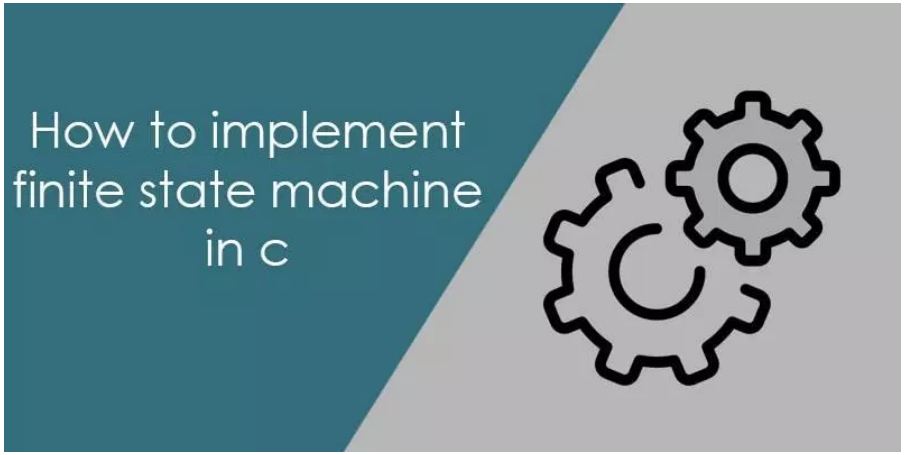
How to implement finite state machine in c

BY **AMLENDRA** ON

15



Nowadays many applications either small or complex use the finite state machine (FSM). A finite state machine in c is one of the popular design patterns for the embedded system.A finite state machine makes the development easy and smooth.

There are a lot of devices which use event base states, like coffee machine, vending machine, POS devices, door lock system etc. Some POS devices are used the event

Categories

8051 Micro-controller
batch file
C Language
C++ Language
communication protocol
Operating System
Uncategorized
Windows Driver

table in which events are registered with an event handler.This event handler executes when the relevant events come.

A finite state machine can have multiple states, it can switch from one state to another state on the basis of internal or external input.This input could be timer expiry signal, hardware or software interrupt .. etc. In the finite state machine, the procedure to change one state to another state is called transition.

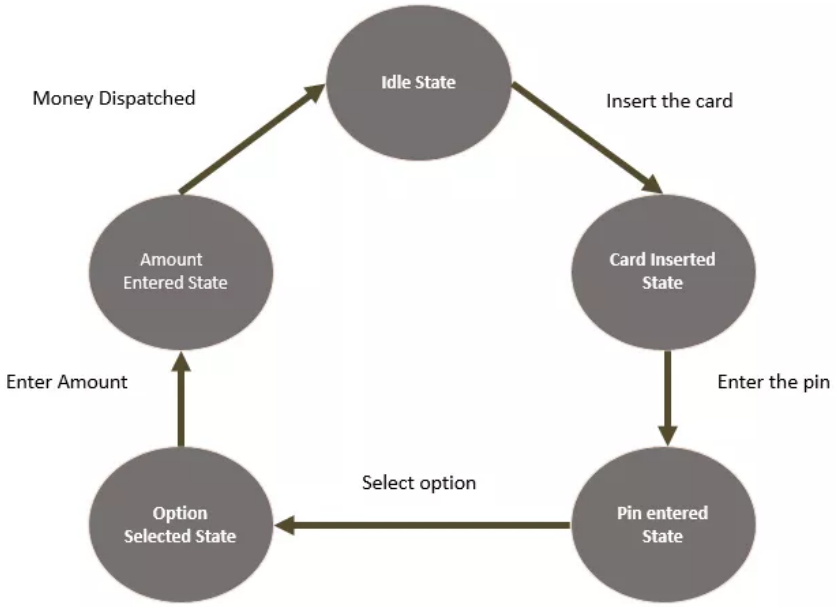
In this article, I will describe some approaches for implementing a state machine in C.

For example, I am considering ATM machine and creating its sample state machine in C. The state of ATM machine could be changed through the coming events.I have mentioned below the sample stats of ATM machine.

The Sample States of the ATM machine.

- Idle State
- Card Inserted State
- Pin entered State
- Option Selected State
- Amount Entered State

Initially, ATM machine would be in the Idle state, When a user inserts the card then it change their state and processes the card. After the card processing, ATM again changes their state and ask the user to enter the pin number.When the user entered the pin then it asks for choice (Balance inquiry, withdrawal, Deposit) and after that change the state and ask to enter the amount and dispatch the entered amount.



Above figure describe the states of the ATM machine.

Recommended steps to create the state machine

- Gather the information which user wants.
- Analyze the all gather information and sketch the state transition diagram.
- create a code skeleton of the state machine.
- Make sure the transition (changing state) work properly
- Implement the all the required information in code skeleton of the state machine.
- Test the implemented state machine.

There are two most popular approaches for implementing an event-based state machine in C. The selection of both approaches depends on requirement and situation.

- Using the conditional statement (**nested switch** or nested if-else).
- Using the lookup table

Using the conditional statement

This is the simplest way to implement the state machine. We have used if-else or the switch case to check the states and triggered event. If the combination of states and triggered event match, execute the event handler to serve the service and update the next state. It depends on a requirement which checks first states or the event.

In below sample code, I am verifying the states first and after that checks the triggered event.If you want you can reverse the procedure that means you can check event first and after that checks the states.

```
#include <stdio.h>

//Different state of ATM machine
typedef enum
{
 Idle_State,
 Card_Inserted_State,
 Pin_Eentered_State,
 Option_Selected_State,
 Amount_Entered_State,


}eSystemState;

//Different type events
typedef enum
{

 Card_Insert_Event,
 Pin_Enter_Event,
 Option_Selection_Event,
 Amount_Enter_Event,
 Amount_Dispatch_Event

}eSystemEvent;

//Prototype of eventhandlers
eSystemState AmountDispatchHandler(void)
{
 return Idle_State;
}
eSystemState EnterAmountHandler(void)
{
 return Amount_Entered_State;
}
```

```
eSystemState OptionSelectionHandler(void)
{
 return Option_Selected_State;
}


eSystemState EnterPinHandler(void)
{
 return Pin_Eentered_State;
}


eSystemState InsertCardHandler(void)
{

 return Card_Inserted_State;
}




int main(int argc, char *argv[]) {

 eSystemState eNextState = Idle_State;
 eSystemEvent eNewEvent;

 while(1)
 {
 //Read system Events
 eSystemEvent eNewEvent = ReadEvent();

 switch(eNextState)
 {
 case Idle_State:
 {
 if(Card_Insert_Event == eNewEvent)
 {
 eNextState = InsertCardHandler();
 }
 }
 break;
 case Card_Inserted_State:
 {
 if(Pin_Enter_Event == eNewEvent)
 {
 eNextState = EnterPinHandler();
 }
 }
 break;
 case Pin_Eentered_State:
 {
 if(Option_Selection_Event == eNewEvent)
 {
 eNextState = OptionSelectionHandler();
 }
 }
 break;
 case Option_Selected_State:
 {

 if(Amount_Enter_Event == eNewEvent)
 {
```

```
eNextState = EnterAmountHandler();
}
}
break;
case Amount_Entered_State:
{
if(Amount_Dispatch_Event == eNewEvent)
{
eNextState = AmountDispatchHandler();
}
}
break;
default:
break;

}
}



return 0;
}
```

Using the lookup table

A lookup table is also a very good technique to implement the state machine. Using the c language we can implement a lookup table in many ways. In below section, I am describing some ways to implement the state machine using the **function pointer** and lookup table.

A state machine in c using a 2D array

We will create a **2D array** containing the function pointers. In which rows and columns represented by the states and events of the finite state machine. This 2D **array** initializes using the designated initializer.

It is the simplest way to implement the state machine, using this technique we can reduce the length of the code. The most important feature of this technique in future if you want to add any new states or events, we can easily integrate with it without any huge hurdle.

Let's see an example,

```
#include <stdio.h>

//Different state of ATM machine
typedef enum
{
```

```
 Idle_State,
 Card_Inserted_State,
 Pin_Eentered_State,
 Option_Selected_State,
 Amount_Entered_State,
 last_State


}eSystemState;

//Different type events
typedef enum
{

 Card_Insert_Event,
 Pin_Enter_Event,
 Option_Selection_Event,
 Amount_Enter_Event,
 Amount_Dispatch_Event,
 last_Event

}eSystemEvent;


//typedef of 2d array
typedef eSystemState (*const afEventHandler[last_State][last_Event])(void);

//typedef of function pointer
typedef eSystemState (*pfEventHandler)(void);



//function call to dispatch the amount and return the ideal state
eSystemState AmountDispatchHandler(void)
{

 return Idle_State;
}


//function call to Enter amount and return amount enetered state
eSystemState EnterAmountHandler(void)
{

 return Amount_Entered_State;
}


//function call to option select and return the option selected state

eSystemState OptionSelectionHandler(void)
{

 return Option_Selected_State;
}


//function call to enter the pin and return pin entered state
eSystemState EnterPinHandler(void)
{
```

```
 return Pin_Eentered_State;
}


//function call to processing track data and return card inserted state
eSystemState InsertCardHandler(void)
{

 return Card_Inserted_State;
}



int main(int argc, char *argv[]) {

 eSystemState eNextState = Idle_State;
 eSystemEvent eNewEvent;

// Table to define valid states and event of finite state machine
 static afEventHandler StateMachine = {

 [Idle_State] ={[Card_Insert_Event]= InsertCardHandler },
 [Card_Inserted_State] ={[Pin_Enter_Event] = EnterPinHandler },
 [Pin_Eentered_State] ={[Option_Selection_Event] = OptionSelectionHandler},
 [Option_Selected_State] ={[Amount_Enter_Event] = EnterAmountHandler},
 [Amount_Entered_State] ={[Amount_Dispatch_Event] = AmountDispatchHandler},

 };

 while(1)
 {
 // assume api to read the next event
 eSystemEvent eNewEvent = ReadEvent();

 if( ( eNextState < last_State) && (eNewEvent < last_Event) && StateMachine[eNext
 {
 // function call as per the state and event and return the next state of the fin
 eNextState = (*StateMachine[eNextState][eNewEvent])();
 }
 else
 {
 //Invalid
 }


 }

 return 0;
}
```

One thing needs to remember, here table is sparse, if the states and events are increasing, this technique increases the wastage of the memory. So before creating the state machine diagram we need to account all the things very precisely at the beginning of the design.

State machine using an array of structure

This is an elegant way to create the finite state machine. The states and events of the state machine are encapsulated in a **structure with function pointer** (Event handler) call at the proper state and event.

```
#include <stdio.h>

//Different state of ATM machine
typedef enum
{
    Idle_State,
    Card_Inserted_State,
    Pin_Eentered_State,
    Option_Selected_State,
    Amount_Entered_State,
    last_State

}eSystemState;

//Different type events
typedef enum
{

    Card_Insert_Event,
    Pin_Enter_Event,
    Option_Selection_Event,
    Amount_Enter_Event,
    Amount_Dispatch_Event,
    last_Event

}eSystemEvent;

//typedef of function pointer
typedef eSystemState (*pfEventHandler)(void);
```

```c
//structure of state and event with event handler
typedef struct
{
    eSystemState eStateMachine;
    eSystemEvent eStateMachineEvent;
    pfEventHandler pfStateMachineEvnentHandler;

}sStateMachine;

//function call to dispatch the amount and return the ideal state
eSystemState AmountDispatchHandler(void)
{

    return Idle_State;
}


//function call to Enter amount and return amount enetered state
eSystemState EnterAmountHandler(void)
{

    return Amount_Entered_State;
}


//function call to option select and return the option selected state

eSystemState OptionSelectionHandler(void)
{

    return Option_Selected_State;
}


//function call to enter the pin and return pin entered state
eSystemState EnterPinHandler(void)
{

    return Pin_Eentered_State;
}


//function call to processing track data and return card inserted state
eSystemState InsertCardHandler(void)
{

    return Card_Inserted_State;
}


//Initialize array of structure with states and event with proper handler
sStateMachine asStateMachine [] = {

    {Idle_State,Card_Insert_Event,InsertCardHandler},

    {Card_Inserted_State,Pin_Enter_Event,EnterPinHandler},

    {Pin_Eentered_State,Option_Selection_Event,OptionSelectionHandler},

    {Option_Selected_State,Amount_Enter_Event,EnterAmountHandler},

    {Amount_Entered_State,Amount_Dispatch_Event,AmountDispatchHandler}
```

```
};


//main function
int main(int argc, char *argv[]) {

    eSystemState eNextState = Idle_State;

    while(1)
    {
     //Api read the event
        eSystemEvent eNewEvent = read_event();

        if((eNextState < last_State) && (eNewEvent < last_Event)&& (asStateMachin
        {
         // function call as per the state and event and return the next state of
            eNextState = (*asStateMachine[eNextState].pfStateMachineEvnentHandler
        }
        else
        {
            //Invalid
        }

    }

    return 0;
}
```

Share this:

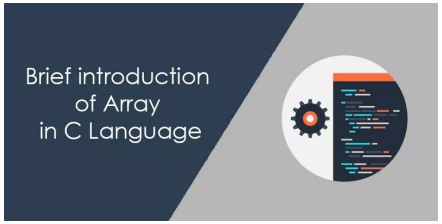
THIS ENTRY WAS POSTED IN **C LANGUAGE**. BOOKMARK THE **PERMALINK**.

About Amlendra

I am an embedded c software engineer and a corporate trainer, currently, I am working as senior software engineer in a largest Software consulting company . I have working experience of different microcontrollers (stm32, LPC, PIC AVR and 8051), drivers (USB and virtual com-port), POS device (VeriFone) and payment gateway (global and first data).

WEBSITE

← **PREVIOUS ARTICLE**



array in c/c++language,Brief introduction

AMLENDRA/

NEXT ARTICLE →

What is flexible array member in c?

AMLENDRA/

15 comments

John

AUGUST 16, 2017 AT 6:30 PM

Nice article on the basics. Thank you

REPLY

Amlendra

AUGUST 17, 2017 AT 3:01 AM

Thank you John..

REPLY

Akhi

AUGUST 18, 2017 AT 2:59 AM

eSystemEvent eNewEvent = ReadEvent();
Can you explain this line?

REPLY

Amlendra

AUGUST 18, 2017 AT 3:47 PM

In this article ReadEvent() behaves like the third party or standard library which returns the current occurred event. For example, In POS device we have read function which when you pressed the key, swipe the card or insert the card then it returns relevant event value. On the basis of this event value, you call the relevant function, like when you swipe the card in POS device then an event value (MAG_CARD_EVENT) return by the read function.

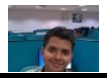
REPLY

Akhi

AUGUST 18, 2017 AT 10:28 PM

Thank you.
So, basically, this function will vary for each program?

REPLY



Amlendra

AUGUST 19, 2017 AT 1:49 AM

It is not vary for program but vary with device, each device have their own API to read the events.

REPLY

Akhi

AUGUST 19, 2017 AT 3:07 AM

got it. Thank you

Akhilesh Gangwar

SEPTEMBER 5, 2017 AT 1:56 PM

From where can I learn these type of embedded programming?
Some embedded applications include making task array and then their functionality is similar to an operating system. How can I learn these types of the programs?

REPLY

Amlendra

SEPTEMBER 6, 2017 AT 4:02 AM

There are a lot of magazines which is published the article regarding the Finite state machine.

REPLY

Harish Reddy

NOVEMBER 24, 2017 AT 3:05 PM

What if we have timer based events. For example, upon occurrence "Card_Insert_Event", the state will be changed to "Card_Inserted_State", then let say we start two timers t1 and t2 (t1 < t2). now the expected event is "Pin_Enter_Event" within t1 expires. if t1 expires and no event then we ask "Do you want more time" and if even t2 expires and no event then next state will

be "Idle_state".
Can I know how to add timers and how to handle them. Thank you.

REPLY

Ratheesh

FEBRUARY 2, 2018 AT 5:32 PM

//structure of state and event with event handler
typedef struct
{
eSystemState eStateMachine;
eSystemEvent eStateMachineEvent;
pfEventHandler pfStateMachineEvnentHandler;

}sStateMachine;

do you really need to include the "eSystemState eStateMachine; " in this structure, in fact I don't see you using that any where.

the next state is determined by the array index rather than the "eSystemState eStateMachine; " variable. In the below code, you have placed the array elements aligning with the enumeration values.

sStateMachine asStateMachine [] = {

{Idle_State,Card_Insert_Event,InsertCardHandler},

{Card_Inserted_State,Pin_Enter_Event,EnterPinHandler},

{Pin_Eentered_State,Option_Selection_Event,OptionSelectionHandler},

{Option_Selected_State,Amount_Enter_Event,EnterAmountHandler},

{Amount_Entered_State,Amount_Dispatch_Event,AmountDispatchHandler}

};

I suppose you've created an additional variable just to avoid confusion while creating the table.

REPLY

Mahendra Kumar

APRIL 30, 2018 AT 9:35 PM

thanks

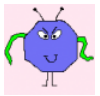
REPLY

sanghita

SEPTEMBER 12, 2018 AT 11:45 AM

I am not able to write the read_event() for my state machine program . Can you please help me

REPLY



FOX

NOVEMBER 30, 2018 AT 9:47 PM

Hi, nice example.
I'm referring now to the last implementation so the "State machine using an array of structure".

Concerning the ATM machine of this example suppose to have to manage more than one single transition from each state.

Suppose to be into the Idle State and no event occur, I suppose to have to add another entry into the asStateMachine, name it with Nothing_To_Do_Event (of course I've also to add it to the eSystemEvent struct):

```
sStateMachine asStateMachine [] = {
{Idle_State ,Nothing_To_Do_Event ,IdleStateHandler},
{Idle_State ,Card_Insert_Event ,InsertCardHandler},
{Card_Inserted_State ,Pin_Enter_Event ,EnterPinHandler},
{Pin_Eentered_State ,Option_Selection_Event ,OptionSelectionHandler},
{Option_Selected_State ,Amount_Enter_Event ,EnterAmountHandler},
```

```
{Amount_Entered_State ,Amount_Dispatch_Event ,AmountDispatchHandler}
};
```

Now when I'm into the main loop I suppose to have also to change the selection logic for the next state because I can have more than a single entry for each state, so the main function has to rewritten in a similar way:

```
int main(int argc, char *argv[])
{
// Init FSM state and event
eSystemState eNextState = Idle_State;

int i=0;

while(1)
{

// ————————————————————————
// FSM core
// ————————————————————————

// Perform the event reading task
eSystemEvent eNewEvent = read_event();

// Check if the state is valid
if ( (eNextState < last_State) && (eNewEvent < last_Event) )
{
// Looking for the correct state-event
for( i=0; i < sizeof(asStateMachine)/sizeof(asStateMachine[0]); i++ )
{
if( (asStateMachine[i].eStateMachine == eNextState) &&
(asStateMachine[i].eStateMachineEvent == eNewEvent) )
{
//state match
eNextState = (*asStateMachine[eNextState].pfStateMachineEventHandler)();
break;
}
}
}
else
{
```

```
//Invalid
}

// ————————————————————
// EXECUTION OF OTHER CODE
// ————————————————————

// … other stuff here …

}

return 0;
}
```

make sense?

Thank!

REPLY

inkyblacksite

FEBRUARY 19, 2019 AT 2:28 PM

I think this line

```
eNextState = (*asStateMachine[eNextState].pfStateMachineEventHandler)();
```

should be

```
eNextState = (*asStateMachine[i].pfStateMachineEventHandler)();
```

Before changing it, you code didn't follow the ATM state transitions.

REPLY

Leave a Reply

Enter your comment here...

Join Aticleworld

You will also get our free C interview questions eBook

Enter your email here*

Subscribe

Pages

- About
- Guest Article
- Blog Posts
- affiliate-disclosure
- disclaimer

Categories

- 8051 Micro-controller
- batch file
- C Language
- C++ Language
- communication protocol
- Operating System
- Uncategorized
- Windows Driver