

<b>Name</b>	Manish Shashikant Jadhav
<b>UID no.</b>	2023301005

## Experiment 6

### AIM :

Rominus pizza wants to implement a pizza scheduling system. Create individual systems implementing each of the above CPU scheduling algorithms as follows:

- FCFS: In a pizza delivery system, how does the First-Come-First-Served (FCFS) scheduling algorithm determine the order in which pizzas are delivered to customers?
- SJF: In a pizza delivery system, how does the Shortest Job First (SJF) scheduling algorithm prioritize which pizzas are delivered first based on their preparation time?
- Round Robin: In a pizza delivery system, how does the Round Robin scheduling algorithm ensure that each delivery driver gets a fair and equal chance to deliver pizzas?
- Priority: In a pizza delivery system, how does the Priority scheduling algorithm prioritize which pizzas are delivered first based on the urgency of the delivery (e.g. VIP customers, emergency deliveries)?

Also, explain which algorithm will be best suited for the following scenario. Is it possible to combine the above algorithms to create an even more efficient algorithm? If yes, then explain how?

### Discussion & Output:

#### Program:

```
#include <stdio.h>
#include <stdlib.h>

// Struct to represent a pizza order
typedef struct {
    int id;
    int prep_time; //in minutes
    int delivery_distance; // in kilometers
    int priority; // Priority level I have kept 1 = highest, 5 = lowest
} PizzaOrder;

typedef struct {
    int id;
```

```

    int time_left;
} DeliveryDriver;

// Function prototypes
void fcfs(PizzaOrder orders[], int num_orders);
void sjf(PizzaOrder orders[], int num_orders);
void round_robin(PizzaOrder orders[], int num_orders, int
time_quantum);
void priority(PizzaOrder orders[], int num_orders);

int main() {
    int num_orders, num_drivers, time_quantum;

    // Get number of pizza orders
    printf("Enter the number of pizza orders: ");
    scanf("%d", &num_orders);

    // creating function for allocating memory for pizza orders
    PizzaOrder *orders = malloc(num_orders * sizeof(PizzaOrder));

    // Get pizza order details
    for (int i = 0; i < num_orders; i++) {
        printf("Enter details for pizza order %d:\n", i + 1);
        orders[i].id = i + 1;
        printf("Preparation time (in minutes): ");
        scanf("%d", &orders[i].prep_time);
        printf("Delivery distance (in kilometers): ");
        scanf("%d", &orders[i].delivery_distance);
        printf("Priority level (1 = highest, 5 = lowest): ");
        scanf("%d", &orders[i].priority);
    }

    // number of delivery drivers
    printf("\nEnter the number of delivery drivers: ");
    scanf("%d", &num_drivers);

```

```

    // creating function for allocating memory for delivery drivers
    DeliveryDriver *drivers = malloc(num_drivers *
sizeof(DeliveryDriver));

    for (int i = 0; i < num_drivers; i++) {
        drivers[i].id = i + 1;
        drivers[i].time_left = 0;
    }

    // time quantum for Round Robin scheduling
    printf("\nEnter the time quantum for Round Robin scheduling: ");
    scanf("%d", &time_quantum);

    fcfs(orders, num_orders);
    sjf(orders, num_orders);
    round_robin(orders, num_orders, time_quantum);
    priority(orders, num_orders);

    free(orders);
    free(drivers);

    return 0;
}

//FCFS
void fcfs(PizzaOrder orders[], int num_orders) {
    printf("\nFCFS Scheduling\n");
    for (int i = 0; i < num_orders; i++) {
        printf("Delivering pizza order %d\n", orders[i].id);
    }
}

// Shortest job first
void sjf(PizzaOrder orders[], int num_orders) {
    printf("\nSJF Scheduling\n");
    for (int i = 0; i < num_orders - 1; i++) {

```

```

        for (int j = 0; j < num_orders - i - 1; j++) {
            if (orders[j].prep_time + orders[j].delivery_distance >
                orders[j + 1].prep_time + orders[j +
1].delivery_distance) {
                // Swap orders
                PizzaOrder temp = orders[j];
                orders[j] = orders[j + 1];
                orders[j + 1] = temp;
            }
        }
    }
    for (int i = 0; i < num_orders; i++) {
        printf("Delivering pizza order %d\n", orders[i].id);
    }
}

// Round-Robin
void round_robin(PizzaOrder orders[], int num_orders, int
time_quantum) {
    printf("\nRound Robin Scheduling\n");
    int remaining_orders = num_orders;
    int current_order = 0;
    while (remaining_orders > 0) {
        for (int i = 0; i < num_orders; i++) {
            if (orders[i].prep_time > 0) {
                printf("Delivering pizza order %d\n", orders[i].id);
                if (orders[i].prep_time > time_quantum) {
                    orders[i].prep_time -= time_quantum;
                } else {
                    orders[i].prep_time = 0;
                    remaining_orders--;
                }
            }
        }
    }
}
}

```

```
//Priority
void priority(PizzaOrder orders[], int num_orders) {
    printf("\nPriority Scheduling\n");
    for (int i = 0; i < num_orders - 1; i++) {
        for (int j = 0; j < num_orders - i - 1; j++) {
            if (orders[j].priority > orders[j + 1].priority) {
                // Swap orders
                PizzaOrder temp = orders[j];
                orders[j] = orders[j + 1];
                orders[j + 1] = temp;
            }
        }
    }
    for (int i = 0; i < num_orders; i++) {
        printf("Delivering pizza order %d\n", orders[i].id);
    }
}
```

## Output:


```
manishj@ubuntu: ~/Desktop/osexp6
manishj@ubuntu:~/Desktop/osexp6$ gcc pizza.c
manishj@ubuntu:~/Desktop/osexp6$ ./a.out
Enter the number of pizza orders: 5
Enter details for pizza order 1:
Preparation time (in minutes): 10
Delivery distance (in kilometers): 5
Priority level (1 = highest, 5 = lowest): 4
Enter details for pizza order 2:
Preparation time (in minutes): 9
Delivery distance (in kilometers): 4
Priority level (1 = highest, 5 = lowest): 3
Enter details for pizza order 3:
Preparation time (in minutes): 12
Delivery distance (in kilometers): 7
Priority level (1 = highest, 5 = lowest): 1
Enter details for pizza order 4:
Preparation time (in minutes): 6
Delivery distance (in kilometers): 2
Priority level (1 = highest, 5 = lowest): 5
Enter details for pizza order 5:
Preparation time (in minutes): 7
Delivery distance (in kilometers): 3
Priority level (1 = highest, 5 = lowest): 2

Enter the number of delivery drivers: 3

Enter the time quantum for Round Robin scheduling: 5

FCFS Scheduling
Delivering pizza order 1
Delivering pizza order 2
Delivering pizza order 3
Delivering pizza order 4
Delivering pizza order 5

SJF Scheduling
Delivering pizza order 4
Delivering pizza order 5
Delivering pizza order 2
Delivering pizza order 1
Delivering pizza order 3
```

	<div><div> Round Robin Scheduling Delivering pizza order 4 Delivering pizza order 5 Delivering pizza order 2 Delivering pizza order 1 Delivering pizza order 3</div><div>Priority Scheduling Delivering pizza order 3 Delivering pizza order 5 Delivering pizza order 2 Delivering pizza order 1 Delivering pizza order 4 manishj@ubuntu:~/Desktop/osexp6\$ gedit pizza.c</div></div>
Question	<p>Explain which algorithm will be best suited for the following scenario. Is it possible to combine the above algorithms to create an even more efficient algorithm? If yes, then explain how?</p> <p>In a pizza delivery scenario where drivers need to deliver orders, the Priority Queue algorithm is the most suitable. This algorithm prioritizes orders based on their preparation time and delivery distance, ensuring that the most important orders are delivered first.</p> <p>While it's not possible to combine the algorithms directly, you can use a combination of approaches to improve efficiency. For example, you could use a priority queue to manage order delivery priorities and then apply Round Robin scheduling to allocate delivery tasks among drivers fairly. This way, urgent orders are handled promptly, and drivers share the workload evenly.</p>
CONCLUSION:	<p>Hence, by completing this experiment I came to know about Combining approaches, like prioritizing orders and fairly assigning them to drivers, can improve efficiency in a pizza delivery system where the Priority Queue algorithm is most suitable for prioritizing orders based on their preparation time and delivery distance.</p>