강의명: 실시간커널

실습 번호: 2

실습 제목: Mutual exclusion(상호 배제)

학생 이름: 김준호

학번: 201710860

1 상호 배제 문제 분석

1.1



Text

Description automatically generated

task1, 2는 각각 priority 20, 21을 갖는 task들이며 각각 1과 2를 여러번 stdout으로 출력하는 코드이다.

각 task들은 OS\_GetRandom(5) ticks만큼 WAIT하였다가 수행되는데 OS\_GetRandom은 0 부터 4까지의 자연수를 return한다.

해당 task들의 수행결과는 아래와 같다.

Graphical user interface

Description automatically generated

중간중간 task2가 수행되다 말고 task1이 수행되어 2뒤로 1이 출력되는 원하지 않는 결과를 나타내게 된다.

1.2

이러한 결과는 task1, task2의 OSTimeDly에 인자로 들어가는 시간에 따라 발생할 수도 있고 아닐 수도 있다.

관찰 결과 task2는 중간에 preempt 되고 task1이 출력을 마치면 나머지를 출력하는 것처럼 보인다. 그러면 task2가 수행되다가 task1이 수행되려면 첫 번째로 task1, task2를 각각 수행하는데 합쳐서 1tick이 넘게 걸리는 경우를 생각해볼 수 있다.

그러나 이는 어쩌다가는 task1, task2가 문제 없이 번갈아가며 수행되는 결과를 설명하지 못한다. 즉 task1, task2 둘 다 수행되는데 1tick이 걸리지 않는다는 의미이다. deadline을 지킨다는 의미로도 해석할 수 있다.

그렇다면 둘 다 OSTimeDly 에 인자로 0 이 들어가면 어떤 결과를 얻을까? 코드를 수정하여 수행해보았다. 코드와 수행결과가 각각 아래와 같다.

Text

Description automatically generated

Background pattern

Description automatically generated

수행 결과로 task1만 수행되는 것을 알 수 있다. 왜 그런가 살펴보기 위해 OSTimeDly가 내부적으로 어떻게 동작하는지 코드를 확인할 필요가 있다.

OSTimeDly의 코드는 아래와 같다.

Graphical user interface, text

Description automatically generated

line 61-71을 살펴보면 인자로 들어가는 tick이 0 초과일 때만 Scheduler를 호출하여 READY STATE로 들어가는 것을 알 수 있고 ticksrk 0인 경우 정말 아무것도 하지 않는 것을 알 수 있다. 즉, OSTimeDly(0)은 아무것도 수행하지 않는 ; 세미콜론과 같다.

따라서 priority가 가장 높은 task1이 READY STATE가 되지 않으므로 task2가 수행될 턱이 없는 것이다.

따라서 아래와 같이 결론을 지을 수 있다.

1. 위와 같은 현상은 task2가 쉬지 않고 수행되다가 task1에 의해 preempt될 때 발생한다.  
 즉, task2의 OSTimeDly의 인자는 0이다.

2. task1의 OSTimeDly의 인자가 0이 들어가면 task2는 수행되지 않으므로 task1의 OSTimeDly의 인자는 1 이상의 값이어야한다.

위 1, 2 의 조건을 만족하면 task1, task2의 출력물이 겹치는 현상이 발생한다.

1.3

task2의 OSTimeDly에 상관없이 task1의 OSTimeDly의 인자가 0이라면 항상 task1만 수행되므로 task2에 의해 preemt 될 일이 없다.

task1의 OSTimeDly가 1이상 이면 매 tick ISR이 수행될 때마다 user task중 priority가 가장 높은 task1 의 실행 여부를 가장 먼저 따질 것이기 때문에 task2를 preempt하면 preemt했지 task1이 preempt 될 일은 없다.

2 상호 배제 문제 해결 (인터럽트)

2.1

인터럽트를 disalble, enable하는 macro는 Ports/os\_cpu.h 에 정의 돼있는

OS\_ENTER\_CRITICAL(), OS\_EXIT\_CRITICAL()을 사용할 수 있다.

이때 interrupt을 disable하는 방법에는 3가지 방법이 있다. 이는 아래와 같다.

1. disable/enable using simple instructions

이 방법은 interrupt가 disable, enable 된 후의 상태를 보장할 수 없기 때문에 사용하면 안된다.

2. 위 문제점을 해결하기 위해 interrupt 을 disable, enable한 상태를 stack에 저장한ㄷ.

하지만 보통 critical section에서 SP register의 값이 바뀌므로 좋은 방법이 아니다.

3. 3번의 방법으로 구현하며, interrupt의 disable, enable 상태를 local variable에 저장하는 방법이다.

Ports/os\_cpu.h 에는 해당 방법으로

OS\_ENTER\_CRITICAL()

OS\_EXIT\_CRITICAL()

이 정의되어 있다. 실제 코드는 아래와 같다.

Text

Description automatically generated with low confidence

여기서 local variable은 cpu\_sr에 해당한다.

define 문은 해당 코드를 뒤에 정의된 코드로 말그대로 대치하는 것이다

따라서 각 task안에 cpu\_sr을 선언해줘야하는데 이의 type은 Ports/os\_cpu.h 의 아래쪽에 가면 아래와 같이 prototype이 선언된걸 볼 수 있다.

Text

Description automatically generated with low confidence

OS\_CPU\_SR\_Save() 의 return 값으로 OS\_CPU\_SR type 을 사용한다.

정리하자면,

interrupt를 disable, enable하여 사용하는 방법은 해당 상태를 local variable에 저장하는 방법을 사용한다.

이는 Ports/os\_cpu.h 에 OS\_ENTER\_CRITICAL(), OS\_EXIT\_CRITICAL() macro로 정의 돼있으며, 우리는 macro에서 사용할 local variable만 선언해주면 되는데 이는 아래와 같이 선언한다.

OS\_CPU\_SR cpu\_sr;

따라서 실습으로 돌아가 위 매크로를 사용하면 코드는 아래와 같다.

Text

Description automatically generated

2.2

Background pattern

Description automatically generated

실습 1과 비교하였을 때 적어도 보이는 출력에서는 문제가 발생하지 않는 것을 알 수 있다.

3 상호 배제 문제 해결 (스케줄러)

3.1

Scheduler를 lock, unlock하는 것은 uC/OS-II 에서 쉽게 사용할 수 있도록 정의해둔 함수를 그저 가져가 사용하기만 하면 된다.

사용 방법은 아래와 같다.

Graphical user interface, application

Description automatically generated

따라서 우리의 task에서 출력하고자 하는 UARTprintf을 …에 넣을 것이다.

구현한 코드는 아래와 같다.

Text

Description automatically generated

3.2

A picture containing text, grass

Description automatically generated

역시 문제 없이 잘 출력되는 것을 확인할 수 있다.

4 상호 배제 문제 해결 (세마포어)

4.1

semaphore를 사용하기 위해선 첫 째로 먼저 semaphore를 생성해야한다.

이는 OSSemCreate() 함수를 호출하여 생성할 수 있다. 이때 처음 semaphore값을 셋팅하는 인자가 중요한데 이를 살펴보기 위해 아래 수업 중 ppt내용을 살펴보자

Text

Description automatically generated

semaphore는 sem이라는 변수가 공유 변수로서 사용되고 이 값이 0을 벗어나 1이 되기 전까지는 PEND 상태에서 벗어날 수 없다.

이때 OSSemCreate(0) 은 저 sem 값을 0으로 초기화한다는 의미이고 OSSemCreate(1)은 sem값을 1로 초기화 한다는 의미이다. 0으로 초기화하면 while문에 갖힌 상태로 시작하는 것이고, 1로 초기화하면 while문을 탈출하면서 시작하는 것이다.

이때 세마포의 역할을 생각해보아야한다. 세마포를 2개를 사용한다고 가정하자.

세마포1을 Task1이 Post하고, 세마포2를 Task2 가 Post한다고 생각하면

이들은 서로 OSTimeDly에 관계없이 Task1, Task2, Task1, Task2와 같이 수행될 것이다.

이는 우리과 원하지 않는 수행 결과이다.

이를 해결하기 위해 세마포1를 공유하여 Pend, Post하는 Task들을 생각해볼 수 있다.

그러면 하나의 세마포를 공유하는 Task들 간에는 어떤 Task에서 세마포가 Post되었을 때, READY State에 있는 Task중 priority가 가장 높은 Task가 다음 세마포를 가지게 된다. 그러나 priority가 더 낮더라도 다른 Task가 가지고 있는 세마포는 뺏어올 수 없다.

이 점을 이용하여 코드를 아래와 같이 작성하였다.

Text

Description automatically generated

세마포를 하나만 선언하였고 이 sem1을 Task1, Task2가 공유한다.

코드는 아래와 같다.

Text

Description automatically generated

4.2

A picture containing background pattern

Description automatically generated

잘 수행되는 것을 확인할 수 있다.

끝.