|  |  |
| --- | --- |
| **Name** | Manish Shashikant Jadhav |
| **UID no.** | 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:** |
| **Question** | **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?**  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.  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. |
| **CONCLUSION:** | 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. |