N. Sai Kiran
03/05/2024

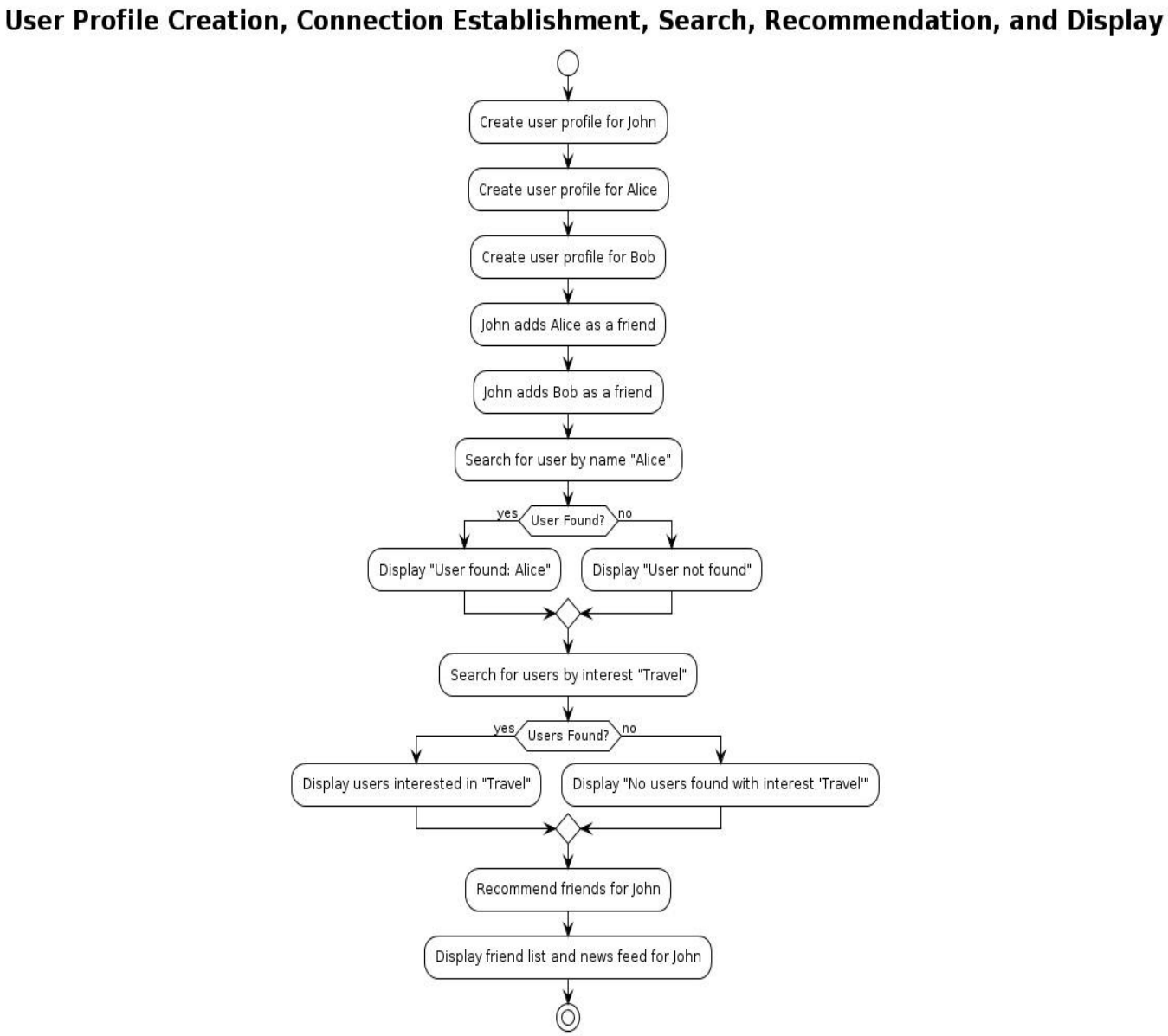
SOCIAL NETWORKING MANAGEMENT

QUESTION:-

Social Network Management System:

Data Structures: Graph (adjacency list) for user connections, linked list for user profiles.
Functionality:
Create user profiles (name, interests, etc.).
Add friends (connect users in the graph).
Search for users by name or interest.
Recommend friends based on mutual connections or interests.
Display a user's friend list and news feed (simulated data or integration with an external API).

FLOWCHART:-


User Profile Creation, Connection Establishment, Search, Recommendation, and Display
Create user profile for John
Create user profile for Alice
Create user profile for Bob
John adds Alice as a friend
John adds Bob as a friend
Search for user by name "Alice"
yes
User Found?
no
Display "User found: Alice"
Display "User not found"
Search for users by interest "Travel"
yes
Users Found?
no
Display users interested in "Travel"
Display "No users found with interest 'Travel'"
Recommend friends for John
Display friend list and news feed for John

MODULAR CODE

network.h

```
#ifndef SOCIAL_NETWORK_H
#define SOCIAL_NETWORK_H

#define MAX_USERS 100
#define MAX_NAME_LENGTH 50
#define MAX_INTEREST_LENGTH 100

    typedef struct User
{
   char name[MAX_NAME_LENGTH];
   char interests[MAX_INTEREST_LENGTH];
   struct User *friends[MAX_USERS];
   int num_friends;
} User;

   typedef struct Graph {
   User users[MAX_USERS];
   int num_users;
```

```
} Graph;



void add_user(Graph *graph, const char *name,
const char *interests);
void add_friendship(Graph *graph, const char
*user1_name, const char *user2_name);
void search_users_by_name(Graph *graph, const
char *name);
void search_users_by_interest(Graph *graph, const
char *interest);
void recommend_friends(Graph *graph, const char
*name);
void display_user_friends(User *user);
void display_news_feed(User *user);

#endif
```

Netwok.c

```
#include <stdio.h>
#include <string.h>
#include "social_network.h"

void add_user(Graph *graph, const char
*name, const char *interests) {
    if (graph->num_users >= MAX_USERS) {
        printf("Cannot add more users.\n");
        return;
    }
    User *user = &graph->users[graph-
>num_users++];
    strcpy(user->name, name);
    strcpy(user->interests, interests);
    user->num_friends = 0;
}
```

```
void add_friendship(Graph *graph, const char
*user1_name, const char *user2_name) {
    User *user1 = NULL, *user2 = NULL;
    for (int i = 0; i < graph->num_users; i++) {
        if (strcmp(graph->users[i].name,
user1_name) == 0) {
            user1 = &graph->users[i];
        }
        if (strcmp(graph->users[i].name,
user2_name) == 0) {
            user2 = &graph->users[i];
        }
    }
    if (user1 && user2) {
        user1->friends[user1->num_friends++] =
user2;
        user2->friends[user2->num_friends++] =
user1;
    }
}
```

```
void search_users_by_name(Graph *graph,
const char *name) {
    printf("Users with name '%s':\n", name);
    for (int i = 0; i < graph->num_users; i++) {
        if (strstr(graph->users[i].name, name)) {
            printf("%s\n", graph->users[i].name);
        }
    }
}

void search_users_by_interest(Graph *graph,
const char *interest) {
    printf("Users interested in '%s':\n", interest);
    for (int i = 0; i < graph->num_users; i++) {
        if (strstr(graph->users[i].interests,
interest)) {
            printf("%s\n", graph->users[i].name);
        }
    }
}
```

```
void recommend_friends(Graph *graph, const
char *name) {
   printf("Recommended friends for %s:\n",
name);

 for (int i = 0; i < graph->num_users; i++) {
       if (strcmp(graph->users[i].name, name) ==
0) {
           User *user = &graph->users[i];
           for (int j = 0; j < user->num_friends; j++) {
               User *friend = user->friends[j];
               for (int k = 0; k < friend->num_friends;
k++) {
                   User *potential_friend = friend-
>friends[k];
                   if (strcmp(potential_friend->name,
name) != 0 &&
                       !is_friend_of_user(user,
potential_friend)) {
                       printf("%s\n", potential_friend-
>name);
                   }
```

```
            }
        }
        break;
    }
  }
}

bool is_friend_of_user(User *user, User
*potential_friend) {
  for (int i = 0; i < user->num_friends; i++) {
    if (user->friends[i] == potential_friend) {
      return true;
    }
  }
  return false;
}
```

Networkmain.c

```
                #include <stdio.h>
#include <stdlib.h>
#include "social_network.h"
#include <time.h>
```

```
int main() {
    Graph social_network;
    social_network.num_users = 0;
   // Create users
    add_user(&social_network, "Alice", "coding,
reading");
    add_user(&social_network, "Bob", "sports,
cooking");
    add_user(&social_network, "Charlie",
"coding, gaming");

    // Add friendships
    add_friendship(&social_network, "Alice",
"Bob");
    add_friendship(&social_network, "Bob",
"Charlie");

    // Search for users
    search_users_by_name(&social_network,
"Alice");
```

```
    search_users_by_interest(&social_network,
"coding");

    // Recommend friends
    recommend_friends(&social_network,
"Alice");

    return 0;
}
```

EXPLINATION:-

Each user profile can be represented by a node in a graph.

Additionally, user profile information such as name, interests, etc., can be stored in a linked list or a similar data structure attached to each node.

Graph for User Connections:

The user connections can be represented using a graph data structure with an adjacency list implementation.

Each user is a node in the graph, and edges represent connections (friendships) between users.

The adjacency list for each user node contains references to other users to whom they are connected.

Creating User Profiles:

When creating a user profile, you would add a new node to the graph to represent the user.

The user's profile information can be stored in a linked list or similar data structure attached to the user node.

Adding Friends:

Adding a friend connection between two users involves adding an edge between their respective nodes in the graph.

Searching for Users:

Searching for users by name or interest involves traversing the graph and looking for nodes that match the search criteria.

This can be done using various graph traversal algorithms like depth-first search (DFS) or breadth-first search (BFS).

Recommend Friends:

Friend recommendations can be based on mutual connections or shared interests.

To recommend friends based on mutual connections, you can look for users who are connected to multiple mutual friends.

To recommend friends based on shared interests, you can look for users who have similar interests to the target user.

Displaying Friend List and News Feed:

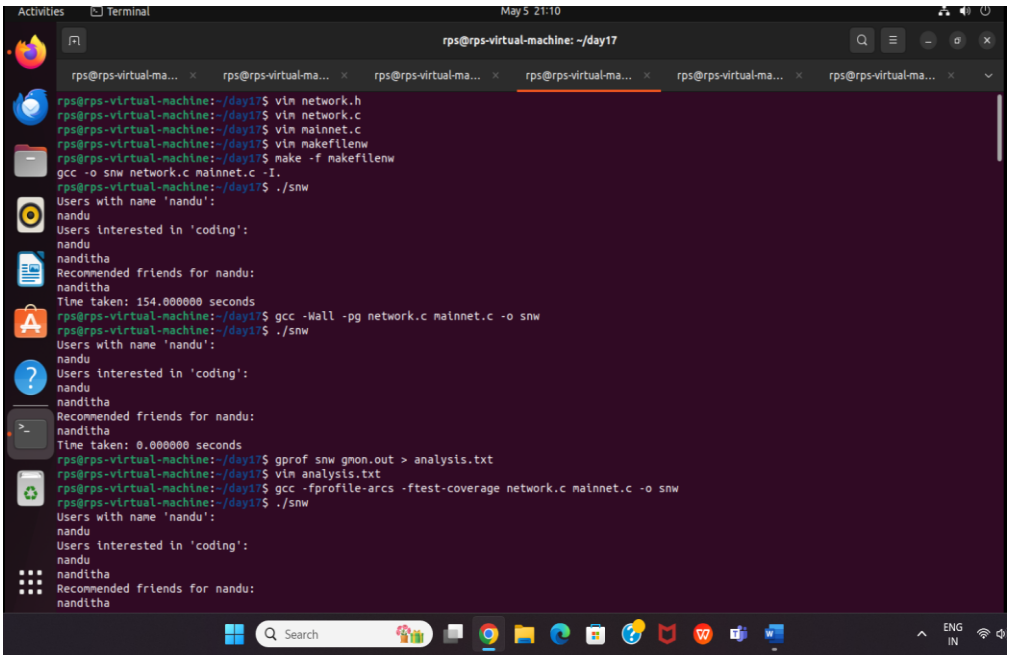
A user's friend list can be obtained by traversing the adjacency list of their node in the graph.

The news feed can be simulated data generated within the system or integrated with an external API to fetch real-time updates from friends.

By implementing these functionalities, you can create a Social Network Management System that allows users to connect with each other,

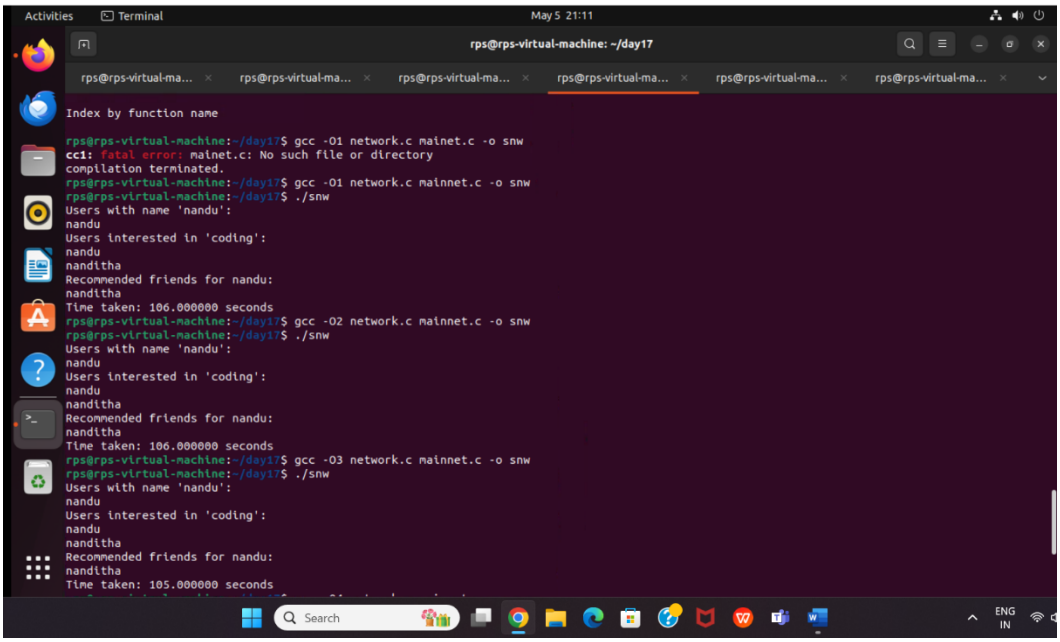
search for other users, receive friend recommendations, and interact with their friends' content.

OUTPUT:-



```
rps@rps-virtual-machine:~/day17$ gcov -b snw-network.c snw-mainnet.c
File 'network.c'
Lines executed:94.00% of 50
Branches executed:100.00% of 36
Taken at least once:80.56% of 36
Calls executed:87.50% of 8
Creating 'network.c.gcov'

File 'mainnet.c'
Lines executed:100.00% of 15
No branches
Calls executed:100.00% of 11
Creating 'mainnet.c.gcov'

Lines executed:95.38% of 65
rps@rps-virtual-machine:~/day17$ gcov -b snw-network.c snw-mainnet.c > coverage.txt
rps@rps-virtual-machine:~/day17$ vim coverage.txt
rps@rps-virtual-machine:~/day17$ gprof snw gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.
```

```
 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) no time propagated

index % time    self  children    called     name

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
```


Activities
Terminal
May 5 21:11
rps@rps-virtual-machine: ~/day17
Index by function name
rps@rps-virtual-machine:~/day17$ gcc -O1 network.c mainet.c -o snw
cc1: fatal error: mainet.c: No such file or directory
compilation terminated.
rps@rps-virtual-machine:~/day17$ gcc -O1 network.c mainnet.c -o snw
rps@rps-virtual-machine:~/day17$ ./snw
Users with name 'nandu':
nandu
Users interested in 'coding':
nandu
nanditha
Recommended friends for nandu:
nanditha
Time taken: 106.000000 seconds
rps@rps-virtual-machine:~/day17$ gcc -O2 network.c mainnet.c -o snw
rps@rps-virtual-machine:~/day17$ ./snw
Users with name 'nandu':
nandu
Users interested in 'coding':
nandu
nanditha
Recommended friends for nandu:
nanditha
Time taken: 106.000000 seconds
rps@rps-virtual-machine:~/day17$ gcc -O3 network.c mainnet.c -o snw
rps@rps-virtual-machine:~/day17$ ./snw
Users with name 'nandu':
nandu
Users interested in 'coding':
nandu
nanditha
Recommended friends for nandu:
nanditha
Time taken: 105.000000 seconds
Search
ENG
IN