CSE437 - HOMEWORK 1

Fatih Selim YAKAR - 161044054

Objective

Definition

Keeping gas temperature and pressure constant.

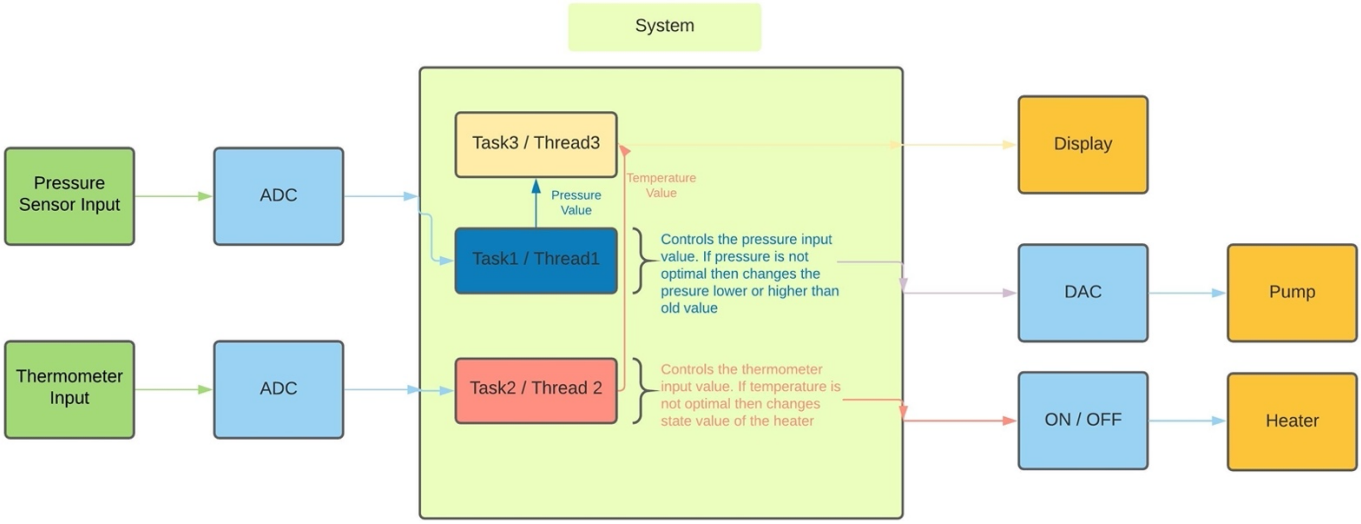
Tasks

- **Task1 / Thread1 (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 port.
    - Periodic (10 ms)
    - Parallel
- **Task2 / Thread2 (Temperature Control)**
    - The function of this task is to manage the Heater ON / OFF value according to the thermometer value read from the port.
    - Periodic (100 ms)
    - Parallel
- **Task3 / Thread3 (Display)**
    - This task function periodically prints the globally defined heater and pump values to the display.
    - Periodic (10 ms)
    - Parallel

Synchronization and Communication

These 3 tasks run parallel. Task1 and Task2 change the private data variable temperature and pressure value, Task3 displays these values. I set the period of Task3 to minimum period value which is th period of the pressure control. In order to properly set and get these private values I mentioned, I used mutex for pressure and temperature. Finally, I ran these tasks in 3 threads.

Block Diagram



Pseudocodes (Based on C++ language)

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

#define PRESSURE_PIN ?;
#define THERMOMETER_PIN ?;

using namespace std;

class Plant_control{

        private:
                mutex pressure_mutex;
                mutex temperature_mutex;
                int current_pressure_value;
                int current_temperature_value;

                void adc_trigger(int port);
                void read_adc(int port,int &value);
                void write_dac(int value);
                void write_switch(bool value);
                bool control_and_return_temperature(int current_temperature);
                int control_and_return_pressure(int current_pressure);
                void get_current_time();
```

```
        public:
                Plant_control();
                void control_pressure_task();
                void control_temperature_task();
                void display_task();

};

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

// Controls and returns the new heater state
bool Plant_control::control_and_return_temperature(int current_temperature){
        bool heater_on;
        if(current_temperature>=upper_bound){
                heater_on=false;
        }
        else if(current_temperature<=lower_bound){
                heater_on=true;
        }
        return heater_on;
}

// Controls and returns the new pressure value
int Plant_control::control_and_return_pressure(int current_pressure){
        int new_pressure;
        if(current_pressure>=upper_bound){
                new_pressure=current_pressure-(current_pressure-upper_bound);
        }
        else if(current_pressure<=lower_bound){
                new_pressure=current_pressure+(lower_bound-current_pressure);
        }
        return new_pressure;
}

// Gets the current system time
void Plant_control::get_current_time(){ /* It was considered applied */ }
```

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

// Task1
void Plant_control::control_pressure_task(){
        int time_difference,end_time,current_pressure,start_time,new_pressure;
        for(;;){
                start_time = this.get_current_time();

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

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

                // Locks the mutex
                this.pressure_mutex.lock()

                // Control the pressure value
                this.current_pressure_value = current_pressure;
                new_pressure = this.control_and_return_pressure(current_pressure);

                // Unlocks the mutex
                this.pressure_mutex.unlock()

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

                end_time = this.get_current_time();

                // Sleep to wait ADC trigger (100 Hz = 10 ms)
                time_difference = end_time – start_time;
                if(time_difference>10){
                        cerr<<"control_pressure_task l deadline miss: ";
                        cerr<<time_difference-10<<"ms exceeded.";
                }
                else{
                        sleep(10 – time_difference);
                }
        }
}

// Task2
void Plant_control::control_temperature_task(){
```

```
        int time_difference,end_time,current_temperature,start_time;
        bool heater_state;
        for(;;){
                start_time = this.get_current_time();

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

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

                // Locks the mutex
                this.temperature_mutex.lock()

                // Control the temperature value
                this.current_temperature_value = current_temperature;
                heater_state=this.control_and_return_temperature(current_temperature);

                // Unlocks the mutex
                this.temperature_mutex.unlock()

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

                end_time = this.get_current_time();

                // Sleep to wait ADC trigger (1000 Hz = 100 ms)
                time_difference = end_time – start_time;
                if(time_difference>100){
                        cerr<<"control_temperature_task l deadline miss: ";
                        cerr<<time_difference-100<<"ms exceeded.";
                }
                else{
                        sleep(100 – time_difference);
                }
        }
}

// Task3
void Plant_control::display_task(){
        int time_difference,end_time,start_time;

        for(;;){
                start_time=this.get_current_time();
```

```
                // Locks the mutex
                this.temperature_mutex.lock();

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

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

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

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

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

                end_time=this.get_current_time();

                // Sleep to wait ADC trigger (100 Hz = 10 ms)
                time_difference = end_time – start_time;
                if(time_difference>10){
                        cerr<<"display_task l deadline miss: ";
                        cerr<<time_difference-10<<"ms exceeded.";
                }
                else{
                        sleep(10 – time_difference);
                }
        }
}

int main (){
        Plant_control obj;
        std::thread th1 (obj.control_pressure_task);
        std::thread th2 (obj.control_temperature_task);
        std::thread th3 (obj.display_task);
        th1.join();
        th2.join();
        th3.join();
        return 0;
}
```