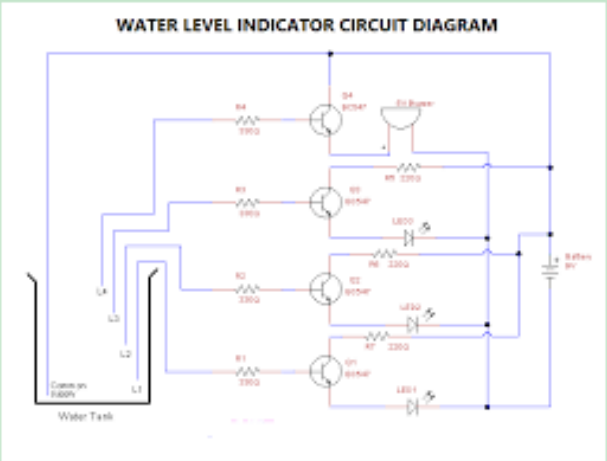
Proposal report

Water indicator

과 목	임베디드시스템S/W설계
담당교수	김태석 교수님
학 과	컴퓨터정보공학부
학 번	2019202103
성 명	이은비
날 짜	2023. 05. 04 (목)

1. 요약

공장, 화학 공장, 변전소 및 기타 액체 저장 시스템에서 사용되는 water level indicator circuit의 일부분인 water level indicator를 구현합니다. 이때 water level indicator circuit은 pump의 activation을 monitoring, rainfall(강우) detection, leakage(누수) detection의 활용에 쓰일 수 있습니다. 그리고 위와 같은 기능을 사용한다고 했을 때 empty, full인 상태에 알려주는 기능을 수행해야 하므로 이런 내용을 task로 하여서 프로그램을 구현합니다. 아래 그림을 참고해서 생각 할 수 있습니다. 차오르는 물의 수위에 따라서 state를 바꾸고 그에 따른 lamp및 text출력을 가능하도록 하는 것입니다.

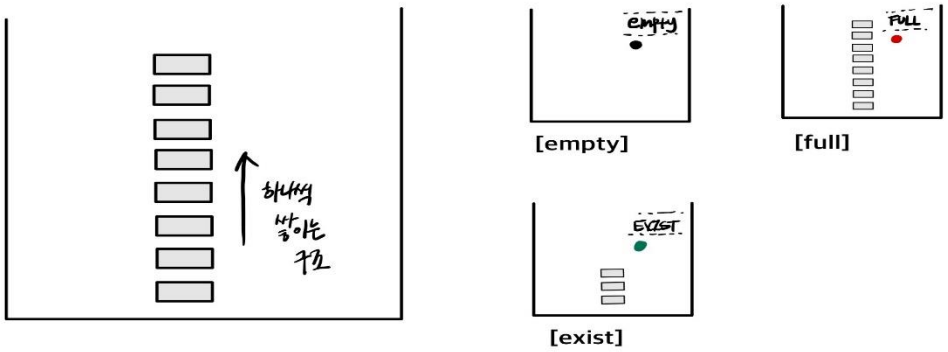


그리고 프로젝트 스펙에서 요구하는 것들은 5개 이상의 Task 생성 및 동작, Semaphore를 활용한 Synchronization, Message queue를 활용한 IPC와 같은 것들이 있고, 이때 프로젝트의 내용과 매칭하여 간략하게 설명하면 다음과 같습니다. task로는 기본적으로 display되는 화면을 구성하는 Task 3개와 물입자를 생성하는 task, 그 물입자의 상태를 체크하여서 임계점에 해당하는 부분마다의 signal 즉 공유자원을 변경할 수 있도록 하는 task, 그리고 그때 공유자원을 변경할 때 경고문구와 lamp를 display하는 task로 구성합니다. 그리고 본 프로젝트에서는 공유자원(lamp.경고문구)와 같은 공유되는 자원에 대해서 binary semaphore(이진형 세마포어)를 사용합니다. Message queue로는 각각의 물입자를 struct로 관리하여서 이 메시지 객체안에 y(위,아래 위치정보)와 state정보와 같은 것들로 각 객체가 message로 관리되도록 합니다.

2. 본문

A. 프로젝트 개요

아래 그림과 같이 display를 구성하며, 각각의 message에 해당하는 block이 아래에서 위로 쌓이는 구조입니다. Empty, exist, full의 state로 구성할 예정이며 각각의 상태에 따라서 문구와 signal lamp의 color를 black, green, red로 display하도록 합니다.



B. RTOS의 필요성

RTOS의 개념과 특징을 살펴보면 시간 운영체제의 핵심은 응용 프로그램 테스크 처리에 걸리는 시간을 일관되게 유지할 수 있는 정도에 따라서 경성(hard) 실시간 운영체제와 연성(soft) 실시간 운영체제로 구분할 수 있으며, 전자가 후자에 비해 지터(jitter)가 적으며 RTOS의 주된 설계 목표는 높은 처리율(throughput)이 아니라, 실시간 성능 보장에 있습니다. 실시간 시스템의 데드라인을 대체로 맞추는 RTOS를 연성 실시간 운영체제라 하고, 데드라인을 결정론적 알고리즘(deterministic algorithm)에 의해 만족하는 경우를 경성 실시간 운영체제라 합니다.

본 프로젝트에서는 water indicator라는 일부를 구현하지만 실제 사용에서는 water indicator circuit으로 공장과 변전소등 물의 상태에 따른 circuit에서의 전류 전도 상태 혹은 물의 누수와 같은 위험감지에 있어서 real-time 즉, 실시간성이 보장되어야 하며 상황에 따라 혹은 적용되는 곳에 따라 경성,연성 실시간 운영체제가 되어야 한다고 생각합니다.

C. 각 task의 정의

Display에 해당하는 task가 우선 되어야 하므로 아래 3가지 task는 각각1,2,3우선순위로 설정합니다. 또한 이후 과정에서는 물입자 즉, message queue에서 message가 쌓이는 과정의 task가 4, 물입자가 차오른 상태에 따른 상태를 check하는 task를 5, check하여 변화된 state에 따른 경고문구 및 lamp와 같은 변화 task를 6으로 구성합니다.

(1)static void TaskViewClear(); //초기화면 보여주기

(2)static void TaskStartDispInit(); //초기화면 보여주기

(3)void TaskViewDisp(); //물입자, 경고문구, 경고등 화면 보여주기

(4)void TaskWaterMake(void *data); //세마포어를 이용한 물입자 생성 TASK

(5)void TaskWaterfilling(void *data); //물입자가 차오르는 상태 check TASK(empty,exist,full)

(6)void TaskSignal(void *data); //경고문구 및 lamp 변화TASK

D. task간 semaphore 와 message queue의 활용방안

Semaphore를 활용한 synchronization은 binary semaphore를 통한 synchronization을 진행할 계획이며 5

번과 6번 task에서 각 state에 해당하는 상태에 따른 signal(공유자원)접근 및 변경을 진행 할 수 있도록 합니다. 또한 4번과 5번task에서 각 정보를 담고 있는 structures 즉, messages가 아래에서 위로 차례로 진행되는 과정을 진행할 때에 message queue를 활용할 수 있도록 합니다.

E. task 동작 결과

화면을 구성하는 task 1,2,3 (1)static void TaskViewClear(); //초기화면 보여주기

(2)static void TaskStartDispInit(); //초기화면 보여주기

(3)void TaskViewDisp(); //물입자, 경고문구, 경고등 화면 보여주기는 string을 출력하는 task가 아니여서 사진을 첨부하지 않고, 이후 task인

(4)void TaskWaterMake(void *data); //세마포어를 이용한 물입자 생성 TASK

(5)void TaskWaterfilling(void *data); //물입자가 차오르는 상태 check TASK(empty,exist,full)

(6)void TaskSignal(void *data); //경고문구 및 lamp 변화TASK

에 대해서 각각의 string을 출력한 결과는 다음과 같습니다. 차례로 main문안에서 priority에 따라 호출하는 사진, 각 task안의 출력되는 내용에 해당하는 사진, 실제 cmd창에서 출력된 string결과 사진입니다.

```c
int main(void)
{
    OSInit(); // uC/OS-II 초기화
    OSTaskCreate(TaskWaterMake, (void*)0, &TaskStk[0][TASK_STK_SIZE - 1], (INT8U)1);
    OSTaskCreate(TaskWaterfilling, (void*)0, &TaskStk[1][TASK_STK_SIZE - 1], 2);
    OSTaskCreate(TaskSignal, (void*)0, &TaskStk[2][TASK_STK_SIZE - 1], 3);
    // 테스크 생성 (적어도 1개 이상)
    OSStart(); // 멀티 테스킹 시작
    return 0;
}
```

```c
void TaskWaterfilling(void* data){
    printf("TaskWaterfilling₩n");
    OSTimeDly(100);
}
void TaskWaterMake(void* data) {
    printf("TaskWaterMake₩n");
    OSTimeDly(100);
}
void TaskSignal(void* data) {
    printf("TaskSignal₩n");
    OSTimeDly(100);
}
```

```
C:\SOFTWARE\PROJECT>test.exe
TaskWaterMake
TaskWaterfilling
TaskSignal
```