Proposal Report

도서 관리 시스템

과 목 임베디드시스템S/W설계

담당교수 김태석 교수님

학 과 컴퓨터정보공학부

학 번 2021202058

성 명 송채영

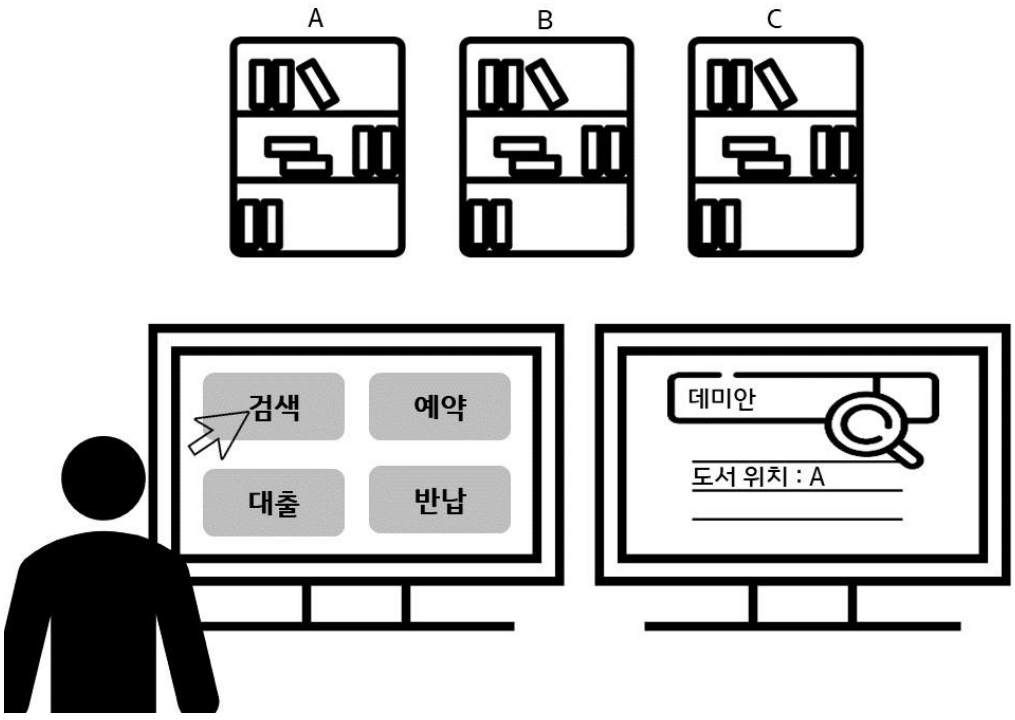
날 짜 2024. 05. 05 (일요일)

1. 요약

uC/OS를 이용해 임베디드 시스템을 구현하는 것이 목적이며 프로젝트의 주제는 도서관 관리 시스템이다.

도서관 관리 시스템은 사용자가 도서를 대출, 반납, 검색, 및 예약을 할 수 있다. 사용자는 인터페이스를 통해 각각의 기능을 선택할 수 있다. 시스템은 도서 대출 및 반납 처리를 동기화하여 동시에 여러 사용자가 동일한 책을 대출하거나 예약하는 것을 방지할 수 있다. 또한 책의 위치를 검색하고 예약하는 기능을 추가하여 사용자가 원하는 책을 찾을 수 있고 미리 예약할 수 있다. 책장은 A, B, C가 있으며 사용자가 원하는 책이 있을 경우 책의 위치(책장)를, 없을 경우 해당 책이 없음을 알려준다. 이미 예약되어 있는 책이거나 다른 사용자가 빌려간 경우 해당 책은 대출 및 예약할 수 없다.

아래는 프로젝트 전반을 설명하는 그림이다.



2. 본문

A) RTOS의 필요성

도서관 관리 시스템에서 RTOS를 사용해야 하는 이유는 주로 실시간 응용 프로그램을 위해서이다. 도서를 대출하거나 반납하고 검색 그리고 예약하는 작업에서 사용자의 요청을 실시간으로 처리할 수 있어야 한다. 또한 여러 사용자가 동시에 동일한 책을 대출하려고 하거나 예약하려고 하는 경우에 동시성 문제가 발생할 수 있다. 이러한 문제를 해결하기 위해 threading, semaphore, mutex 등의 매커니즘을 사용하여 여러 사용자 간의 동시성 및 동기화 문제를 해결할 수 있어야 한다. RTOS는 이러한 여러 task 간의 문제를 해결할 수 있고 작업들을 좀 더 안전하고 효율적으로 처리할 수 있어 RTOS를 사용해야 한다.

B) 개발할 kernel function 설명

개발하려고 하는 kernel function은 OS_TASK.C 코드에서의 OSTaskCreate() 부분이다. OSTaskCreate()함수는 uC/OS-II에서 새로운 task를 생성하는 함수로, 이 함수에서 새로운 task를 만들고 해당 task의 우선순위, 스택 크기, 작업 함수 등을 설정할 수 있다. OSTaskCreate() 함수를 수정하여 도서 대출 및 반납 task의 우선 순위를 설정하고, 대출 및 반납 task 내에서 도서의 대출 상태를 확인하는 동기화 기능을 추가하려고 한다. 이를 통해 시스템이 여러 작업을 처리할 때 도서 대출 및 반납 task에 더 많은 우선권을 부여하여 작업이 더 빨리 처리될 수 있고, 사용자가 동시에 동일한 책을 대출하거나 반납하는 것 역시 방지할 수 있다.

OS_TASK.C 코드의 OSTaskCreate() 함수 부분으로, 커널이 수정된 것을 알 수 있는 string을 넣은 소스코드 사진이다.

```
#if OS_TASK_CREATE_EN > 0
INT8U  OSTaskCreate (void (*task)(void *pd), void *pdata, OS_STK *ptos, INT8U prio)
{
    printf("Kernel Modified, OSTaskCreate() Called.\n");
#if OS_CRITICAL_METHOD == 3                 /* Allocate storage for CPU status register
    OS_CPU_SR  cpu_sr;
#endif
    OS_STK    *psp;
    INT8U      err;
```

```
int main(void)
{
    OSInit();

    Semaphore = OSSemCreate(1); // Semaphore 생성

    MessageQueue = OSQCreate(NULL, 10); // Message Queue 생성

    OSTaskCreate(Task1, (void*)0, &Task1Stk[TASK_STK_SIZE - 1], 1);
    OSTaskCreate(Task2, (void*)0, &Task2Stk[TASK_STK_SIZE - 1], 2);
    OSTaskCreate(Task3, (void*)0, &Task3Stk[TASK_STK_SIZE - 1], 3);
    OSTaskCreate(Task4, (void*)0, &Task4Stk[TASK_STK_SIZE - 1], 4);
    OSTaskCreate(Task5, (void*)0, &Task5Stk[TASK_STK_SIZE - 1], 5);

    //OSTaskCreate(TaskStart, (void*)0, &TaskStk[TASK_STK_SIZE - 1], 1);

    OSStart(); // 멀티 테스킹 시작
    return 0;
}
```

Task에서 OSTaskCreate 함수를 호출하는 소스코드 사진이다.

```
C:₩SOFTWARE₩PROJECT>test.exe
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Task 1 is running.
Task 2 is running.
Task 3 is running.
Task 4 is running.
Task 5 is running.
```

실행했을 때의 결과로 잘 출력되는 것을 확인할 수 있다.

C) 각 Task의 정의

총 5개의 task가 있으며, task 1은 사용자 인터페이스, task 2는 도서 대출 처리, task 3은 도서 반납 처리, task 4는 책 위치 검색, task 5는 책 예약 처리에 해당한다.

우선 task1은 사용자 인터페이스로 사용자와 상호작용을 담당하는 task이다. 사용자의 입력을 받아 도서 대출, 도서 반납, 도서 검색, 도서 예약 등의 요청을 처리한다. 즉 이 task는 도서 대출 및 반납처리, 책 검색, 예약 처리에 해당하는 각각의 task에 요청을 보내고 그 결과를 받아 사용자에게 제공한다. task 1의 경우 사용자와 상호작용을 담당하고 사용자의 입력을 받고 처리하는 것이 시스템의 핵심 순위이므로 우선순위를 1로 설정하였다.

task 2는 도서 대출 처리를 담당한다. 사용자가 대출을 요구할 때 semaphore를 획득하여 해당 작업을 수행하고 작업이 완료되면 semaphore를 해제하여 다음 요청이 처리될 수 있도록 한다. 우선 순위를 2로 설정하였다.

task 3은 도서 반납 처리를 담당한다. 사용자가 반납을 요구할 때 semaphore를 획득하여 해당 작업을 수행하고, 작업이 완료되면 semaphore를 해제하여 다음 요청이 처리될 수 있도록 한다. 우선 순위를 3으로 설정하였다.

task 4는 책의 위치를 검색하는 일을 처리한다. Message Queue를 이용하여 사용자의 요청을 받고 처리 결과를 반환한다. 예를 들어 사용자가 책을 검색할 경우 message Queue를 사용하여 사용자에게 요청 받고 처리 결과를 알려준다. 우선 순위를 4로 설정하였다.

task 5는 책을 예약하는 일을 처리한다. 사용자가 특정 책을 대출할 수 없는 경우 예약을 요청하면 해당 책을 대출 가능할 때까지 대기열에 넣어두고 대출이 가능해질 경우 사용자에게 알려준다. 우선 순위를 5로 설정하였다.

task 1을 제외하고는 우선순위를 각각 2, 3, 4, 5로 초기값으로 설정하고 각 task의 상황에 맞게 우선 순위를 조정할 예정이다.

D) task 동작 결과

아래는 각 task에 해당 task임을 알 수 있는 간단한 string을 넣은 소스코드이다.

```
void Task1(void* pdata)
{
    INT16U i;
    for (i = 0; ; i++)
    {
        printf("Task 1 is running.₩n");
        OSTimeDly(1);
    }
}
void Task2(void* pdata)
{
    INT16U i;
    for (i = 0; ; i++)
    {
        printf("Task 2 is running.₩n");
        OSTimeDly(1);
    }
}
```

```
void Task3(void* pdata)
{
    INT16U i;
    for (i = 0; ; i++)
    {
        printf("Task 3 is running.₩n");
        OSTimeDly(1);
    }
}
void Task4(void* pdata)
{
    INT16U i;
    for (i = 0; ; i++)
    {
        printf("Task 4 is running.₩n");
        OSTimeDly(1);
    }
}
void Task5(void* pdata)
{
    INT16U i;
    for (i = 0; ; i++)
    {
        printf("Task 5 is running.₩n");
        OSTimeDly(1);
    }
}
```

아래는 위 코드를 실행했을 때의 결과이다. 설정한 우선순위에 맞게 task가 출력이 되는 것을 확인할 수 있다.

```
C:₩SOFTWARE₩PROJECT>test.exe
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Kernel Modified, OSTaskCreate() Called.
Task 1 is running.
Task 2 is running.
Task 3 is running.
Task 4 is running.
Task 5 is running.
Task 1 is running.
Task 2 is running.
Task 3 is running.
Task 4 is running.
Task 5 is running.
Task 1 is running.
Task 2 is running.
Task 3 is running.
Task 4 is running.
Task 5 is running.
Task 1 is running.
Task 2 is running.
Task 3 is running.
Task 4 is running.
Task 5 is running.
```