Alarme PIR-RFID

Alec Coelho

Novembro 2023

1 Introdução

O projeto consistirá, como mencionado no título, em um sistema de segurança que, ao detectar uma entrada não autorizada, soará um alarme. Para realizar o projeto, será utilizado um sensor de presença por infravermelho (PIR), um módulo leitor de radiofrequência (RFID) com chip MFRC522 além de LEDs e um buzzer. Estes periféricos estarão ligados à um microcontrolador ESP32, também conectado ao computador do usuário ou alimentado por bateria externa.

2 Classes

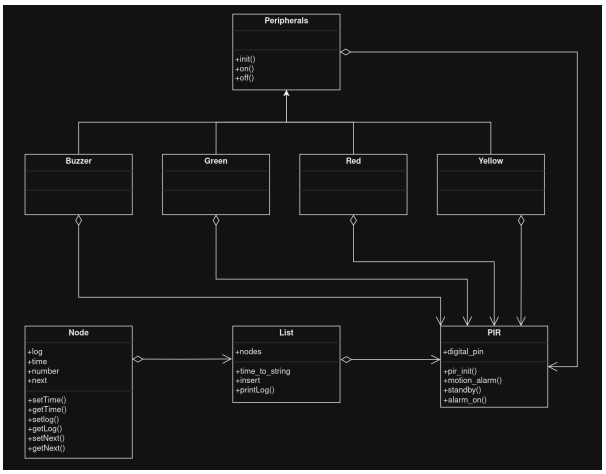

Figura 1: Diagrama de classes

Utilizando a linguagem de programação C++ com orientação a objetos, o software embarcado utiliza-se de diversas classes que representam, de forma geral, os periféricos e sensores utilizados, além das classes utilizadas para registro (logs). Primeiramente, a classe *Peripherals* foi utilizada como uma classe virtual herdada por todos os periféricos, que consistem em 3 LEDs e um buzzer. E possui apenas 3 métodos: *init*, *on* e *off*. Estes métodos são responsáveis, respectivamente, por inicializar, ligar e desligar o periférico de acordo com o pino definido em cada classe derivada.

Seguidamente, as classes "Nodo"e "*List*"foram utilizadas com o intuito de registrar cada entrada detectada pelo sensor de presença, seja esta autorizada ou não. Cada objeto da classe "Nodo"representa um destes registros. A classe *List* agregará variáveis do tipo nodo, mas terá apenas um objeto criado, e este será responsável por encadear os nodos contendo os registros, além de imprimi-los na porta serial quando solicitado.

Por fim, a classe "PIR"é a mais volumosa em termos de linhas de código, pois esta representa o sensor de movimento, que será responsável por administrar o uso dos periféricos e da lista de registros, agregando todas as outras classes no programa (exceto por "Nodo"). Seus métodos incluem uma função para inicialização, *standby* (declarado como *friend*), *motion alarm* e *alarm on*, explicados de forma mais detalhada na sessão abaixo.

3 Funcionamento

Ao energizar o ESP32 através de sua entrada USB ou pinos "VCC"e "GND", o programa será iniciado, e automaticamente entrará em modo de *stand-by* chamando o método de PIR com este mesmo nome.

Desta forma, o programa manterá o LED amarelo aceso até que seja detectado um cartão magnético no módulo RFID. Caso esta detecção ocorra, o sistema entrará em modo de ativação do sensor PIR, onde permacerá por 2 minutos, imprimindo o tempo restante a cada 10 segundos e piscando o LED verde a cada segundo (Caso seja detectado um cartão magnético novamente dentre o intervalo de 2 minutos, a ativação será cancelada e o sistema retornará para o modo de *stand-by*). Ao fim desta contagem, o LED verde permanecerá ligado de forma constante, indicando que o método *motion alarm* foi chamado. Durante a execução desta tarefa, o sistema mantém uma leitura constante da porta digital do sensor PIR. Caso haja tensão fornecida pelo dispositivo, indicando alguma detecção de movimento, uma nova contagem de 10 segundos será iniciada. Caso algum cartão magético seja detectado pelo módulo RFID antes que a contagem acabe, o movimento detectado será considerado uma entrada autorizada, e o sistema retornará ao modo de *stand-by*. Caso a contagem seja encerrada, esta entrada será considerada como não-autorizada, e o alarme será acionado pelo método *alarm on*, piscando os 3 LEDS juntos de um buzzer soando no mesmo ritmo, e o sistema permanecerá desta forma até que o módulo RFID detecte um cartão magnético.

```
//-------------    LOOP     -------------------------

void loop() {

  standby(); //Waits for card aproximation

  bool pir_activated = motion_sensor.pir_init(PIR_digital);

  if(!pir_activated){
    return;
  }//endif

  Serial.println("SYSTEM ACTIVATED");

  PRESENCE presence = motion_sensor.motion_alarm();
}  //endloop
```

Figura 2: Função *main*

4 Logs

Como requisitado nas especificações do presente projeto, o software implementado conta com a função de armazenar *data logs* na forma de registros de entradas detectadas através das classes "Nodo"e "Lista". De acordo com o que foi mencionado anteriormente, cada nodo possuirá um registro. Este registro conterá a data e hora de uma entrada, um número contendo a ordem cronológica de cada registro, e se esta entrada foi ou não autorizada. Para solicitar ou apagar a lista de registros, basta o usuário enviar os caracteres correspondentes descritos na saída serial enquanto o sistema estiver em modo de *stand-by*. Segue exemplo de impressão da lista:

```
---------- Log entries -------------
[22/11/2023 - 15:33:25] 1. Authorized entrance detected
[22/11/2023 - 15:33:66] 2. UNAUTHORIZED ENTRANCE DETECTED
[22/11/2023 - 15:34:22] 3. Authorized entrance detected
[22/11/2023 - 15:34:36] 4. Authorized entrance detected
Total of entries: 4
-----------------------------------
```

Figura 3: Exemplo de requisição de logs

Há três possibilidades de envio de caracter para solicitação dos *Logs*: "1"para imprimir a lista através da porta serial; "2"para destruir a lista atual e limpar todos os registro; e "3"para imprimir a lista e, após, apagar todos os registros. Estes comandos também serão listados através da porta serial toda vez que o sistema retornar ao modo de *Stand-by*, como demonstrado na figura 4.

```
Stanby mode on.
Aproximate tag/card to activate motion sensor alarm
or enter a character in the serial monitor for log options:
1 - Print log list
2 - Clear log list
3 - Print and clear log list
```

Figura 4: Exemplo de requisição de logs

Para configuração da data e hora dos logs, foram utilizadas as bibliotecas "*chrono*"e "*ctime*". Desta forma, pode-se utilizar das funções *now()* e *to_time_t()* para, respectivamente, obter a hora

atual do sistema no momento de upload do código para o microcontrolador e converter este horário para *time_t*, um *struct* definido dentro da biblioteca *chrono*. Para acessar a data e hora e inseri-los em cada nodo da lista, o método *insert* da classe *List* utiliza ponteiros para acessar as informações em cada endereço do *struct* mencionado, juntá-las e transformá-las em string, para facilitar sua transmissão para porta serial.

```
38  void List::insert(std::string newLog){    //saves a new log into the list
39
40      auto currentTime = std::chrono::system_clock::now();
41      std::time_t time = std::chrono::system_clock::to_time_t(currentTime);
42      std::tm* localTime = std::localtime(&time);
43
44      int year =  localTime->tm_year + 1953;
45      int month =  localTime->tm_mon + first_month;
46      int day =  localTime->tm_mday + first_day;
47      int hour =  localTime->tm_hour + 15;
48      int min =  localTime->tm_min + 32;
49      int sec =  localTime->tm_sec + 17;
50
51      std::string current_time = this->time_to_string(year, month, day, hour, min, sec);
```

Figura 5: Aquisição de data e hora

5 Plano de testes

Para testagem do sistema, este será montado em uma *protoboard*, ligando todos os componentes com *jumpers* e alimentando o ESP32 com um cabo USB conectado a um computador (esta alimentação também pode ser substituída por uma bateria de 3,3 a 5V). Seguidamente, uma função *delay()* já destacada no código fonte(figura 6) do projeto será comentada, para maior agilidade na testagem de todas as funções do sistema.

Após as devidas preparações, será novamente realizado *upload* do código do microcontrolador, e um LED amarelo deverá acender para indicar que o sistema já foi iniciado e está em modo de *Stand-by*. Uma tag e cartão compatíveis com sistemas RFID serão utilizadas para acionamento do módulo MFRC522, alternando os modos de atuação do microcontrolador até que todas as possibilidades sejam testadas. Estas incluem: funcionamento do modo de *Stand-by*; ativação da contagem de ativação do sensor de movimento; retorno ao modo de *Stand-by* caso a contagem seja abortada; ativação do sensor PIR; registro

```
47      if (i%10 == 0){
48      Serial.println(String("Seconds remaining: ") + i);
49      }//endif
50
51
52
53      delay(1000);    //---> DEACTIVATE FOR TESTING
54
55
56
57      if (reader.PICC_IsNewCardPresent()){    //makes so
```

Figura 6: Função *delay()* a ser desativada para testagem

de entradas e respectiva autorização; e, por fim, solicitação dos *Logs*. Além disso, também será utilizado o software CuteCom para visualização de todos os dados enviados para a porta serial do microncontrolador, garantindo o funcionamento correto e visualização dos *Logs*.

6 Conclusão

Tendo em vista todos os aspectos abordados, a utilização de um sensor de presença por infravermelho (PIR), um módulo leitor de radiofrequência (RFID) e outros periféricos, todos controlados por um microcontrolador ESP32, mostrou-se uma abordagem robusta para alcançar os objetivos do projeto.

A estrutura de classes adotada, implementada em C++ com orientação a objetos, proporciona uma organização clara e modular do código, facilitando a manutenção e expansão do sistema. As classes representam adequadamente os diferentes componentes do sistema, desde periféricos até o registro de entradas, permitindo uma compreensão intuitiva da estrutura do software.

A capacidade de entrar em modo de stand-by, ativar o sensor PIR após a detecção de um cartão RFID autorizado e acionar o alarme em caso de entrada não autorizada demonstra a aplicabilidade prática do sistema em ambientes de segurança.

A implementação dos logs proporciona uma funcionalidade valiosa, permitindo o registro e monitoramento das entradas detectadas. A capacidade de so-

licitar, apagar e visualizar os logs fornece ao usuário controle e transparência sobre as atividades registradas pelo sistema.

Em suma, o projeto atingiu seus objetivos ao criar um sistema de segurança que combina eficácia na detecção de entradas não autorizadas com uma interface intuitiva para o usuário. Possíveis melhorias e expansões podem ser consideradas para futuras iterações do projeto, incluindo a integração de tecnologias adicionais e aprimoramentos na interface do usuário.

Toda a documentação e código do presente projeto encontra-se disponível em: https://github.com/Coelho50/EmbeddedCppProject