Ade suresh

Me24i1030

Question 6

EDSA

ASSIGNMENT -1

 #include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Part a) Request and Dispatch

//function enter into request and dispatch

void requestAndDispatch() {

    printf("Part a) Request and Dispatch \n");

    // Queue implementation

    char\* delivery\_queue[6] = {"Food", "Medicine", "Tools", "Water", "Parts", "Fuel"};

    int front = 0, rear = 5;

    // Stack implementation

    char\* urgent\_stack[6];

    int top = -1;

    // Enqueue all 6 requests

    printf("Enqueued requests: ");

    for (int i = 0; i <= rear; i++) {

        printf("%s ", delivery\_queue[i]);

    }

    printf("\n");

    // Dequeue and push onto stack

    while (front <= rear) {

        urgent\_stack[++top] = delivery\_queue[front++];

    }

    // Pop to show dispatch order

    printf("Dispatch order (LIFO): ");

    while (top >= 0) {

        printf("%s ", urgent\_stack[top--]);

    }

    printf("\n");

    // Creativity Bonus

    printf("Creativity Bonus: LIFO suits urgency because the last received urgent request (like 'Fuel') \n");

    printf("may be most critical (e.g., needed to refuel other drones mid-flight) and should be handled first.\n\n");

}

// Part b) Flight Log Unit

//function enter into flight logunit

void flightLogUnit() {

    printf("Part b) Flight Log Unit\n");

    char\* flight\_log[6];

    int log\_size = 6;

    int current = 0;

    // Insert first 6 deliveries

    for (int i = 0; i < log\_size; i++) {

        char del\_name[10];

        sprintf(del\_name, "Del%d", i+1);

        flight\_log[i] = strdup(del\_name);

        printf("Logged: %s\n", flight\_log[i]);

    }

    // Handle overflow for Del7 and Del8

    for (int i = log\_size; i < log\_size + 2; i++) {

        printf("Archiving oldest: %s\n", flight\_log[0]);

        // Shift all elements left

        free(flight\_log[0]);

        for (int j = 0; j < log\_size - 1; j++) {

            flight\_log[j] = flight\_log[j+1];

        }

        // Add new delivery at end

        char del\_name[10];

        sprintf(del\_name, "Del%d", i+1);

        flight\_log[log\_size-1] = strdup(del\_name);

        printf("Logged: %s\n", flight\_log[log\_size-1]);

    }

    // Creativity Bonus

    printf("Creativity Bonus: Archiving oldest logs helps comply with airspace regulations by \n");

    printf("maintaining a manageable record size while preserving recent activity data for safety audits.\n\n");

}

// Part c) Overloaded Drone Tracker

//function for singly list node

typedef struct SinglyNode {

    char\* drone\_id;

    struct SinglyNode\* next;

} SinglyNode;

//function for doubly list node

typedef struct DoublyNode {

    char\* drone\_id;

    struct DoublyNode\* prev;

    struct DoublyNode\* next;

} DoublyNode;

//function for overloadeDrone Tracker

void overloadedDroneTracker() {

    printf("Part c) Overloaded Drone Tracker \n");

    // Singly linked list for overloaded drones

    SinglyNode\* overloaded\_head = NULL;

    SinglyNode\* current\_singly = NULL;

    // Doubly linked list for serviced drones

    DoublyNode\* serviced\_head = NULL;

    DoublyNode\* serviced\_tail = NULL;

    // Insert Drone3 and Drone6 to singly linked list

    SinglyNode\* drone3 = (SinglyNode\*)malloc(sizeof(SinglyNode));

    drone3->drone\_id = "Drone3";

    drone3->next = NULL;

    SinglyNode\* drone6 = (SinglyNode\*)malloc(sizeof(SinglyNode));

    drone6->drone\_id = "Drone6";

    drone6->next = NULL;

    overloaded\_head = drone3;

    drone3->next = drone6;

    // Print overloaded drones

    printf("Overloaded drones: ");

    current\_singly = overloaded\_head;

    while (current\_singly != NULL) {

        printf("%s ", current\_singly->drone\_id);

        current\_singly = current\_singly->next;

    }

    printf("\n");

    // Remove Drone3 from singly list and add to doubly list

    overloaded\_head = drone6; // Remove Drone3

    DoublyNode\* serviced\_drone3 = (DoublyNode\*)malloc(sizeof(DoublyNode));

    serviced\_drone3->drone\_id = "Drone3";

    serviced\_drone3->prev = NULL;

    serviced\_drone3->next = NULL;

    serviced\_head = serviced\_drone3;

    serviced\_tail = serviced\_drone3;

    // Print serviced drones (forward)

    printf("Serviced drones (forward): ");

    DoublyNode\* current = serviced\_head;

    while (current != NULL) {

        printf("%s ", current->drone\_id);

        current = current->next;

    }

    printf("\n");

    // Print serviced drones (backward)

    printf("Serviced drones (backward): ");

    current = serviced\_tail;

    while (current != NULL) {

        printf("%s ", current->drone\_id);

        current = current->prev;

    }

    printf("\n");

    // Clean up

    free(drone3);

    free(drone6);

    free(serviced\_drone3);

    // Creativity Bonus

    printf("Creativity Bonus: Drone3 overload likely caused by excess Tools payload. \n");

    printf("Fix: Techs recalibrated weight sensors and redistributed load to balance capacity.\n\n");

}

// Part d) Emergency Rerouting

typedef struct CircularNode {

    char\* drone\_id;

    struct CircularNode\* next;

} CircularNode;

//function enter into emergency rerouting

void emergencyRerouting() {

    printf(" Part d) Emergency Rerouting \n");

    CircularNode\* head = NULL;

    CircularNode\* drone1 = (CircularNode\*)malloc(sizeof(CircularNode));

    drone1->drone\_id = "Drone1";

    CircularNode\* drone4 = (CircularNode\*)malloc(sizeof(CircularNode));

    drone4->drone\_id = "Drone4";

    // Create circular list

    head = drone1;

    drone1->next = drone4;

    drone4->next = head; // Circular reference

    // Traverse twice

    printf("Emergency drones (traversed twice): ");

    CircularNode\* current = head;

    for (int i = 0; i < 4; i++) {

        printf("%s ", current->drone\_id);

        current = current->next;

    }

    printf("\n");

    // Clean up

    free(drone1);

    free(drone4);

    // Creativity Bonus

    printf("Creativity Bonus: For Drone4's wind resistance, added aerodynamic shell \n");

    printf("with gyroscopic stabilizers to maintain flight stability during storms.\n");

}

//main function

int main() {

    requestAndDispatch();

    flightLogUnit();

    overloadedDroneTracker();

    emergencyRerouting();

    return 0;

}

REPORT :

Assignment Title:

Cargo Drone Traffic Controller Simulation

Name: ADE.SURESH,

Roll Number: ME24I1030,

Date: 13-04-2024.

Problem Statement

This assignment simulates a cargo drone traffic management system using fundamental data structures in C. The goal is to manage and monitor various operations such as delivery requests, urgent dispatches, flight logging, drone overload tracking, and emergency rerouting. The system should efficiently handle real-world drone logistics scenarios using appropriate structures like queues, stacks, and linked lists.

Key Objectives:

Implement delivery request queuing and urgent dispatch using FIFO and LIFO principles.

Maintain a limited-size flight log, discarding oldest entries when capacity is exceeded.

Track overloaded drones and their servicing using singly and doubly linked lists.

Simulate emergency rerouting with circular linked list traversal.

Showcase creativity and real-world applicability of these operations

DESIGN EXPLANATION

Data Structures Chosen:

Queue (Array) – For enqueuing delivery requests in the order they arrive.

Stack (Array) – To reverse delivery request order for urgent dispatch.

Circular Buffer (Array with shifting) – Used for flight logs with limited memory.

Singly Linked List – To track currently overloaded drones (simple one-way tracking).

Doubly Linked List – For managing serviced drones, allowing forward and backward traversal.

Circular Linked List – For emergency rerouting where drones cycle through predefined paths.

Why These Structures?

Each structure models a specific real-world scenario:

Stacks ensure urgent tasks are handled in reverse order.

Queues represent orderly dispatching.

Linked lists allow dynamic memory use and flexible insertion/removal.

Circular lists simulate continuous routes like emergency patrol paths.

LOGIC OF THE CODE

Part a) Request and Dispatch

Six delivery types are enqueued into a delivery\_queue.

These requests are then moved to a urgent\_stack to reverse order (LIFO).

The stack is popped to show the urgent dispatch sequence.

Part b) Flight Log Unit

A flight log array of size 6 stores delivery IDs (Del1 to Del6).

When new deliveries (Del7, Del8) arrive, the oldest logs are removed using a left shift.

The new deliveries are then appended to the end of the log.

Part c) Overloaded Drone Tracker

A singly linked list stores drones marked as overloaded (e.g., Drone3, Drone6).

Drone3 is then removed from the overloaded list and added to a doubly linked list to indicate it was serviced.

The doubly list is printed in both forward and backward directions.

Part d) Emergency Rerouting

A circular linked list is built using Drone1 and Drone4.

Traversal of the circular list is demonstrated twice to simulate repeated rerouting cycles.

Key Variables and Functions

delivery\_queue[6], urgent\_stack[6]: Arrays used for queue and stack behavior.

flight\_log[6]: Circular buffer to hold limited-size flight logs.

SinglyNode, DoublyNode, CircularNode: Structs used for various linked lists.

requestAndDispatch(), flightLogUnit(), overloadedDroneTracker(), emergencyRerouting(): Modular functions for different simulation tasks.

main(): Runs all the parts in order.

Sample Output (Optional)

Part a) Request and Dispatch

Enqueued requests: Food Medicine Tools Water Parts Fuel

Dispatch order (LIFO): Fuel Parts Water Tools Medicine Food

Creativity Bonus: LIFO suits urgency because the last received urgent request (like 'Fuel')

may be most critical (e.g., needed to refuel other drones mid-flight) and should be handled first.

 Part b) Flight Log Unit

Logged: Del1

Logged: Del2

Logged: Del3

Logged: Del4

Logged: Del5

Logged: Del6

Archiving oldest: Del1

Logged: Del7

Archiving oldest: Del2

Logged: Del8

Creativity Bonus: Archiving oldest logs helps comply with airspace regulations by

maintaining a manageable record size while preserving recent activity data for safety audits.

Part c) Overloaded Drone Tracker

Overloaded drones: Drone3 Drone6

Serviced drones (forward): Drone3

Serviced drones (backward): Drone3

Creativity Bonus: Drone3 overload likely caused by excess Tools payload.

Fix: Techs recalibrated weight sensors and redistributed load to balance capacity.

Part d) Emergency Rerouting

Emergency drones (traversed twice): Drone1 Drone4 Drone1 Drone4

Creativity Bonus: For Drone4's wind resistance, added aerodynamic shell

with gyroscopic stabilizers to maintain flight stability during storms.

 Code Execution Successful