Project – Final report

지하철 좌석 확인 시스템

과 목	임베디드시스템S/W설계
담당교수	김태석 교수님
학 과	컴퓨터공학과
학 번	2015722087
성 명	김민철
날 짜	2020. 05. 27 (수)

광운대학교
KwangWoon University

1. 요약

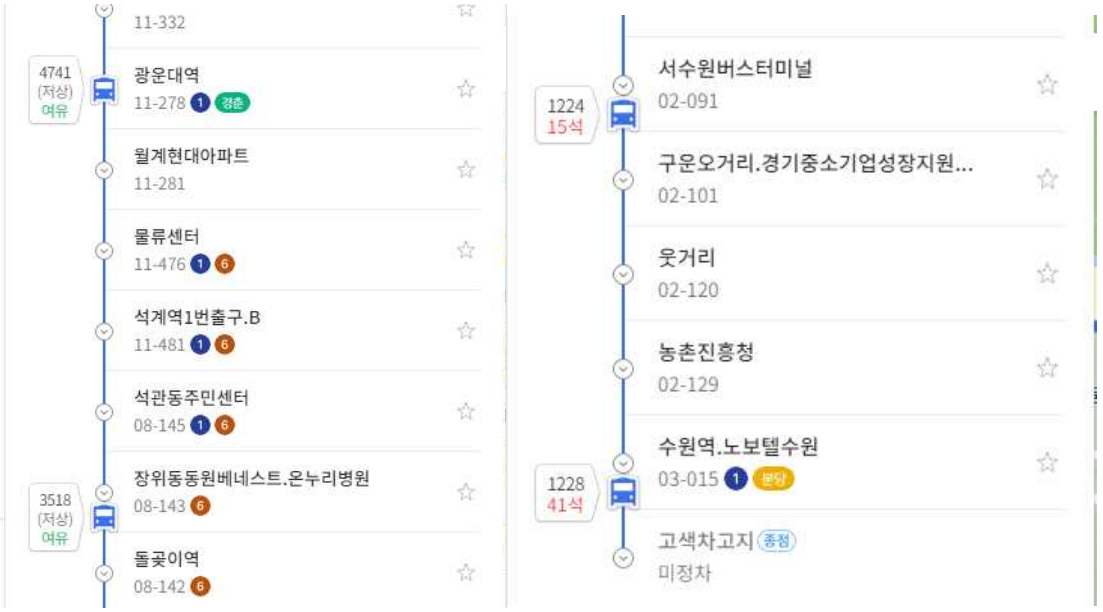
시내 버스의 경우에는 여유, 보통, 혼잡 세 가지의 표시로 버스의 탑승객이 어느 정도 있는지 정류장 혹은 휴대폰으로 확인할 수 있게 하여 이용자로 하여금 편의성을 제공하고, 시외 버스의 경우에는 좌석이 몇 자리가 남아 있는지 확인할 수 있게 하는 편의성을 제공한다. 하지만 버스와 함께 가장 많이 이용되는 교통 수단인 지하철의 경우에는 이용자의 입장에서 제공받을 수 있는 정보가 없다는 생각이 들어 이번 프로젝트를 기획하게 되었다.

이번 프로젝트를 통해 개발하게 될 시스템에서 기존의 시내 버스, 시외 버스에서 제공하는 편의성을 지하철에서도 제공할 수 있게 하는 것이 목표이다. 지하철은 각 열차마다, 칸 마다 탑승 시에 카드를 찍지도 않기 때문에 정확한 인원 파악이 어렵고, 정확한 인원 파악을 위한 시스템을 구현하기에는 많은 비용이 소모될 것 이다. 따라서 확인하는 방법으로는 좌석을 이용하는 것을 생각하였다.

좌석에 착좌 센서를 부착하여 좌석에 앉아 있는지 확인하여 좌석이 일정 수 이상 비어 있을 경우에는 여유, 좌석이 어느 정도 남아 있을 경우에는 보통, 좌석이 모두 이용 중이라면 혼잡의 정보 혹은 남아있는 잔여 좌석 수를 이용자들에게 제공한다.

2. 본문

A. 프로젝트 개요



이미 개발되어 서비스되고 있는 시내 버스와 좌석 버스의 정보이다.

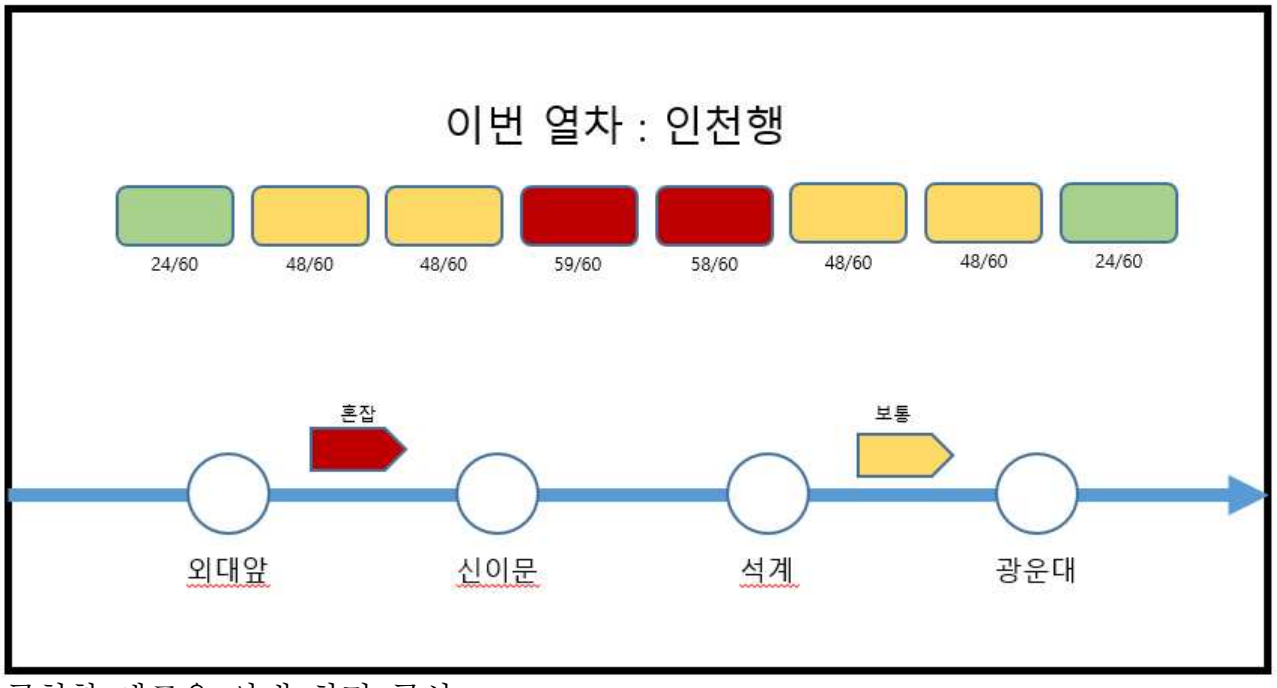
그에 비해 지하철의 정보는 열차의 도착, 출발 시간 외에는 혼잡 정도에 대한 정보가 존재 하지 않는다.

열차 내부에서 사람이 혼잡하지 않을 경우에는 좌석도 비어있으며, 손잡이를 잡고 있는 이용객도 없다.

하지만 혼잡할 경우에는 좌석이 가장 먼저 가득 차며, 손잡이를 잡거나, 손잡이 밑에 서있는 사람들이 많아진다.

이를 지하철의 해당 열차를 탑승해야 알 수 있는 것이 아니라, 휴대폰 혹은 외부에서도 확인하게 하는 것이 이용자를 위한 편의라고 판단하였다. 지하철을 이용하다보면 어떤 칸은 혼잡스러운 데에 비해 또 어떤 칸은 쾌적한 경우를 볼 수가 있었다. 이런 혼잡한 정도의 정보를 이용자들에게 제공한다면 보다 많은 사람들이 쾌적하게 지하철을 이용할 수 있을 것 같다는 생각이 들었다.

-기존의 안내 화면 구성



-구현할 새로운 안내 화면 구성

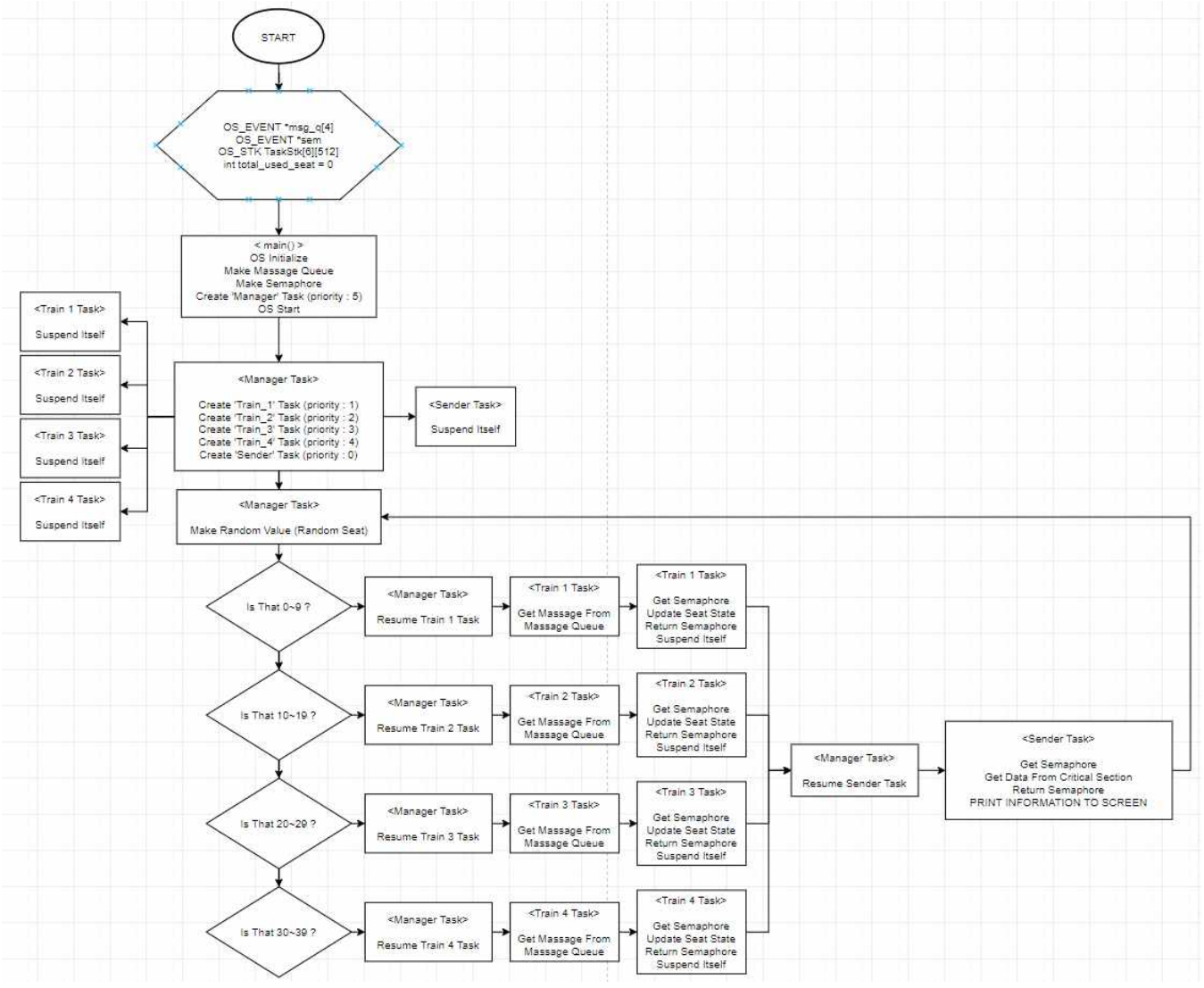
구현된 프로그램이 외부로 데이터를 전송하기만 한다면, 안내 화면과 어플리케이션을 통해 사용자에게 정보를 제공하는 것은 어렵지 않다고 생각한다.

- 구현된 프로그램에 대한 간략한 설명

이번 프로젝트에서는 착좌 센서를 실제로 이용하여 구현하는 것이 아니라, 프로그램으로 어느 정도의 구조만 구현하는 것이기 때문에 먼저 착좌 센서의 역할인 특정 좌석에 앉거나 일어서는 동작을 Random하게 값을 추출하여 좌석을 선택하여 동작하도록 구현하였습니다.
지하철의 칸 수 같은 경우에도 10칸인 지하철도 있고 6칸인 지하철도 있으나 프로젝트에서 예시로 사용할 지하철의 칸 수를 확장하는 것에는 어려움이 없기 때문에 코드의 가독성을 위해 4칸인 지하철로 가정하고 구현하였습니다.
Sender Task에서 데이터를 외부로 전송하여야 하지만 전송할 대상이 없어 이번 프로젝트에서는 결과를 출력하도록 하였습니다.
또 Sender가 매번 동작시마다 데이터를 전송할 경우 과부하가 올 수 있어 실제로는 특정 시간(ex 10초)마다 전송하는 것이 좋을 것 같지만, 이번 프로젝트에서는 결과를 바로 보여드리기 위해 0.3초 간격으로 동작하고 즉시 결과를 출력하도록 하였습니다.

B. 프로젝트의 구조

-Flow Chart



-Pseudo Code

OS초기화
세마포어, 메시지 큐(4개) 생성
Manager 태스크 생성 (우선 순위 : 5)
OS실행

=====>>

Manager 태스크 실행
    Train 1,2,3,4 태스크 실행 (각 우선 순위 : 1,2,3,4)
        Suspend()
    Sender 태스크 실행 (우선 순위 : 0)
        Suspend()

    <무한 루프>
        난수로 입력 값 생성 (rand_seat)
        rand_seat에 맞는 train_num에 seat_num를 메시지 큐를 통해 전달
        해당 train 번호에 맞는 태스크 Resume(train_num)

        Train (train_num)
            세마 포어 획득
            seat 번호에 맞는 좌석 상태 변경
            전체 좌석 수 변경
            세마 포어 반환
            Suspend()

        Sender 태스크 Resume()

        Sender
            세마포어 획득
            전체 좌석 수 정보 복사
            세마포어 반납
            모든 정보 출력
            Suspend()

C. 각 Task의 정의

Manager - 전체 프로그램의 Main 함수 역할을 수행할 Task이다. 실제 적용이 된다면 각 지하철 칸 마다의 착좌 센서의 입력을 각 칸 Task가 받아야 하지만, 이번 프로젝트에서는 Manager에서 난수를 사용하여 변화를 만들 좌석을 선택하게 하여 특정 칸, 특정 좌석을 선택하고 각 칸에 Message Queue를 통해 Message를 전달한 뒤 Task를 Resume하게 하여 동작하도록 구현하였다.
이번 프로젝트에서는 항상 동작하며, 다른 Task들을 생성하고 동작을 명령하기 때문에 가장 낮은 우선 순위를 할당하였다.

Train_* - 각 칸을 담당하는 Task이다. 실제로는 착좌 센서를 통한 칸 별 입력을 모두 직접 처리하지만, 착좌 센서가 존재하지 않기 때문에 이번 프로젝트에서는 각 칸의 특정 좌석을 변경하라는 Message를 Manager Task에게 Message Queue를 통해서 받게 된다. Message를 받으면 해당 좌석의 상태를 변경하고 Semaphore를 통해 동기화를 유지하면서 전체 좌석 개수를 변경시킨다.
이번 프로젝트에서는 Manager에 의해 동작을 수행하기 때문에 Manager보다 높은 우선 순위를 부여하였다.

Sender - 운행 중인 지하철의 정보를 인터넷을 통해 외부로 전달하는 Task이다. 실제로는 정보를 외부(지하철 시스템, 역사 시스템 등)으로 전달해야 하지만, 이번 프로젝트에서는 정보를 출력하도록 처리하였다. Semaphore를 이용하여 전체 좌석 개수를 지역변수에 복사하여 Semaphore를 최대한 적은 시간 소유하도록 구현하였다.
이번 프로젝트에서는 데이터 출력을 하고 끝나기 때문에 가장 높은 우선 순위를 부여하였다.

D. task간 semaphore와 message queue의 활용방안

Semaphore - ‘전체 좌석 개수’를 수정, 읽기 하는 동작에서는 모두 Semaphore를 활용하여 동기화 처리하도록 하였다. 특정 Task가 Semaphore를 오래 소유하는 것이 최대한 없도록 하였으며 필요한 경우에만 최대한 짧게 소유하도록 구현하여 충돌을 최소화하였다.

Message Queue - Manager Task에서 각 칸의 Train Task로 변화가 생길 좌석의 번호를 전달할 때 사용한다. 각 칸별로 하나의 Message Queue를 두어 Message가 Train별로 혼동되지 않도록 처리하였다.

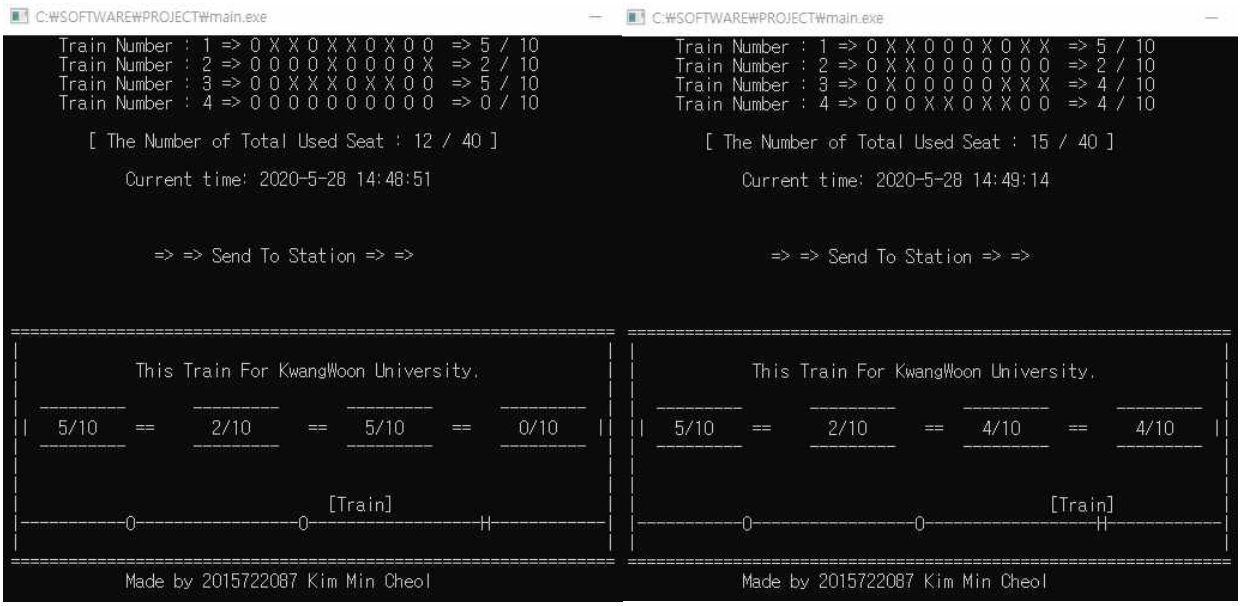
E. 제안서 대비 변경사항

우선 구현할 Task의 개수를 줄였다. 각 칸별 Task가 10개여도, 100개여도 어려울 것은 없지만, 많아 진다면 프로젝트의 목적, 기능을 설명하는 데에 효율이 떨어진다고 생각하여 각 칸별 Task는 4개만 구현하였다.

또 각 칸마다의 event 생성을 할 수가 없어 Manager에서 착좌 센서 역할을 난수의 생성을 통해 대체하였으며 이때 난수를 통한 정보를 Message Queue를 통해 각 칸별 Task로 전달하도록 수정하였다.

마지막으로 Manager에서 Sender로 데이터 전송을 Message Queue를 사용하려했으나 전역 변수로 사용하여 불필요한 자원 손실을 최소화하였으며 Semaphore를 통해 Sender에서 전역 변수를 읽어오도록 수정하였다.

F. 동작 예시



- 중간 지점을 기준으로 위쪽 정보는 저장되어 있는 좌석 정보 그대로를 출력한 것이고, 그 정보를 역사 시스템으로 전달했다고 가정하고 현재 역사에서 출력되는 지하철 안내 화면과 비슷하게 출력해본 모습이다. 사용 중인 좌석의 개수가 나오면서 어느 칸이 여유로운지 확인하고 이동하여 탑승할 수 있다.

3. 고찰

윈도우나 리눅스 환경에서만 동작하는 프로그램만을 다루다가 임베디드 환경에서 동작하는 프로그램의 구현이라 처음에는 굉장히 어려울 거라 생각했다. 복잡할 것 같았고 성공하지 못할 것 같은 프로젝트였지만 막상 해보니 윈도우 환경에서 구현했던 것과 다른 점이 거의 없었다. 처음부터 직접 모든 것을 진행했다면 굉장히 복잡하고 어려웠을지 모르겠지만, 기본 제공 코드를 보면서 생각보다 너무 쉽게 접할 수 있었고 생각했던 것 만큼 좋은 결과물은 아니지만 Message Queue와 Semaphore를 모두 활용하여 UCOS 환경의 프로그램을 구현하고 새로운 것을 배웠다는 것에 의미를 두니 배운 점도 많고 이해가 되지 않았던 부분들에 대해서 직접 코드를 작성하면서 이해가 되면서 기분 좋게 마무리할 수 있는 과제가 된 것 같다.