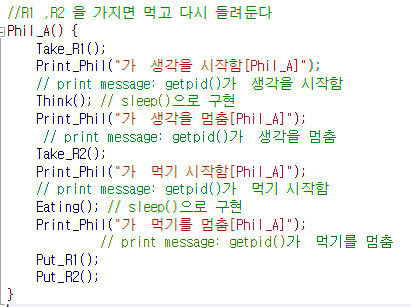
//Put R!, R2, R3

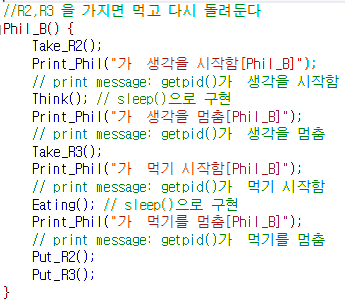


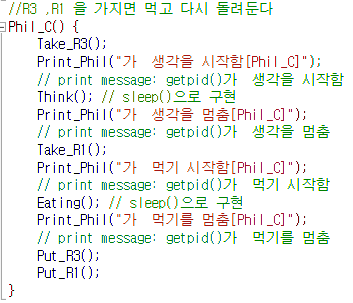
//Think(), Eating() 임의로 sleep(2) 함수를 넣어줬다.



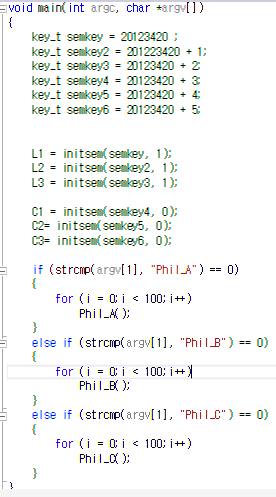
//Philosoper\_A , B , C



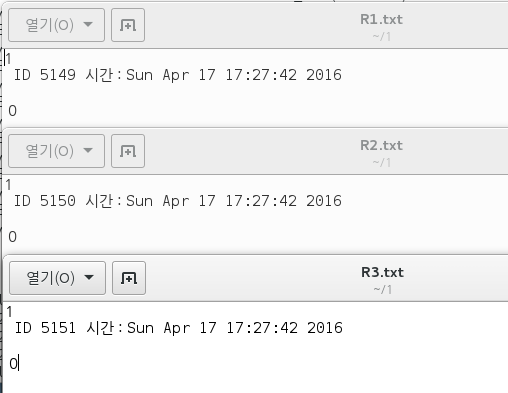




//main



1. Phil\_A, Phil\_B, Phil\_C 를 별도의 프로세스로 실행하여 데드락이 걸리는 상황을 만들고 이 상황의 스냅 샷을 보이고 설명하시오.



5150[B] Take\_R2() -> 5150[B] Thjink() -> 5151[C] Take\_R3() -> 5151[C] Think() ->

5149[A] Take\_R1() -> 5149[A] Think() -> **5150[B] R3 기다림 -> 5151[C] R1 기다림 -> 5149[A] R2 기다림**

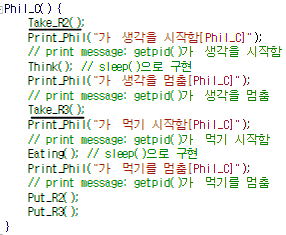
DeadLock이 발생한다.

3.4  Deadlock을 다음 두가지 방법으로 Prevention(예방)하시오. 두가지 종류의 상태를 보여주는 실행 결과를 제시하고 설명하시오.

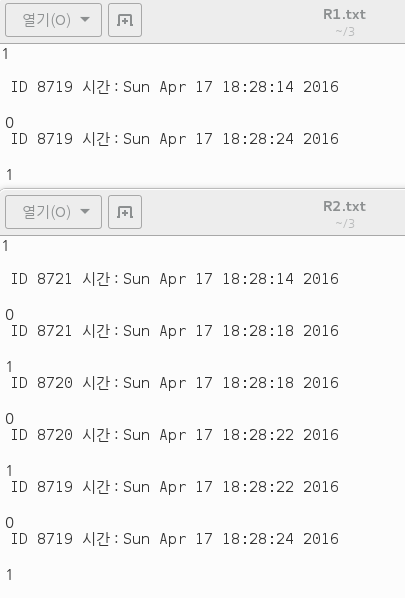
* Circular wait 조건을 깨는 방법

환형 대기조건을 방지하기 위해서는 모든 자원 유형에 대하여 일련번호를 지정하고(위에서는 R1, R2 ,R3), 교착상태 방지를 위해 프로세서는 항상 자원 일련번호 순서(오름차순을 사용한다)으로 요청하도록 한다.

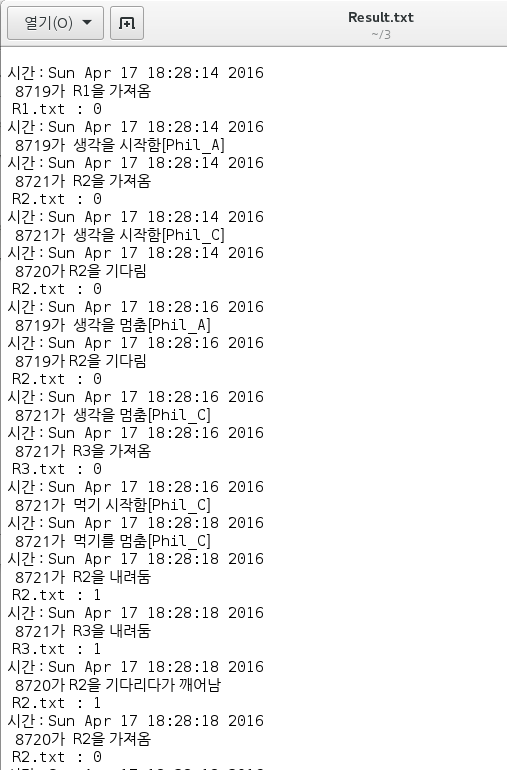
자원 R의 요청 순서를 A : R1 -> R2 B: R2->R3 C: R2 –R3 로 지정하였다.

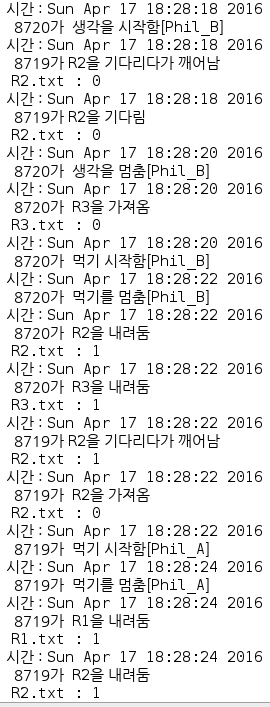
//Phil\_C 를 R3 -> R1 에서 R2 -> R3 로 수정  


//실행 결과



//Result





요약하자면

[A] Take\_R1 (R1 = 0)

-> [A] Think()

-> [C] Take\_R2 (R2 = 0)

-> [C] Think()

->[B] Waiting

->[A] 생각을 멈춤

->[A]가 R2를 기다림

->[C] 생각을 멈춤

->[C] -> Take\_R3() (R3 = 0)

**->[C] Eating**

->[C] Put\_R2() (R2 = 1)

->[C] Put\_R3() (R3 = 1)

->[B]가 R2를 기다리다가 깨어남 (R2 =1)

->[B] Take\_R2() (R2 = 0)

->[B] Think()

->[A]가 R2를 기다리다가 꺠어남 (R2 = 0)

->[A]가 R2를 기다림

->[B] Take\_R3() (R3 = 0)

**->[B] Eating()**

->[B] Put\_R2() (R2 =1)

->[B] Put\_R3() (R3=1)

->[A]가 R2를 기다리다가 꺠어남 (R2 = 1)

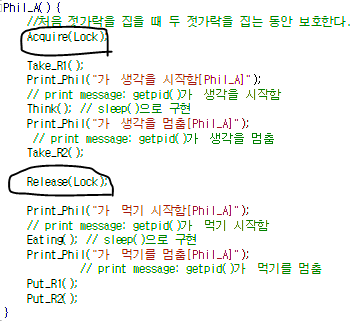
->[A] Take\_R2()

**->[A] Eating()**

->[A] Put\_R1() (R1 = 1)

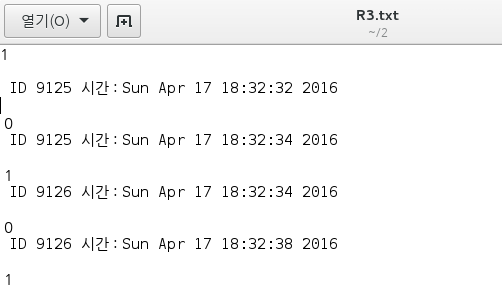
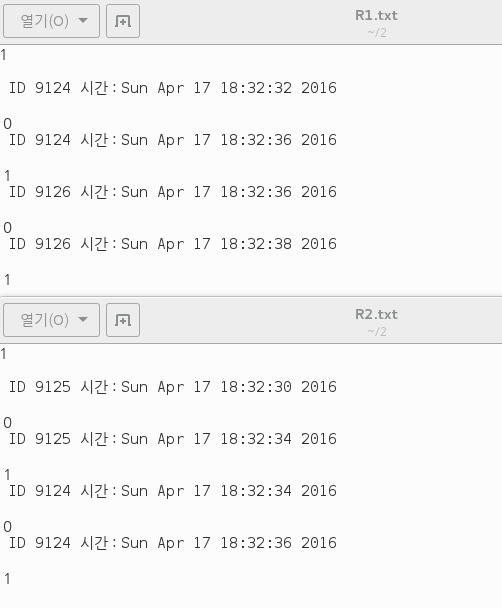
->[A] Put\_R2() (R2 =1)

* 젓가락을 한꺼번에 가져와서 Hold & Wait 조건을 깨는 방법(Lock이 추가로 필요함)

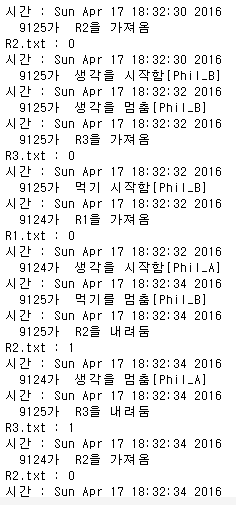


두 R1, R2 를 얻는 과정을 Lock으로 보호함으로 구현한다.

각각 Phil\_B(), Phil\_C()에도 같게 적용시킨다.



`



요약하면

->[B] Take\_R2() (R2 =0)

->[B] Think()

->[B] Take\_R3() (R3=0)

**->[B] Eating()**

->[A] Take\_R1() (R1 =0)

->[A] Think()

->[B] Put\_R2() (R2 =1)

->[B] Put\_R3() (R3 =1)

->[A] Take\_R2() (R2=0)

->[C] Take\_R3() (R3=0)

->[C] Think()

**->[A] Eating()**

->[A] Put\_R1() (R1 =1)

->[A] Put\_R2() (R2=1)

->[C] Take\_R1() (R1=0)

**->[C] Eating()**

->[C] Put\_R3() (R3 =1)

->[C] Put\_R1() (R1=1)

으로 B –A –C 순서대로 데드락 없이 작동한다.