Final Report

비트코인 채굴장 관리 시스템

과 목	임베디드시스템S/W설계
담당교수	김태석 교수님
학 부	컴퓨터정보공학부
학 번	2018202065
성 명	박철준
날 짜	2023. 05. 18 (목요일)

광운대학교
KwangWoon University

● 요약

■ 시스템 요약 설명

해당 제안서는 비트코인 채굴장 관리 프로그램을 위한 제안서로 비트코인 채굴장에서 관리하는 많은 양의 채굴기를 화면에 모니터링을 하여 365일 24시간 문제없이 가동될 수 있도록 보조의 하는 시스템이다.

개발하고자 하는 비트코인 채굴장 관리 프로그램의 요구사항은 채굴기 고장을 탐지하고 다수의 채굴기가 고장이 난 경우까지 모니터링하여 해결을 위한 알림 문구 출력, 문제 해결 시 알림 문구 제거하여 사용자의 채굴장 관리에 도움을 주는 것이다.

● 본문

■ 프로젝트 개요

▶ 프로젝트 전체 설명

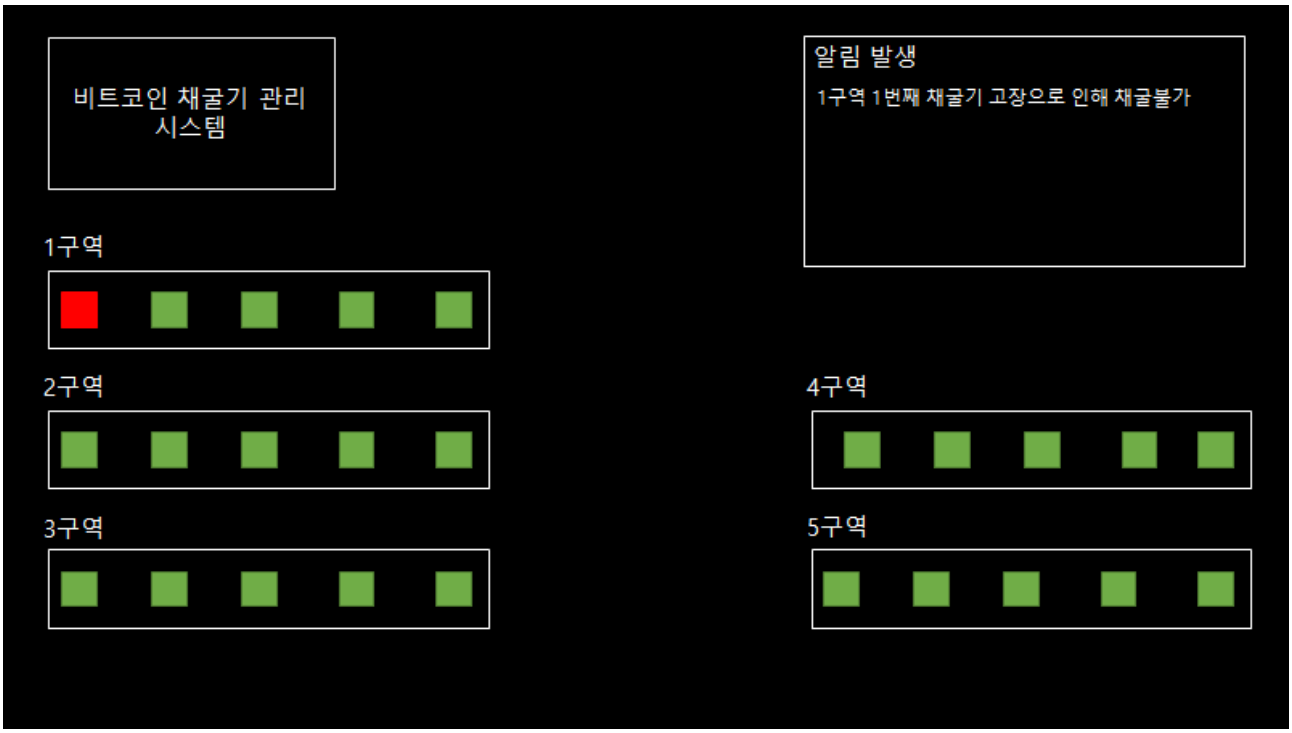
비트코인 채굴장에서 관리하는 많은 양의 채굴기를 화면에 모니터링을 하여 365일 24시간 문제없이 가동될 수 있도록 보조의 하는 시스템이다. 해당 프로젝트에서는 채굴장에서 관리하는 채굴기를 총 60대로 가정하고 해당 채굴기가 문제없이 작동하는 것에 대해 모니터링을 진행한다. 문제가 되는 사항으로는 채굴기 고장으로 인한 채굴기 1대의 동작 정지 및 해당 고장이 중첩되어 발생하여 다수 채굴기의 동작 정지로 정의한다. 해당 문제가 발생 시 시스템은 해결을 위한 알림 문구를 출력하고 사용자를 통해 문제가 해결될 시 알림 문구를 삭제하게 된다. 또한, 다수의 문제(해당 프로그램에서는 최대 22개 문제까지 해결)가 발생 시 해결 TASK의 우선순위가 되었을 때 한꺼번에 문제를 해결하고 결과적으로 고장을 내버려 두지 않는다.

▶ 그림을 통한 설명

시스템의 초기 설계 화면은 아래와 같다.

전체적으로 채굴장을 5구역으로 나누어 채굴기를 관리하도록 하였다.

아래는 채굴기에 문제가 발생 시의 보이는 설계화면이다.



그림은 1구역의 채굴기 한대에 문제가 생겨 이상이 생긴 모습으로 알림 문구가 출력된다.

아래는 다수의 채굴기의 고장 발생시 보이는 설계화면이다.

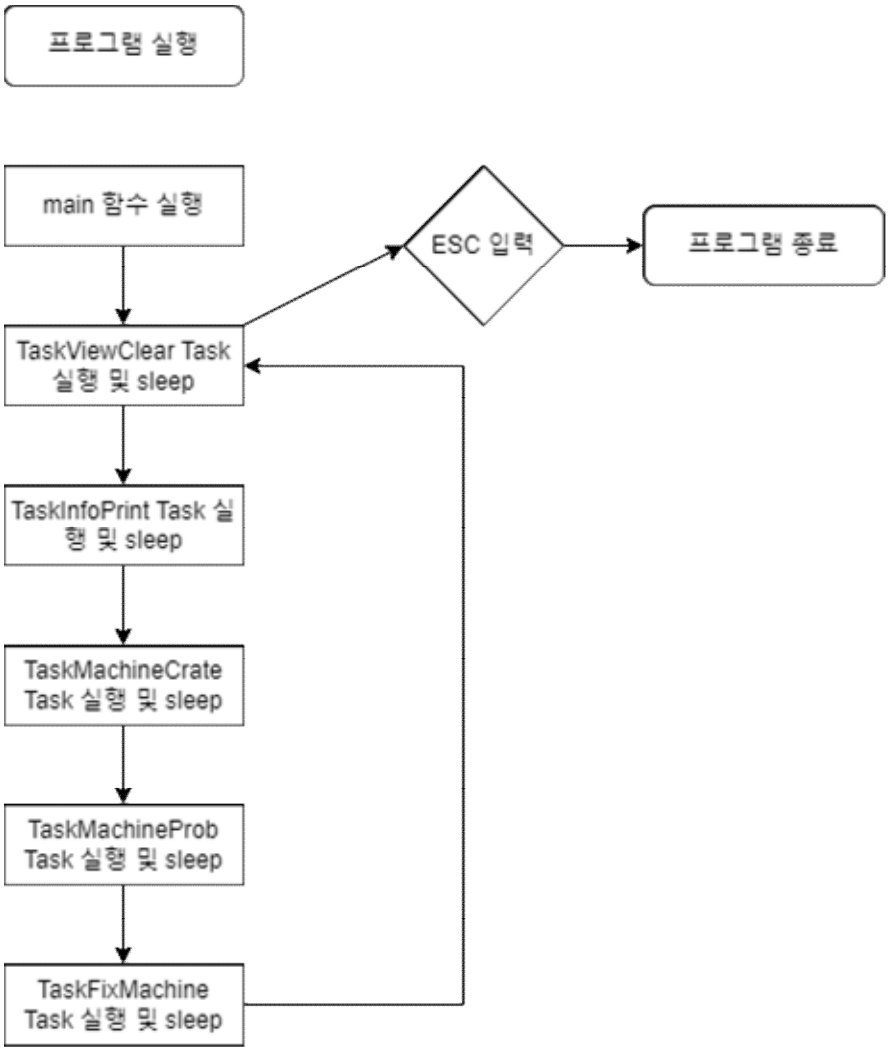
채굴기 고장에 대한 알림 문구가 누적되어 출력된다.

아래는 고장 채굴기가 수리된 모습에 대한 설계화면이다.

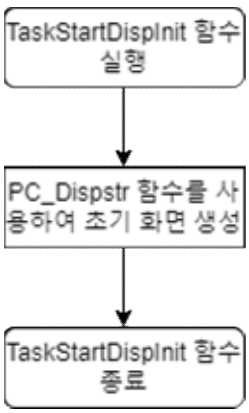
■ 프로젝트 구조

▶ Flow chart

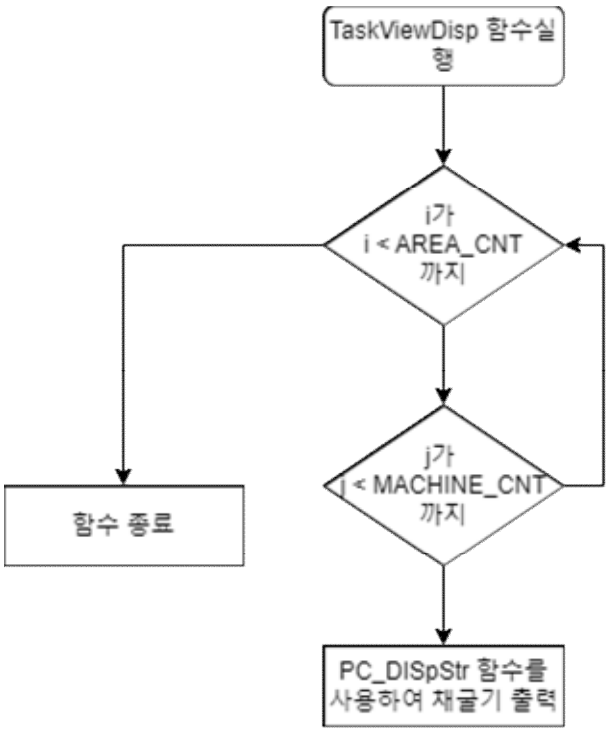
1. 전체 시스템



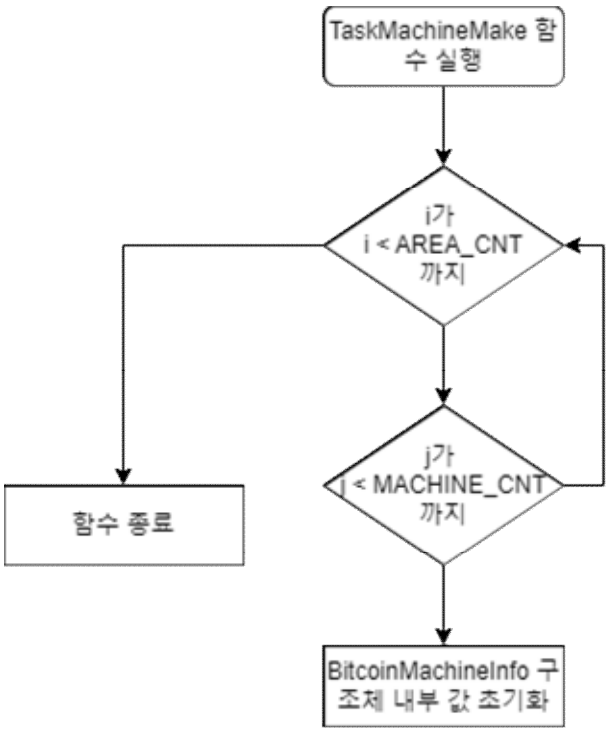
2. TaskStartDispInit 함수



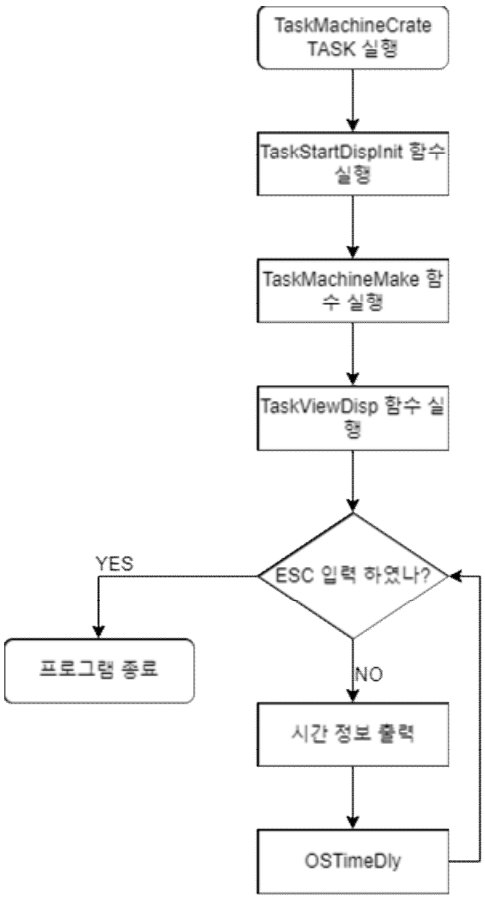
3. TaskViewDisp 함수



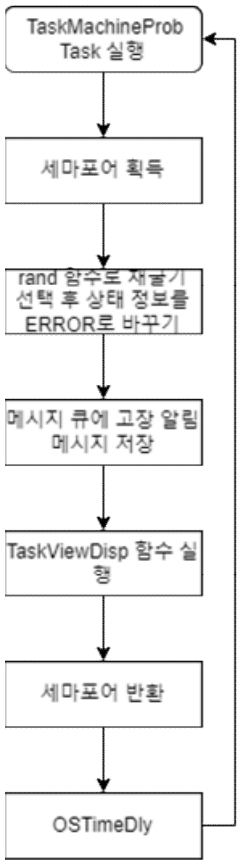
4. TaskMachineMake 함수



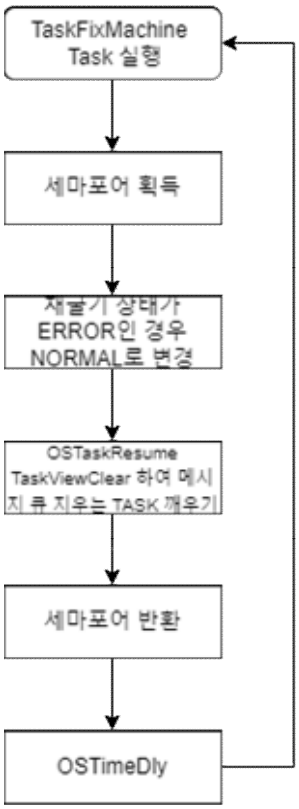
5. TaskMachineCrate TASK



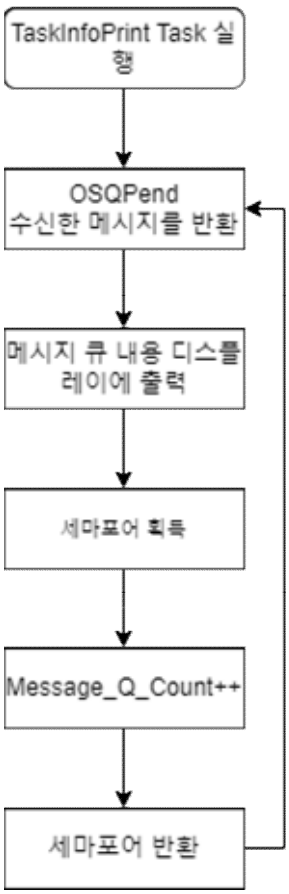
6. TaskMachineProb TASK



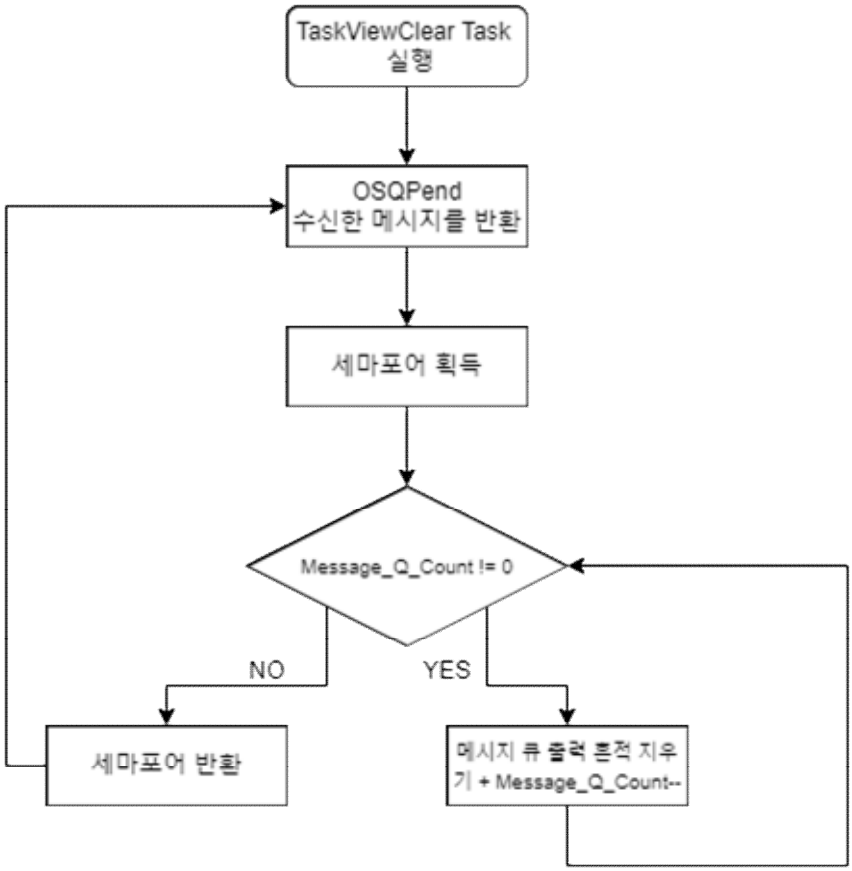
7. TaskFixMachine TASK



8. TaskInfoPrint TASK



9. TaskViewClear TASK



▶ Pseudo code

1. main

```
#include "Includes.h"
#include <time.h>

#define TASK_STK_SIZE   512

#define TASK_COUNT  5//태스크 개수
#define TASK_PRIO   10


#define MACHINE_CNT     12
#define AREA_CNT    5

#define NORMAL  1
#define ERROR   2

#define N_MSG 100 //메시지 큐 사이즈
OS_EVENT* msg_q; //메시지 큐를 위한 OS_EVENT 포인터
void* msg_array[N_MSG]; // 메시지 큐 배열


typedef struct {
    INT8U color;
    INT8U posX;
    INT8U posY;
    INT8U state;
}BITCOIN_MACHINE_INFO;//채굴기 정보를 가지는 구조체


OS_STK TaskStk[TASK_COUNT][TASK_STK_SIZE];
OS_EVENT* sem;// 세마 포어를 위한 OS_EVENT 포인터



BITCOIN_MACHINE_INFO BitcoinMachineInfo[AREA_CNT][MACHINE_CNT];// 구조체 배열 전역변수로 선언


INT8U Bit_MachinePos_X[MACHINE_CNT] = { 3,7,11,15,19,23,27,31,35,39,43,47 }; // 비트코인 채굴기의 X좌표

INT8U Bit_MachinePos_Y[AREA_CNT] = {7,11,15,19,23}; //비트코인 채굴기의 Y좌표

INT16U Message_Q_Count = 0; // 메시지 큐 사용 횟수 카운트
```

```
int  main (void)
{
    INT16U i;
    OSInit();

    msg_q = OSQCreate(msg_array, (INT16U)N_MSG); // 메시지큐 구조체를 생성하고, 큐 구조체가 포인터 배열을 가르키도록 초기화
    sem = OSSemCreate(1);//세마포어 생성 및 초기화
    if (msg_q가 0일때)
    {
        creating msg_q is failed 문구 출력;
        return -1;
    }


    OSTaskCreate(TaskViewClear, (void*)NULL, &TaskStk[0][TASK_STK_SIZE - 1], TASK_PRIO / 4);
    OSTaskCreate(TaskInfoPrint, (void*)NULL, &TaskStk[0][TASK_STK_SIZE - 1], TASK_PRIO/2);
    OSTaskCreate(TaskMachineCrate, (void *)NULL, &TaskStk[0][TASK_STK_SIZE - 1], TASK_PRIO);
    OSTaskCreate(TaskMachineProb, (void*)NULL, &TaskStk[1][TASK_STK_SIZE - 1], (INT8U)(TASK_PRIO + 1));
    OSTaskCreate(TaskFixMachine, (void*)NULL, &TaskStk[2][TASK_STK_SIZE - 1], (INT8U)(TASK_PRIO + 2));

    OSStart();
    return 0;
}
```

2. TaskStartDispInit

```
void  TaskStartDispInit()//초기화면 그리기
{
   PC_DispStr(0, 0,  "  ------------------------------                    ------------------------  ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 1,  "  |비트코인 채굴장 관리 프로그램|                  |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 2,  "  ------------------------------                   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 3,  "           현재 TASK:                              |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 4,  "                                                   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 5,  "  1구역                                            |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 6,  "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 7,  "  |                                              |   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 8,  "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 9,  "  2구역                                            |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 10, "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 11, "  |                                              |   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 12, "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 13, "  3구역                                            |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 14, "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 15, "  |                                              |   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 16, "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 17, "  4구역                                            |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 18, "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 19, "  |                                              |   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 20, "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 21, "  5구역                                            |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 22, "  ------------------------------------------------   |                        | ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 23, "  |                                              |   |------------------------| ", DISP_FGND_GRAY + DISP_BGND_BLACK);
   PC_DispStr(0, 24, "  ------------------------------------------------                           ", DISP_FGND_GRAY + DISP_BGND_BLACK);
}
```

3. TaskViewDisp

```
//현재 채굴기 상태 그리기
static void TaskViewDisp()
{
    INT8U i, j;

    for(i를 0부터 AREA_CNT까지 증가){
        for(j를 0부터 MACHINE_CNT까지 증가){
            if(채굴기 상태가 정상이든 고장상태이든지 상관 없이){
                PC_DispStr(BitcoinMachineInfo[i][j].posX, BitcoinMachineInfo[i][j].posY, "■", BitcoinMachineInfo[i][j].color);
            }
        }
    }


}
```

4. TaskMachineMake

```
//비트코인채굴기 상태 초기화
void TaskMachineMake()
{


    INT8U color, posX, posY, position = 0;

    for (i를 0부터 AREA_CNT까지 증가)
    {
        posY = Bit_MachinePos_Y[i];


        for (j를 0부터 MACHINE_CNT까지 증가)
        {
            posX = Bit_MachinePos_X[j];
            BitcoinMachineInfo[i][j].posX = posX;
            BitcoinMachineInfo[i][j].posY = posY;
            BitcoinMachineInfo[i][j].color = DISP_FGND_GREEN;
            BitcoinMachineInfo[i][j].state = NORMAL;


        }
    }


}
```

5. TaskMachineCrate

```
//전체 비트코인 채굴기 생성 관리 및 화면 제어
void TaskMachineCrate (void *pdata)
{

    INT8U msg[40];
    INT8U i;
    INT16S key;

    TaskStartDispInit 함수 실행

    TaskMachineMake 함수 실행

    TaskViewDisp 함수 실행

    for (;;) {
        if (ESC 키 입력하면) {                              //See if key has been pressed
            프로그램 종료
        }

        현재 시간 정보 저장 출력

        TaskViewDisp 함수 실행 //현재 채굴기 상태 출력

        OSTimeDly(1);

    }

}
```

6. TaskMachineProb

```
//채굴기 기계 고장 TASK
void TaskMachineProb(void* pdata)
{
    while (1)
    {
        INT8U err;

        세마포어를 획득
        // critical path

        srand(time(NULL));
        INT8U error_aera = (rand() % 5);
        INT8U error_machine = (rand() % 12);

        while (비트코인 채굴기 상태가 ERROR 이면)
        {

            error_aera = (rand() % 5);
            error_machine = (rand() % 12);
        }

        BitcoinMachineInfo[error_aera][error_machine].color = DISP_FGND_RED;
        BitcoinMachineInfo[error_aera][error_machine].state = ERROR;

        char msg[100];//메시지큐에 들어갈 메시지 변수
        sprintf(msg, "%d 구역 %d번째 채굴기 고장₩n", error_aera+1, error_machine+1); // 변수에 메시지 저장

        err = OSQPost(msg_q, msg); // 메시지 큐에 저장
        while (err != OS_NO_ERR)
        {
            err = OSQPost(msg_q, msg);
        }

        현재 채굴기 상태 출력

        세마포어 반환

        OSTimeDly(4);
    }
}
```

7. TaskFixMachine

```
// 채굴기 기계 수리 TASK
void TaskFixMachine(void* pdata)
{
    while (1)
    {
        INT8U err;
        세마포어 획득
        INT8U i, j;
        for (i를 0부터 AREA_CNT까지 증가) {
            for (j를 0부터 MACHINE_CNT까지 증가) {
                if (BitcoinMachineInfo[i][j].state == ERROR) {
                    BitcoinMachineInfo[i][j].state = NORMAL;
                    BitcoinMachineInfo[i][j].color = DISP_FGND_GREEN;
                }
            }
        }

        메시지 큐 출력 흔적을 지우는 TASK 깨우기
        현재 채굴기 정보 디스플레이에 출력

        세마포어 반환
        OSTimeDly(rand() % 22);//채굴기를 수리할 타이밍을 정해주는데 디스플레이에 최대한으로 담을 수 있는 양 이내로 고장 개수를 나타내고 이 후에는 수리
    }
}
```

8. TaskInfoPrint

```
//채굴기 고장상태를 출력 하는 TASK
void TaskInfoPrint(void* pdata)
{


    void* msg;
    INT8U err;
    for (;;) {

        msg = OSQPend(msg_q, 0, &err); //수신한 메시지 큐를 반환
        if (msg != 0)
        {
            메시지 큐 내용 디스플레이에 출력
            세마포어를 획득
            Message_Q_Count++; // 메시지 큐 사용 카운트 증가
            세마포어 반환
        }
    }
}
```

9. TaskViewClear

```
//메시지큐 흔적 지우기
void TaskViewClear(void* pdata)
{
    while (1)
    {
        TASK 중지

        INT8U err;

        세마포어를 획득
        while (Message_Q_Count != 0)
        {
            출력 흔적 지우기
            Message_Q_Count--; // 메시지 큐 사용 카운트 감소
        }
        출력 흔적 지우기
        세마포어 반환
    }
}
```

■ 각 task의 정의

1. TaskMachineCrate

전체 비트코인 채굴기 생성 관리 및 지속적인 화면 제어 TASK (3순위)

2. TaskMachineProb

채굴기 고장으로 인해 해당 채굴기를 사용하지 못하게 하는 TASK (4순위)

3. TaskFixMachine

채굴기 수리로 고장난 채굴기를 정상상태로 변경하는 TASK (5순위)

4. TaskInfoPrint

채굴기 고장 알림을 메시지 큐로 받아 이를 출력하는 TASK (2순위)

5. TaskViewClear

출력화면에 메시지 큐에서 출력한 고장 알림을 지우는 TASK (1순위)

■ task간 semaphore와 message queue의 활용방안

▶ semaphore 활용방안

전역 변수로 선언한 BITCOIN_MACHINE_INFO 구조체와 Message_Q_Count 변수를 공유 자원으로 할당하고 해당 구조체와 변수를 사용할 때를 critical path로 생각하여 critical path로 들어가기 전 세마포어 획득 나오기 전 세마포어 반환으로 사용하였고 아래는 해당 과정을 나타낸 화면이다.

```
OSSemPend(sem, 0, &err);//세마포어를 획득
TaskMachineMake();// 채굴기 생성하고 정상상태로 초기화
OSSemPost(sem);//세마포어 반환
```

전체 비트코인 채굴기 생성 관리 및 지속적인 화면 제어 TASK인 TaskMachineCrate에서 채굴기를 생성하고 정상상태로 초기화를 진행할 때 BITCOIN_MACHINE_INFO 구조체를 사용하기 때문에 해당 과정에서 세마포어를 사용한다.

```
OSSemPend(sem, 0, &err);//세마포어를 획득
// critical path

srand(time(NULL)); // (11)
INT8U error_aera = (rand() % 5);
INT8U error_machine = (rand() % 12);

while (BitcoinMachineInfo[error_aera][error_machine].state == ERROR)
{

    error_aera = (rand() % 5);
    error_machine = (rand() % 12);
}

BitcoinMachineInfo[error_aera][error_machine].color = DISP_FGND_RED;
BitcoinMachineInfo[error_aera][error_machine].state = ERROR;
char msg[100];//메시지큐에 들어갈 메시지 변수
sprintf(msg, "%d 구역 %d번째 채굴기 고장₩n", error_aera+1, error_machine+1); // 변수에 메시지 저장
err = OSQPost(msg_q, msg); // 메시지 큐에 저장
while (err != OS_NO_ERR)
{
    err = OSQPost(msg_q, msg);
}

TaskViewDisp(); //현재 채굴기 상태 출력

OSSemPost(sem);//세마포어 반환
```

채굴기 고장으로 인해 해당 채굴기를 사용하지 못하게 하는 TASK인 TaskMachineProb에서 고장 상태로 변경하는 과정에서 BITCOIN_MACHINE_INFO 구조체를 사용함으로 세마포어를 통해 접근을 통제한다.

```
OSSemPend(sem, 0, &err);//세마포어 획득
INT8U i, j;
for (i = 0; i < AREA_CNT; i++) {
    for (j = 0; j < MACHINE_CNT; j++) {
        if (BitcoinMachineInfo[i][j].state == ERROR) {
            BitcoinMachineInfo[i][j].state = NORMAL;
            BitcoinMachineInfo[i][j].color = DISP_FGND_GREEN;
        }
    }
}

OSTaskResume(TASK_PRIO / 4);//메시지 큐 출력 흔적을 지우는 TASK 깨우기
TaskViewDisp();//현재 채굴기 정보 디스플레이에 출력

OSSemPost(sem);//세마포어 반환
//PC_DispStr(22, 3, "중지 후 TaskFixMachine      ", DISP_FGND_YELLOW + DISP_BGND_BLUE);
OSTimeDly(rand() % 22);//채굴기를 수리할 타이밍을 정해주는데 디스플레이에 최대한으로 담을 수 있는 양 이내로 고장 개수를 나타내고 이 후에는 수리
```

채굴기 수리로 고장난 채굴기를 정상상태로 변경하는 TASK인 TaskFixMachine에서 고장상태의 채굴기를 정상 상태로 변경하는 과정에서 BITCOIN_MACHINE_INFO 구조체를 사용함으로 세마포어를 통해 접근을 통제한다.

```
msg = OSQPend(msg_q, 0, &err); //수신한 메시지를 반환
if (msg != 0)
{
    PC_DispStr(54, Message_Q_Count+1, msg, DISP_FGND_GRAY + DISP_BGND_BLACK); // 메시지 큐 내용 디스플레이에 출력
    OSSemPend(sem, 0, &err);//세마포어를 획득
    Message_Q_Count++; // 메시지 큐 사용 카운트 증가
    OSSemPost(sem);//세마포어 반환
}
OSTimeDly(2);
```

채굴기 고장 알림을 메시지 큐로 받아 이를 출력하는 TASK인 TaskInfoPrint에서 메시지 큐 사용 카운트를 증가 시키기 위해 사용하는 전역 변수인 Message_Q_Count를 사용함으로 세마포어를 통해 다른 TASK에서 접근하였을 시 이를 통제한다.

```
OSSemPend(sem, 0, &err);//세마포어를 획득
while (Message_Q_Count != 0)
{
    PC_DispStr(54, Message_Q_Count+1, "                          ", DISP_FGND_GRAY + DISP_BGND_BLACK); // 흔적 지우기
    Message_Q_Count--; // 메시지 큐 사용 카운트 감소
}
PC_DispStr(54, Message_Q_Count+1, "                          ", DISP_FGND_GRAY + DISP_BGND_BLACK); // 흔적 지우기
OSSemPost(sem);//세마포어 반환
OSTimeDly(2);
```

출력화면에 메시지 큐에서 출력한 고장 알림을 지우는 TASK인 TaskViewClear에서 메시지 큐 사용 카운트를 감소를 위해 사용하는 전역 변수인 Message_Q_Count를 사용함으로 세마포어를 통해 다른 TASK에서 접근하였을 시 이를 통제한다.

▶ message queue 활용방안

채굴기가 고장상태로 변했을 시 해당 채굴기 위치 정보에 대한 알림 문구를 메시지 큐에 저장하고 해당 메시지 큐를 출력하는 TASK에서는 항상 메시지 큐를 기다리다 메시지 큐가 들어오면 깨어나 해당 메시지 큐에 들어잇는 알림 문구를 출력하는 방식으로 사용하였다. 아래는 이에 대한 과정을 나타낸 화면이다.

```
char msg[100];//메시지큐에 들어갈 메시지 변수
sprintf(msg, "%d 구역 %d번째 채굴기 고장₩n", error_aera+1, error_machine+1); // 변수에 메시지 저장
err = OSQPost(msg_q, msg); // 메시지 큐에 저장
while (err != OS_NO_ERR)
{
    err = OSQPost(msg_q, msg);
}
```

채굴기 고장으로 인해 해당 채굴기를 사용하지 못하게 하는 TASK인 TaskMachineProb에서 메시지 큐에 들어갈 메시지를 생성 후 OSQPost를 통해 메시지 큐에 저장한다.

```
void* msg;
INT8U err;
for (;;) {
    //PC_DispStr(22, 3, "TaskInfoPrint         ", DISP_FGND_YELLOW + DISP_BGND_BLUE);
    msg = OSQPend(msg_q, 0, &err); //수신한 메시지를 반환
    if (msg != 0)
    {
        PC_DispStr(54, Message_Q_Count+1, msg, DISP_FGND_GRAY + DISP_BGND_BLACK); // 메시지 큐 내용 디스플레이에 출력
        OSSemPend(sem, 0, &err);//세마포어를 획득
        Message_Q_Count++; // 메시지 큐 사용 카운트 증가
        OSSemPost(sem);//세마포어 반환
    }

}
```

채굴기 고장 알림을 메시지 큐로 받아 이를 출력하는 TASK인 TaskInfoPrint에서 OSQPend를 통해 메시지 큐에 메시지가 들어옴을 감지하면 깨어나 내용을 디스플레이에 출력하고 다시 OSQPend를 통해 대기 상태로 들어간다.

■ 제안서 대비 변경사항

▶ 설계 화면에 대한 변경 사항

제안성에서는 회선 집합 부분에서 네트워크 장비의 상태까지 망 관제가 가능한 것으로 설계하였지만 네트워크 장비의 경우에는 채굴기 관리 시스템의 성격상 채굴기 상태만 관제하는 것이 오히려 목적에 맞는 시스템일 것으로 판단하여 해당 부분을 삭제하였다. 그에 따라 설계한 TASK도 수정하였다. 이는 아래와 같다.

▶ 각 task의 정의에 대한 변경사항

기존 제안서의 TASK는 아래와 같다.

1. TaskRandProb

랜덤한 확률로 채굴기 고장 혹은 인터넷 고장을 발생시키는 Task 및 이를 통해 얻은 문제를 우선순위로 해결하는 TASK (1순위)

2. TaskMachineProb

채굴기 고장으로 인해 해당 채굴기를 사용하지 못하게 하는 TASK (2순위)

3. TaskNetworkProb

인터넷 고장으로 인해 다수 채굴기를 사용하지 못하게 하는 TASK (3순위)

4. TaskFixNetwork

인터넷 수리로 채굴기를 정상상태로 변경하는 TASK (4순위)

5. TaskFixMachine

채굴기 수리로 채굴기를 정상상태로 변경하는 TASK (5순위)

1번, 3번, 4번 TASK의 경우 인터넷 고장은 고려하지 않음으로 해당 TASK를 삭제하였다.

삭제한 TASK를 대신하여 아래와 같은 TASK를 설계하였다.

1. TaskMachineCrate

전체 비트코인 채굴기 생성 관리 및 지속적인 화면 제어 TASK (3순위)

2. TaskInfoPrint

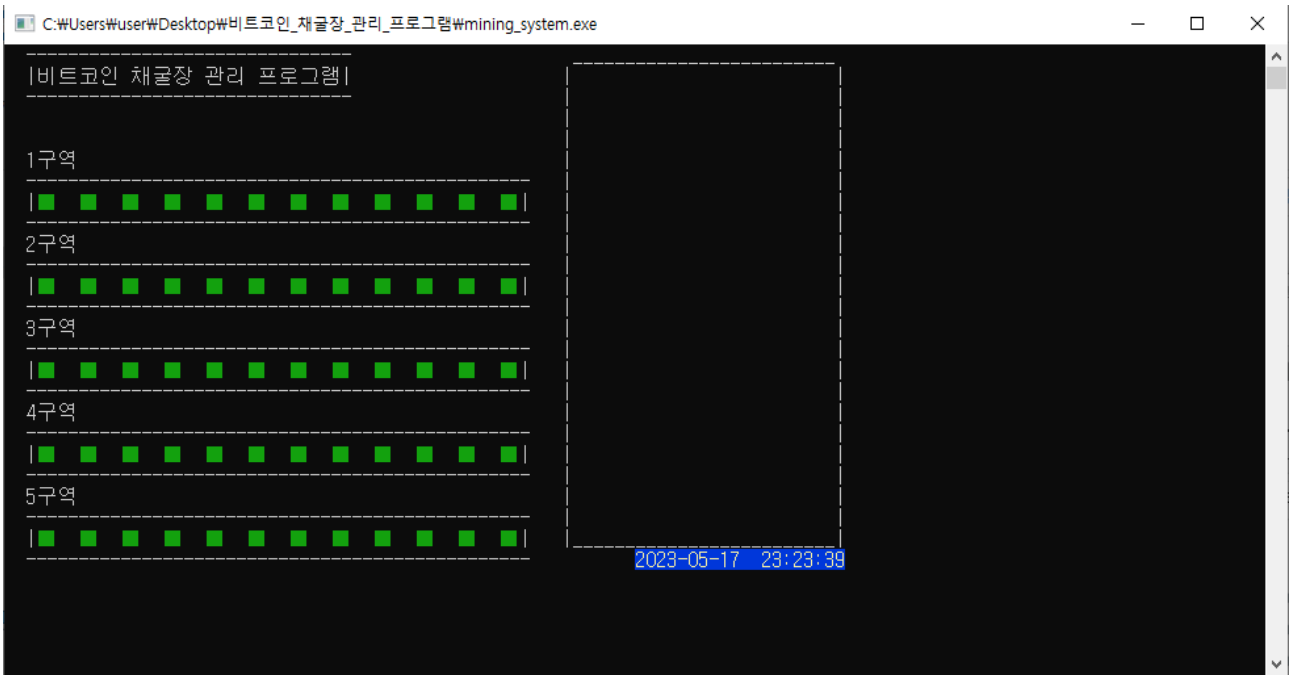
채굴기 고장 알림을 메시지 큐로 받아 이를 출력하는 TASK (2순위)

3. TaskViewClear

출력화면에 메시지 큐에서 출력한 고장 알림을 지우는 TASK (1순위)

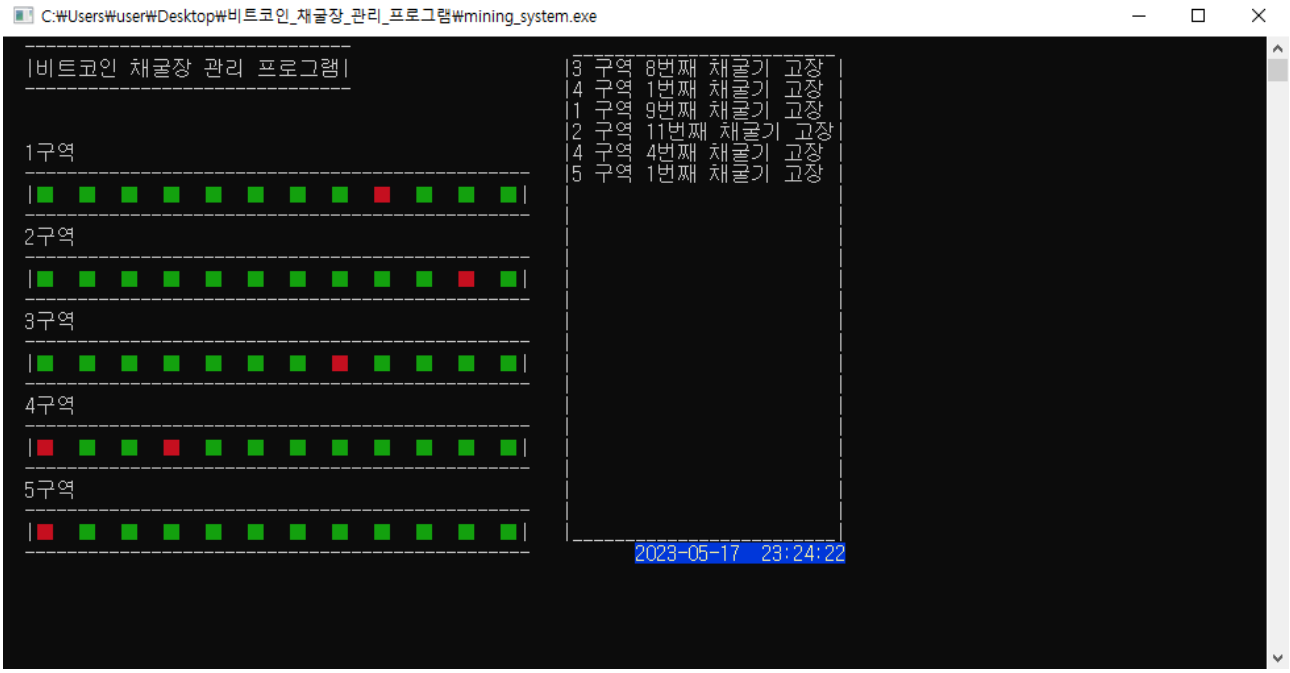
■ 동작 예시

시스템의 초기화면은 아래와 같다.



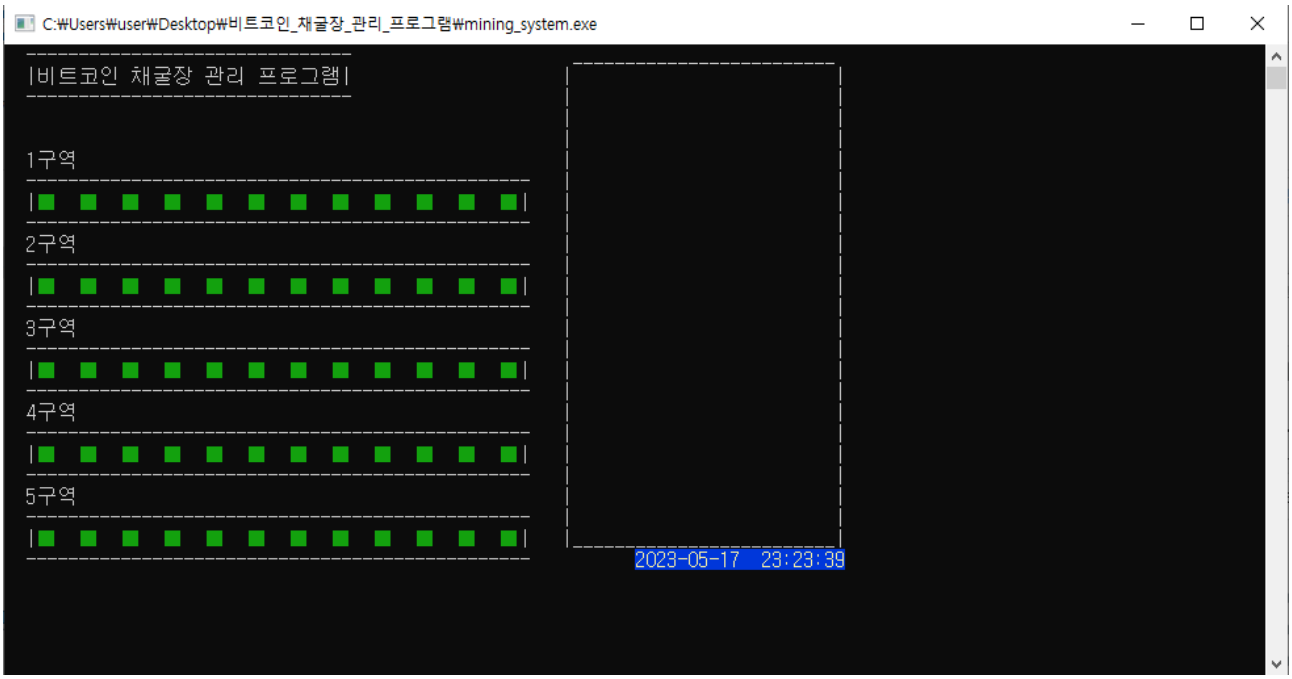
전체적으로 채굴장을 5구역으로 나누어 총 60개 채굴기를 관리하도록 하였다. 초기 화면은 현재 작동 중인 TASK에 대한 정보를 알려주고 60개의 채굴기가 정상상태로 초록색이며 알림 문구는 아직 고장이 난 채굴기가 없으므로 출력하지 않는다.

아래는 채굴기 고장 문제로 인해 채굴이 불가능 상태가 된 모습이다.



그림은 1구역, 2구역, 3구역, 4구역, 5구역에 채굴기에 문제가 생겨 빨간색으로 변한 것을 알 수 있고 해당 위치에 대한 알람 문구를 출력하여 사용자에게 알려준다.

아래는 채굴기 고장 상태를 수리하는 TASK를 실행한 후 모습이다.



고장 상태의 채굴기가 정상으로 변경되었고 알람 문구들이 삭제되었다. 이후 이러한 과정이 무한 반복된다.

● 고찰

기존의 C 프로그램은 포크나 쓰레드를 사용하지 않는 이상 하나의 프로세스에서 절차적인 실행으로 쉬운 제어가 가능하였지만, 실시간 임베디드 시스템인 uCOS에서는 다수의 TASK를 생성하고 실행한 경우 각 TASK가 우선순위에 의해 복잡한 방식의 흐름으로 실행되기 때문에 TASK를 제어하기 위해선 테스크의 대기 상태와 다음 우선순위 혹은 공유 자원에 대한 세마포어, TASK 간의 메시지 전달을 위한 메시지 큐를 이용해야 하기 때문에 해당 프로젝트에서는 TASK 설계 과정에서부터 대기상태에 들어갈 조건(메시지 큐를 받기위한 대기 상태, SUSPEND함수, TIMEDELAY함수)과 그에 따른 다음으로 실행할 TASK에 대한 우선순위를 정해 줌으로써 원활한 흐름으로 동작하게 하였고 해당 과정에서 고려하지 못하는 제어 흐름이 있을 수 있으므로 해당 상황이 발생할 때 전역으로 사용하는 구조체와 변수가 영향을 받지 않도록 세마포어를 통한 제어를 추가하였다.