|  |
| --- |
| Atomicity |
| (제공된 코드 중심으로 설명)  먼저 t1이 모두 실행된 후, t2가 실행되는 순서로 진행되어야 한다. 그래서 37번째줄~리턴되기 전 44번째 줄까지 lock한다. 왜 리턴 직전에 lock을 끝내냐면 스레드가 lock을 하고 리턴되어야 하기 때문이다.  (결과는 짠 코드위주로 설명)    결과를 보면 코드 21번째 줄 메인이 시작되면서 결과로 main: begin이 출력된다.  그리고 t2가 실행되면서 56번째줄에서 sleep을 하면서 t1이 lock되는 시간을 갖게 되고, thread1에 42줄~ 50줄까지 t1이 실행되면 끝까지 실행되고 100이 나온다. 그 후 thread2도 마저 실행되면서 57번째줄부터 60번째줄까지 lock이 되어있기에 thread1을 기다린다. 때문에 thread1이 완료되고 lock이 반납되면 thread2이 lock을 가지는 순서를 갖게 된다.  (코드) |
| Ordering |
| (제공된 코드 중심으로 설명)  항상 38번째줄이 17번째줄보다 먼저 실행되어야 하는데, 32번째 sleep에서 오래 기다려서 17번째줄이 먼저 실행되어 thd초기화와 thd역참조와 race condition이 생겨서 문제가 발생한다.  그래서 routine함수의 16, 17번 줄 앞뒤로 lock을 걸어주고, 39번째 줄에도 앞뒤로 lock을 걸어줘서 race condition을 방지한다. 그러면 부모인 main이 먼저 실행되다가 38번째 줄 thd가 먼저 초기화가 완성된 뒤, 15번째줄 routine이 실행되면서 순서가 확실하게 정해진다.  (결과는 짠 코드위주로 설명)    메인에서 45번째줄의 my\_thread\_t타입의 thd가 초기화가 완료가 될때까지 lock을 걸어놓는다. 그러면 결과에서 ordering: begin까지 완료되고, 28번째줄부터 36번째줄까지 완료되어 초기화가 완료된다. 그러면 lock을 반납하고 이때 자식인 routine이 실행되는데 이때에도 lock을 걸어놓아야 main -> routine순서로 진행될 수 있다. routine에서도 리턴 직전까지 lock을 걸어놓고(17번째줄 ~ 20번째줄) 결과로 rountine: begin, routine: stae is 0를 얻을 수 있다. main으로 돌아와서 ordering: end문구까지 출력하면 결과를 얻을 수 있다.  (코드) |
| Deadlock – circular |
| 1. 설명   코드를 보면, thread1이 생성되고, thread1은 L1을 갖는다. 그리고 thread2가 생성되고, thread1은 또 L2를 갖는다. 그러고 난 뒤 L1을 반환하고, thread2가 L2를 요구한다. 여기서 circular가 성립되려면 thread1은 L2를 thread2가 요구할 때 주기위해 L2를 unlock해야한다. 이 때 thread1이 L2를 unlock하는 부분을 lock을 해주면, thread2가 L2를 요구할 때 이를 만족시킬 수 없다. 그래서 thread1이 L2를 unlock하는 부분을 lock해주면, circular이 깨지게 된다.   1. 코드   테이블이(가) 표시된 사진  자동 생성된 설명   1. 결과 |
| Deadlock – circular. |
| 1. 설명   먼저 thread1에서 L1과 L2를 lock이 된다. thread2는 thread1이 L1과 L2를 unlock하면, L2를 lock하고 L1을 차례로 lock할 수 있다. 따라서 thread1에서 L1과 L2를 unlock하는 부분을 무한루프를 돌려서 unlock되지 못하게 한다면 thread2에서도 L2와 L1을 쓸 수 없게된다.   1. 코드   테이블이(가) 표시된 사진  자동 생성된 설명   1. 결과텍스트이(가) 표시된 사진     자동 생성된 설명 |
| Deadlock – no preemption : 할당된 자원은 사용이 끝날때까지 강제로 뺏을 수 없다. |
| 1. 설명   먼저 thread2가 L2를 점유하고 있는데 이 L2를 thread1도 점유 하려 한다면, 데드락이 발생한다. 그러므로 thread2가 L2를 unlock할때까지 thread1은 대기한다. |
| 느낀점 |
| 너무 어렵습니다… |