Real Time System Architectures

HOMEWORK 1

FURKAN ÖZEV 161044036

OBJECIVE:

Keep gas temperature and pressure constant.

TASKS:

TASK1 : PRESSURE CONTROL

- ➔ The function of this task is to manage the DAC value given to the Pump according to the pressure value read from the ADC port. The DAC value ranges from 1 to 100.
- ➔ Since this task will run 100 times a second, its period is 10 ms. So, it is periodic task.
- ➔ The ADC Pressure Port is periodically listened to, and the new DAC value is set according to the current pressure value.
- ➔ It is parallel task.

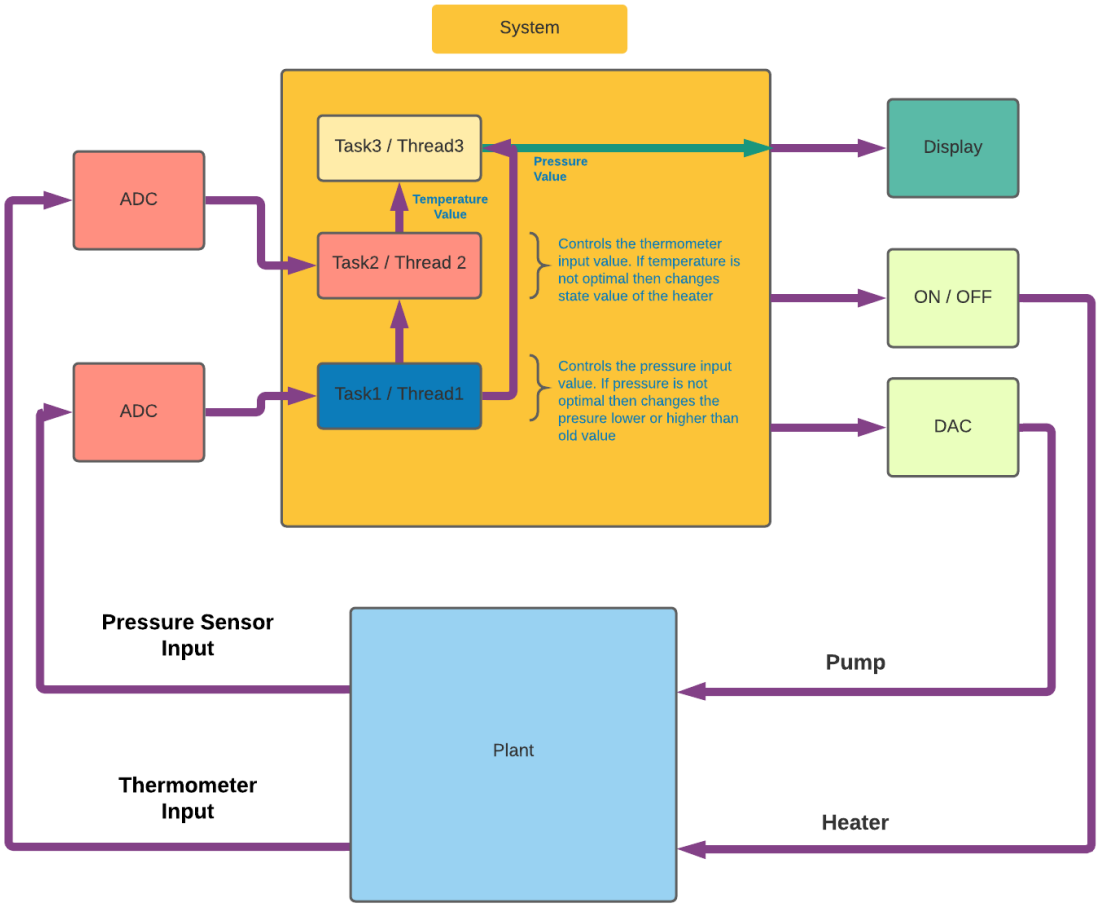
TASK2 : TEMPERATURE CONTROL

- ➔ The function of this task is to manage the Heather button as On / Off according to the thermometer value read from the ADC port.
- ➔ Since this task will run 10 times a second, its period is 100 ms. So, it is periodic task.
- ➔ The ADC Thermometer Port is periodically listened to, and the new Heather buton status is set according to the current temperature value.
- ➔ It is parallel task.

TASK3 : DISPLAY

- ➔ This task function periodically prints the globally defined heater and pump
- ➔ values to the display.
- ➔ Since this task will run 10 times a second, its period is 100 ms. So, it is periodic task.
- ➔ It is parallel task.

BLOCK DIAGRAM:



CODES:

```cpp
#include <iostream>
#include <thread>
#include <mutex>

#define PRESSURE_PIN ?;
#define THERMOMETER_PIN ?;

using namespace std;

class Controller{

    private:
        mutex pressureMutex;
        mutex temperatureMutex;
        int currentPressure;
        int currentTemperature;

        void adc_trigger(int port);
        void read_adc(int port,int &value);
        void write_dac(int value);
        void write_switch(bool value);
        bool newTemperature(int temperature);
        int newPressure(int pressure);
        void currentTime();

    public:
        Controller();
        void pressureTask();
        void temperatureTask();
        void displayTask();
};

void Controller::adc_trigger(int port) { /* It was considered applied */ }
void Controller::read_adc(int port,int &value){ /* It was considered applied */ }
void Controller::write_dac(int value){ /* It was considered applied */ }
void Controller::write_switch(bool value){ /* It was considered applied */ }

// Controls and returns the new pressure value
int Controller::newPressure(int pressure){
    int new_pressure;

    if(pressure >= upper_bound){
        new_pressure = pressure-(pressure-upper_bound);
    }
    else if(pressure <= lower_bound){
        new_pressure = pressure+(lower_bound-pressure);
    }
    return new_pressure;
}

// Controls and returns the new heater state
bool Controller::newTemperature(int temperature){
    bool heaterStatus;
    if(temperature >= upper_bound)
        heaterStatus = false;

    else if(temperature <= lower_bound)
        heaterStatus = true;

    return heaterStatus;
}

// Gets the current system time
void Controller::currentTime(){ /* It was considered applied */ }

// Contrustor of the Controller class
Controller::Controller(){ /* Intentionally empty */ }
```

```
// Task1: Pressure Control
void Controller::pressureTask(){
    int end_time, pressure, start_time;

    for(;;){
        start_time = this.currentTime();

        // Trigger ADC to take pressure value
        this.adc_trigger(PRESSURE_PIN);

        // Read the current pressure value
        this.read_adc(PRESSURE_PIN, pressure);

        // Locks the mutex
        this.pressureMutex.lock();

        // Control the pressure value
        this.currentPressure = this.newPressure(pressure);

        // Unlocks the mutex
        this.pressureMutex.unlock();

        // Write this value to DAC
        this.write_dac(pressure);

        end_time = this.currentTime();

        // Sleep to wait ADC trigger (100 Hz = 10 ms)
        sleep(10 - (end_time - start_time));
    }
}
```

```
// Task2: Temperature Control
void Controller::temperatureTask(){
    int end_time, temperature, start_time;
    bool state;

    for(;;){
        start_time = this.currentTime();

        // Trigger ADC to take temperature value
        this.adc_trigger(TEMPERATURE_PIN);

        // Read the current temperature value
        this.read_adc(TEMPERATURE_PIN, temperature);

        // Locks the mutex
        this.temperatureMutex.lock();

        // Control the temperature value
        this.currentTemperature = temperature;
        state = this.newTemperature(temperature);

        // Unlocks the mutex
        this.temperatureMutex.unlock();

        // Write this value to switch
        this.write_switch(state);

        end_time = this.currentTime();

        // Sleep to wait ADC trigger (1000 Hz = 100 ms)
        sleep(100 - (end_time - start_time));
    }
}
```

```
// Task3: Display
void Controller::displayTask(){
    int end_time, start_time;

    for(;;){
        start_time = this.currentTime();

        // Locks the mutex
        this.temperatureMutex.lock();

        // Prints the temperature value
        cout<<this.currentTemperature<<endl;

        // Unlocks the mutex
        this.temperatureMutex.unlock();

        // Locks the mutex
        this.pressureMutex.lock();

        // Prints the pressure value
        cout<<this.currentPressure<<endl;

        // Unlocks the mutex
        this.puressure_mutex.unlock();

        end_time = this.currentTime();

        // Sleep to wait ADC trigger (100 Hz = 10 ms)
        sleep(10 - (end_time - start_time));
    }
}
```

```
int main (){
    Controller obj;

    thread th1 (obj.pressureTask);
    thread th2 (obj.temperatureTask);
    thread th3 (obj.displayTask);

    th1.join();
    th2.join();
    th3.join();

    return 0;
}
```