강의명: 실시간커널

실습 번호: 1

실습 제목: Milti tasks(멀티 테스크)

학생 이름: 김준호

학번: 201710860

1 Simple multi tasks

1.1

Text

Description automatically generated

OSTaskCreate의 첫 번째 인자는 Task로 실행할 함수의 address,

두 번째 인자는 task에 pdata로 전달할 data, 세 번째 인자는 사용할 스택의 주소(이는 스택포인터 처럼 사용될 것이다.) 네 번째 인자는 priority이다.

실습에서 생성해야할 Tasks는 총 7개이고, 그 중 하나는 main에서 생성하였고 나머지 TaskTime, Task1,…,Task5까지 직접 생성해야한다.

각 생성하는 코드에서 첫 번째 인자들은 모두 뒤에서 정의할 함수들이다.

두 번째로는 task에 전달할 data인데 이는 사용하지 않으므로 NULL을 취하는 것이다.

Stack은 push하면 할 수록 address가 감소한다. 따라서 stack의 가장 마지막 주소를 전달하여야 하므로 TaskStk들의 마지막 주소를 전달한 것이다.

마지막 네 번째 인자는 priority이다. 이는 실습 조건대로 priority를 부여하였다.

Text

Description automatically generatedText

Description automatically generated

위에서 Tasks를 생성할 때 첫 번째 인자로 주었던 함수들이다.

반복해서 실행되어야하므로 무한 루프안에서 로직을 수행하였다.

TaskTime은 1초 마다 Time=1, Time=2…와 같이 출력하여야한다.

OSTimeDly는 인자를 Tick 단위로 받는 Wait함수이다. 이때 1초는 100ticks 이므로 TaskTime 함수 내에 있는 OSTimeDly함수에 100을 인자로 주었다.

OSTimeDly 함수를 쓰는 이유는 Wait 기능을 쉽게 사용하기 위함도 있지만,

Diagram

Description automatically generated

메뉴얼에 나와있는 것과 같이 OSTimeDly를 호출한 task가 priority가 자신 보다 낮은 task에게 CPU를 양보하기 위해 사용된다.

OSTimeDly를 사용하지 않고 if문으로 제어하면 Tasktime말고는 수행되지 않는다.

Task1, 2, 3, 4, 5는 각각 1초, 2초, 3초, 4초, 5초 씩을 기다렸다가 각 Task1, 2, 3, 4, 5를 출력한다. 그렇기에 각각 OSTimeDly(100), OSTimeDly(200), …, OSTimeDly(500)와 같이 사용하였다.

이들도 역시 각자의 TaskN(1 <= N <= 5) 을 출력하고는 다음에 출력할 Task가 있다면 양보하여야하므로 OSTimeDly를 TaskTime이 사용한 이유와 같은 이유로 사용하였다.

1.2

Text

Description automatically generated

1.3

각 TaskN들은 N의 배수이므로 출력된 Time의 약수인 N들의 TaskN들이 출력됐다면 프로그램이 바르게 수행된 것이다.

Time=1 부터 Time=19까지 모두 문제없이 바르게 출력 되었으므로 바르게 수행되었다.

1.4

위에서 설명한 바와 같이 OSTimeDly가 수행되면 현재 수행되고 있던 Task는 WAIT 상태가 되었다가 시간이 모두 지나면 READY 상태가 된다. 이때 CPU는 다음 Task를 READY 상태에 놓은 Task들 중에서 찾게 되는데 이때 가장 높은 priority를 갖는 Task를 실행하게 된다. 따라서 각 라인에서 모든 Task들이 READY 상태에 있다면 실행될 Task들의 순서는 아래와 같다.

TaskTime -> Task1 -> Task2 -> Task3 -> Task4 -> Task5

(왜냐하면 아래의 Task들의 priority가 각각 20, 21, 22, 23, 24, 25 이기 때문이다.

priority 값은 작을 수록 priority가 높다.)

이때 Time=10 이라면 READY 상태에 있는 Task들은

TaskTime(1초를 10번 기다리고 끝남), Task1(1초를 10번 기다리고 끝남), Task2(2초를 5번 기다리고 끝남), Task5(5초를 2번 기다리고 끝남) 이다.

나머지는 WAIT 상태로 있을 것이다; Task3(3초를 3번 기다렸고 4번째를 기다리는 중 1초가 지났다.), Task4(4초를 2번 기다렸고, 3번째를 기다리는중 2초가 지났다.)

따라서 위에서 설명한 순서대로,

TaskTime, Task1, Task2, Task5가 실행된다.

1.5

1, 2, 3, 4, 5의 최소 공배수는 60 이므로 Time=60일 때 최초로 모든 Tasks가 출력된다.

Graphical user interface, text, application

Description automatically generated

1.6

A picture containing text, whiteboard

Description automatically generated

1.7

저번 학기 MicroController 강의를 수강하며 PBL로

“각 Task들은 주어진 시간만큼 기다렸다가 자신의 차례가 오면 실행한다. 이때 context switching을 한다.” 이를 구현하는 것이었다.

OS에 대한 지식도 없었으며, sleep을 위해서는 task들이 while문에서 기다려야한다는 사실도 알지 못했기 때문에 정말 많은 시행 착오와 실패 뒤에 겨우 엉거주춤하게 돌아가는 프로그램을 작성하였다.

위의 기능을 코드 단 한 줄 OSTimeDly()로 할 수 있다는 것은 user 입장에서는 아주 짜릿한 일이 아닐 수 없다.

실시간 멀티 태스킹 프로그래밍이 간단해지는 이유는 위와 같은 한 줄로 여러 내부적인 기능이 있기 때문이다.

1. 현재 수행중인 Task는 OSTimeDly 실행과 동시에 RUN state -> WAIT state로 state가 바뀐다. 이를 user가 관여하지 않는다.

2. 유저는 각 Task가 주어진 시간만큼 기다렸다가 해당 시간이 끝났는지 아닌지를 확인할 필요가 없다.

3. 주어진 시간 만큼 wait한 뒤 다시 READY state로 돌아왔을 때 CPU가 다음엔 어떤 Task를 수행할지 user가 관여하지 않아도 된다.

4. 코드가 짧고 복잡하지 않아서 유지보수에도 큰 도움이 된다.

즉, user는 쉽게 “주어진 시간만큼 기다렸다가 각자의 차례가 오면 실행해” 를 단 한 줄의 간단한 로직으로 수행할 수 있다.

그렇기에 프로그래밍이 쉬워지는 것이다.

2 Multi tasks의 priority

2.1

위에서 설명하였던 바와 같이 CPU는 OSTimeDly가 실행되면 현재 수행되고 있던 Task를 Wait 상태로 바꾸고 READY 상태인 그 다음 가장 priority가 높은 task를 수행한다.

실습 참고 자료에 제시된 수행 화면처럼 Task5, Task4, Task3, Task2, Task1 처럼 역순으로 실행되려면 단지 priority가 Task5가 가장 높고, 그 다음은 순서대로 작아지면 된다.

priority는 값이 작을 수록 priority가 높아지기 때문에 아래와 같이 priority를 인자로 주어

OSTaskCreate함수를 수행하면된다.

Text

Description automatically generated

2.2

Text

Description automatically generated

2.3

A picture containing text, whiteboard

Description automatically generated

2.4

이는 preemtive kernel이 다음으로 수행될 Task는 READY state에 있는 priority가 그 중 가장 높은 task가 실행되는 것을 보장하기 때문이다.

따라서 순서가 바뀌더라도 각 Task 를 정의한 함수에서 OSTimeDly가 호출되는 것은 모두 동일하기 때문이다.

3 Multi tasks 응용

3.1

Text

Description automatically generated

Task0 – Task9까지 하는 일이 모두 동일하는 것에서 Task함수들을 모두 하드코딩하는 것은 효율적이지 않다.

TaskN (1 <= N <= 9)는 화면에 정해진 바운더리 안에 각자의 Task 번호를 랜덤한 위치에 랜덤한 색깔로 출력하는 기능을 수행한다. 따라서 해당 Task는 각 Task의 번호인 N을 제외하고는 모두 공유할 수 있고, 코드는 아래와 같이 작성하였다.

Text

Description automatically generated

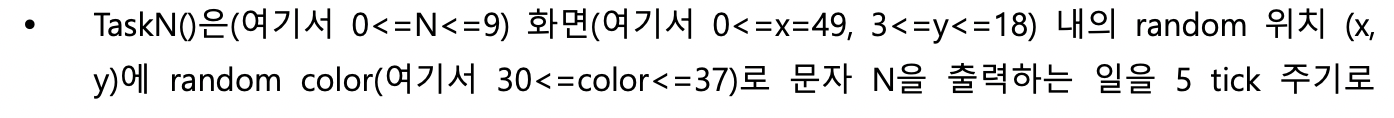
* /home/rk/uCOS-II-2.92.10- tm4c1294xl/Lib/tm4c1294xl/tm4c1294xl.h

위 경로에서 OS\_GetRandom 함수의 프로토타입을 찾아볼 수 있으며 아래와 같다.

Text

Description automatically generated

x, y, color의 범위는 실습 자료에 아래와 같이 주어진다.



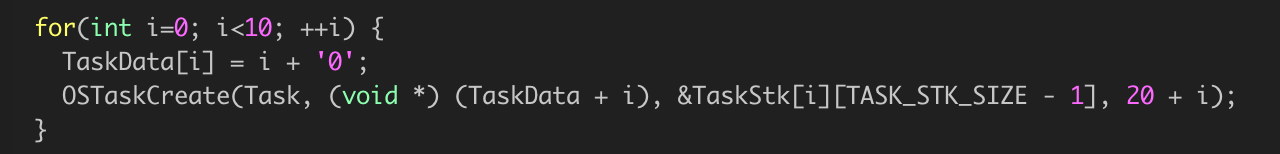
따라서 line 82-84 를 위와 같이 작성하였고, line 86-87에서 랜덤한 위치와, 랜덤한 색깔까지 정해 주었다.

이때 각 TaskN의 N을 출력해야하는데 이것은 OSTaskCreate함수를 호출하여 해당 Task를 생성할 때 인자로 해당 Task의 N을 전달한 주소이다. 이는 void 포인터로 전달 되므로 char 포인터로 casting하여 이 주소의 값을 참조하였다.

OSTaskCreate을 한 코드는 아래와 같다.

A picture containing shape

Description automatically generated



for문을 돌면서 전역 변수로 선언된 TaskData Array에 TaskN의 N에 해당하는 index에 각 N을 저장한다.

OSTaskCreate 함수의 첫 번째 인자로 공유할 함수의 주소, 두 번째 인자로 각 TaskN의 N이 저장된 TaskData array의 index N의 주소(인자로 pdata를 전달할 때는 꼭 void\* 로 전달하여야한다.), index N의 Stack의 역할을 해줄 스택 포인터의 주소, 그리고 Task0, 1, .., 9 순서대로 priority는 20, 21, 22, 29 로 지정해준다.

Text

Description automatically generated

line 56, 63:

Real-time kernel random tasks와 그 아래 정보를 출력하는 출력문들이 출력될 때 각 Task들이 수행되며 UARTputcolor를 수행하며 색깔이 바뀌어 출력되는 현상이 있었다. 이를 막기 위해 검은색(default color)으로 명시적으로 지정해주어 출력하게 하였다.

line 70-71:

수행이 끝나면 화면을 지우고 Task를 삭제한다.

3.2

A screenshot of a computer

Description automatically generated with medium confidence