EVENT MANAGEMENT SYSTEM

ARJUN T N
18
C PROGRAMMING
DATE : 20-07-24

INTRODUCTION

Project Overview

The Auction Management System is designed to facilitate the management of auction items, the bidding process, and the closure of auctions. This system allows users to add items, bid on them, and close auctions effectively

Problem Statement

Traditional auction processes can be difficult and prone to errors, making it difficult to manage items, bids, and closing processes efficiently

Objective

The objective of this project is to create a robust and user friendly auction management system using C programming. This system will allow for the easy addition of items, bidding on available items, and closing of auctions, while ensuring data persistence through file handling

System Requirements

The requirements for executing C programming code are:

➢Hardware Requirement

- A computer with a modern processor (Intel i3 or equivalent)
- At least 2 GB of RAM
- 100 MB of free disk space

➢Software Requirement

- Operating System: Windows, Linux, or MacOS
- Compiler: GCC or any other C compiler
- Text Editor: VS Code, Notepad, Gedit or any other code editor.

Design & Development

Description of the Program logic

The program utilizes a menu-driven interface to interact with users. It allows users to sign in ,log in, add new events ,update existing events etc. Data persistence is achieved through file handling, where events are saved to and loaded from a file.

KEY COMPONENTS

- Structure Definition: Defines the Item structure to store item details.
- File Handling: Functions to save and load data from a binary file.
- Menu-Driven Interface: Provides a user-friendly interface for interaction.

Main Menu: Displays a menu with the options

PSEUDOCODE

```
MAX_EVENTS = 100
MAX_USERS = 50
MAX_USERNAME_LENGTH = 20
MAX_PASSWORD_LENGTH = 20
struct Event {
name: array of characters
 date: array of characters
 description: array of characters
}
struct User {
 username: array of characters
  password: array of characters
}
events: array of Event of size MAX_EVENTS
eventCount: integer initialized to 0
```

users: array of User of size MAX_USERS

userCount: integer initialized to 0

login()

signup()

saveEventsToFile()

loadEventsFromFile()

createEvent()

viewEvents()

updateEvent()

registerParticipants()

```
• main() {
  loadEventsFromFile()
  print "Welcome to Eventify - Your Event Management System!"    loggedIn = 0
 while (not loggedIn) {
     print "\nPlease choose an option:"
      print "1. Log in“
      print "2. Sign up"
        print "3. Exit"
choice = read integer from input
 switch (choice) {
  case 1:              loggedIn = login()
                        break
 case 2:              signup()
                      break
  case 3:              print "\nExiting Eventify. Goodbye!"
          saveEventsToFile()
         exit program
```

```
default:
        print "\nInvalid choice!
       Please enter 1, 2, or 3."       }
  }
while (loggedIn) {
       print "\nMain Menu:"
    print "1. Create a new event"
    print "2. View existing events“
    print "3. Update event details"
    print "4. Register participants for an event"
    print "5. Log out"
    choice = read integer from input
     switch (choice) {
          case 1:              createEvent()
                                 break
          case 2:              viewEvents()
                                 break
```

```
case 3:
            updateEvent()
            break
case 4: registerParticipants()
              break
     case 5:
   print "\nLogging out..."
     loggedIn = 0
       break
    default:
 print "\nInvalid choice!
 Please enter a number between 1 and 5."
}
  }
```

```
saveEventsToFile()
   print "\nGoodbye!“
}
login()
{    username: array of characters
    password: array of characters
   print "\nEnter username:"
   read username from input
  print "Enter password:"
  read password from input
    for i = 0 to userCount - 1 {
    if (username equals
  users[i].username and password
equals users[i].password)
  {          print "\nLogin successful!"
       return 1       }
```

```
    print "\nLogin failed.
 Invalid username or password."
   return 0
}
signup() {
  if (userCount >= MAX_USERS) {
        print "\nCannot sign up more users. User limit reached."
return    }
print "\nEnter new username:"
  read username from input
    users[userCount].username = username
   print "Enter new password:"
 read password from input
 users[userCount].password = password
```

```
 userCount++
 print "\nSign up successful!
 Please log in to continue."}
saveEventsToFile() {
   fp = open file "events.txt" for writing
  if (fp is null) {
print "\nError opening file to save events."
   return
}    for i = 0 to eventCount - 1 {
      fprintf(fp, "%s,%s,%s\n", events[i].name, events[i].date, events[i].description)
}
 close file fp}
loadEventsFromFile() {    fp = open file "events.txt" for reading    if (fp is null) {
print "\nNo events file found. Starting with no events."
    return
 }
```

```
while (read event details from fp) {
       events[eventCount].name = event name
    events[eventCount].date = event date
 events[eventCount].description = event description
      eventCount++
 }    close file fp}createEvent() {
if (eventCount >= MAX_EVENTS) {
       print "\nCannot create more events. Event limit reached."        return
 }
print "\nEnter event details:"
  print "Event name:"
 read event name from input
   events[eventCount].name = event name
```

```
print "Event date (YYYY-MM-DD):"
 read event date from input
  events[eventCount].date = event date
 print "Event description:"
 read event description from input
events[eventCount].description = event description
eventCount++
  print "\nEvent 'events[eventCount - 1].
name' created successfully!“
}viewEvents() {
   if (eventCount == 0) {
      print "\nNo events found."
  return
}    print "\nList of Events:"
   for i = 0 to eventCount - 1 {
   print "\nEvent (i + 1)"
print "Name: events[i].name"
    print "Date: events[i].date"
  print "Description: events[i].description"    }
}
```

```
updateEvent() {
  if (eventCount == 0) {
   print "\nNo events found to update."
return
 }
  print "\nEnter event number to update (1 to eventCount):"
  read eventNumber from input
eventNumber--
  if (eventNumber < 0 or eventNumber >= eventCount) {
   print "\nInvalid event number."
return    }
    print "\nCurrent event details:"
   print "Name: events[eventNumber].name"
   print "Date: events[eventNumber].date"
  print "Description: events[eventNumber].description"
  print "\nEnter new event name:"
```

```
read new event name from input
    events[eventNumber].name = new event name
print "Enter new event date (YYYY-MM-DD):“
    read new event date from input
 events[eventNumber].date = new event date
print "Enter new event description:"
   read new event description from input
    events[eventNumber].description = new event description
    print "\nEvent details updated successfully!“
}registerParticipants() {
   if (eventCount == 0) {
      print "\nNo events found for participant registration."
       return
}    print "\nEnter event number to register participants (1 to eventCount):“
    read eventNumber from input
```

```
eventNumber–
   if (eventNumber < 0 or eventNumber >= eventCount) {
    print "\nInvalid event number."
 return
   }
  print "\nParticipant registration for event
 'events[eventNumber].name':"
  print "Enter participant name:"
   read participantName from input
 print "Participant 'participantName’
 registered successfully for event
'events[eventNumber].name'!"}
```

TEST CASES

Test Cases for Login and Signup

1.Valid Login:

- Provide an existing username and correct password.
- Expected: Successful login message.

2.Invalid Login:

- Provide a non-existent username or incorrect password.
- Expected: Login failed message.

3.Signup with Available Slots:

- Create new username and password within the limits
- Expected: Successful signup message.

4.Signup when User Limit Reached:

- Attempt to create a new user when
- Expected: Message indicating user limit reached, unable to signup.

Test Cases for Event Management

5. Create Event:

- Input valid event name, date, and description.
- Expected: Event creation successful message.

6.Create Event when Event Limit Reached:

- Attempt to create a new event when MAX_EVENTS is already reached.
- Expected: Message indicating event limit reached, unable to create event.

7.View Events:

- Ensure events are correctly loaded and displayed.
- Expected: List of events with their names, dates, and descriptions.

8.Update Event:

- Select an existing event to update.
- Input valid new details for name, date, and description.
- Expected: Event details updated successfully message.

9.Update Event with Invalid Event Number:

- Attempt to update an event with an out-of-range event number.
- Expected: Invalid event number message.

Results

```
Please choose an option:
1. Log in
2. Sign up
3. Exit
2

Enter new username: Arjun
Enter new password: 12345

Sign up successful! Please log in to continue.

Please choose an option:
1. Log in
2. Sign up
3. Exit
1

Enter username:
Arjun
Enter password:
12345

Login successful!

Main Menu:
1. Create a new event
2. View existing events
3. Update event details
4. Register participants for an event
5. Log out
1

Enter event details:
Event name: College Day
Event date (YYYY-MM-DD): 16-07-24
Event description: Main day of college

Event 'College Day' created successfully!
```

Discussion of result

The testing of the event management system confirms that it performs as intended across its main functions. Here are the key takeaways from the results:

1) **Event Registration**
2) **View existing Events**
3) **Event Updation**
4) **Adding participants**

CONCLUSION

Summary of the Project :

This project developed a secure and efficient voting system using C programming. The system allows for:

1) **Event Registration**
2) **View existing Events**
3) **Event Updation**
4) **Adding participants**

Future Enhancement

To improve the system, we can consider the following future enhancements:

- **Artificial Intelligence and Machine Learning Integration:**
- **Predictive Analytics:** Use AI to predict attendee preferences, behavior, and event success metrics based on historical data.
- **Chatbots:** Implement AI-powered chatbots for handling attendee inquiries and providing real-time assistance during events.
- **Augmented Reality (AR) and Virtual Reality (VR) Integration:**
- **Virtual Venue Tours:** Offer virtual tours of event venues to potential clients.
- **AR for Event Navigation:** Use AR to guide attendees through large event venues, providing directions and information overlays.

Appendices: Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_EVENTS 100
#define MAX_USERS 50
#define MAX_USERNAME_LENGTH 20
#define MAX_PASSWORD_LENGTH 20

struct Event {
    char name[50];
    char date[20];
    char description[200];
};

struct User {
    char username[MAX_USERNAME_LENGTH];
    char password[MAX_PASSWORD_LENGTH];
};

struct Event events[MAX_EVENTS];
int eventCount = 0;

struct User users[MAX_USERS];
int userCount = 0;
```

```
int login();
void signup();
void saveEventsToFile();
void loadEventsFromFile();
void createEvent();
void viewEvents();
void updateEvent();
void registerParticipants();

int main() {
    loadEventsFromFile();

    printf("Welcome to Eventify - Your Event Management System!\n");

    int loggedIn = 0;
    while (!loggedIn) {
        printf("\nPlease choose an option:\n");
        printf("1. Log in\n");
        printf("2. Sign up\n");
        printf("3. Exit\n");
```

```
int choice;
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            loggedIn = login();
            break;
         case 2:
            signup();
            break;
         case 3:
            printf("\nExiting Eventify. Goodbye!\n");
            saveEventsToFile();
            exit(0);
         default:
            printf("\nInvalid choice! Please enter 1, 2, or 3.\n");
            break;
      }
   }

   while (loggedIn) {
      printf("\nMain Menu:\n");
      printf("1. Create a new event\n");
      printf("2. View existing events\n");
      printf("3. Update event details\n");
      printf("4. Register participants for an event\n");
      printf("5. Log out\n");
```

```
int choice;
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        createEvent();
        break;
      case 2:
        viewEvents();
        break;
      case 3:
        updateEvent();
        break;
      case 4:
        registerParticipants();
        break;
      case 5:
        printf("\nLogging out...\n");
        loggedIn = 0;
        break;
      default:
        printf("\nInvalid choice! Please enter a number between 1 and 5.\n");
        break;
    }
  }

  saveEventsToFile();
  printf("\nGoodbye!\n");

  return 0;
```

```
int login() {
    char username[MAX_USERNAME_LENGTH];
    char password[MAX_PASSWORD_LENGTH];

    printf("\nEnter username: ");
    scanf("%s", username);
    printf("Enter password: ");
    scanf("%s", password);

    for (int i = 0; i < userCount; i++) {
        if (strcmp(username, users[i].username) == 0 && strcmp(password, users[i].password) == 0) {
            printf("\nLogin successful!\n");
            return 1;
        }
    }

    printf("\nLogin failed. Invalid username or password.\n");
    return 0;
}

void signup() {
    if (userCount >= MAX_USERS) {
        printf("\nCannot sign up more users. User limit reached.\n");
        return;
```

```
    printf("\nEnter new username: ");
    scanf("%s", users[userCount].username);

    printf("Enter new password: ");
    scanf("%s", users[userCount].password);

    userCount++;
    printf("\nSign up successful! Please log in to continue.\n");
}

void saveEventsToFile() {
    FILE *fp = fopen("events.txt", "w");
    if (fp == NULL) {
        printf("\nError opening file to save events.\n");
        return;
    }

    for (int i = 0; i < eventCount; i++) {
        fprintf(fp, "%s,%s,%s\n", events[i].name, events[i].date, events[i].description);
    }

    fclose(fp);
}

void loadEventsFromFile() {
    FILE *fp = fopen("events.txt", "r");
    if (fp == NULL) {
        printf("\nNo events file found. Starting with no events.\n");
        return;
    }

    while (fscanf(fp, "%[^,],%[^,],%[^\n]\n", events[eventCount].name, events[eventCount].date, events[eventCount].description) == 3) {
        eventCount++;
    }

    fclose(fp);
}

void createEvent() {
    if (eventCount >= MAX_EVENTS) {
        printf("\nCannot create more events. Event limit reached.\n");
        return;
    }
```

```
    printf("\nEnter event details:\n");
    printf("Event name: ");
    scanf(" %[^\n]", events[eventCount].name);

    printf("Event date (YYYY-MM-DD): ");
    scanf(" %s", events[eventCount].date);

    printf("Event description: ");
    scanf(" %[^\n]", events[eventCount].description);

    eventCount++;
    printf("\nEvent '%s' created successfully!\n", events[eventCount - 1].name);
}

void viewEvents() {
    if (eventCount == 0) {
        printf("\nNo events found.\n");
        return;
    }

    printf("\nList of Events:\n");
    for (int i = 0; i < eventCount; i++) {
        printf("\nEvent %d\n", i + 1);
        printf("Name: %s\n", events[i].name);
        printf("Date: %s\n", events[i].date);
        printf("Description: %s\n", events[i].description);
    }
}

void updateEvent() {
    if (eventCount == 0) {
        printf("\nNo events found to update.\n");
        return;
    }

    printf("\nEnter event number to update (1 to %d): ", eventCount);
    int eventNumber;
    scanf("%d", &eventNumber);
    eventNumber--; // Adjust for zero-indexed array

    if (eventNumber < 0 || eventNumber >= eventCount) {
        printf("\nInvalid event number.\n");
        return;
    }

    printf("\nCurrent event details:\n");
    printf("Name: %s\n", events[eventNumber].name);
    printf("Date: %s\n", events[eventNumber].date);
    printf("Description: %s\n", events[eventNumber].description);
```

```
printf("\nEnter new event name: ");
    scanf(" %[^\n]", events[eventNumber].name);

    printf("Enter new event date (YYYY-MM-DD): ");
    scanf(" %s", events[eventNumber].date);

    printf("Enter new event description: ");
    scanf(" %[^\n]", events[eventNumber].description);

    printf("\nEvent details updated successfully!\n");
}

void registerParticipants() {
    if (eventCount == 0) {
        printf("\nNo events found for participant registration.\n");
        return;
    }

    printf("\nEnter event number to register participants (1 to %d): ", eventCount);
    int eventNumber;
    scanf("%d", &eventNumber);
    eventNumber--; // Adjust for zero-indexed array

    if (eventNumber < 0 || eventNumber >= eventCount) {
        printf("\nInvalid event number.\n");
        return;
    }

    printf("\nParticipant registration for event '%s':\n", events[eventNumber].name);
    printf("Enter participant name: ");
    char participantName[50];
    scanf(" %[^\n]", participantName);
    printf("Participant '%s' registered successfully for event '%s'!\n", participantName, events[eventNumber].name);
}
```