DAYANANDA SAGAR UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY
KUDLU GATE
BANGALORE - 560068

MINI PROJECT REPORT

ON

"OS PROJECT OF ROUND ROBIN ALGORITHM"

Operating Systems Laboratory(20CS3506)
5th SEMESTER
BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

Submitted by

KEERTHANA M G -(ENG20AM0033)
SAHANA R-(ENG20AM0049)
NIVEDHA S-(ENG21AM3030)

Under the supervision of
Supervisor(s) name
Prof.Gaurav Kumar

DAYANANDA SAGAR UNIVERSITY

School of Engineering, Kudlu Gate, Bangalore-560068

CERTIFICATE

This is to certify that Ms ***KEERTHANA M G , SAHANA R , NIVEDHA S*** *bearing USN* ***ENG20AM0033 , ENG20AM0049 , ENG21AM3030*** *has satisfactorily completed her Mini Project as prescribed by the University for the* ***fifth*** *semester B.Tech. programme in Computer Science & Engineering during the year* ***third*** *at the School of Engineering, Dayananda Sagar University., Bangalore.*

Date: _____

_____

Signature of the faculty in-charge

Max Marks	Marks Obtained

_____

Signature of Chairman

Department of Computer Science & Engineering

DECLARATION

We hereby declare that the work presented in this mini project entitled- ROUND ROBIN ALGORITHM , has been carried out by us and it has not been submitted for the award of any degree, diploma or the mini project of any other college or university.

KEERTHANA M G -(ENG20AM0033)
SAHANA R-(ENG20AM0049)
NIVEDHA S-(ENG21AM3030)

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

We are especially thankful to our **Chairman** Dr.Udaya Kumar Reddy K.R, for providing necessary departmental facilities, moral support and encouragement.

We are very much thankful to our **guide** Prof.Gaurav Kumar , for providing help and suggestions in completion of this mini project successfully.

We have received a great deal of guidance and co-operation from our friends and we wish to thank all that have directly or indirectly helped us in the successful completion of this project work.

KEERTHANA M G -(ENG20AM0033)
SAHANA R-(ENG20AM0049)
NIVEDHA S-(ENG21AM3030)

TABLE OF CONTENTS

ABSTRACT

Scheduling is the process of allocating processes to the CPU in order to optimize some objective function. There are many algorithms used to schedule processes. The Round Robin (RR) CPU scheduling algorithm is one of these algorithms which is effective in time sharing and real time operating systems. It gives reasonable response time.

In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

CONTENTS

1. INTRODUCTION

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems.This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

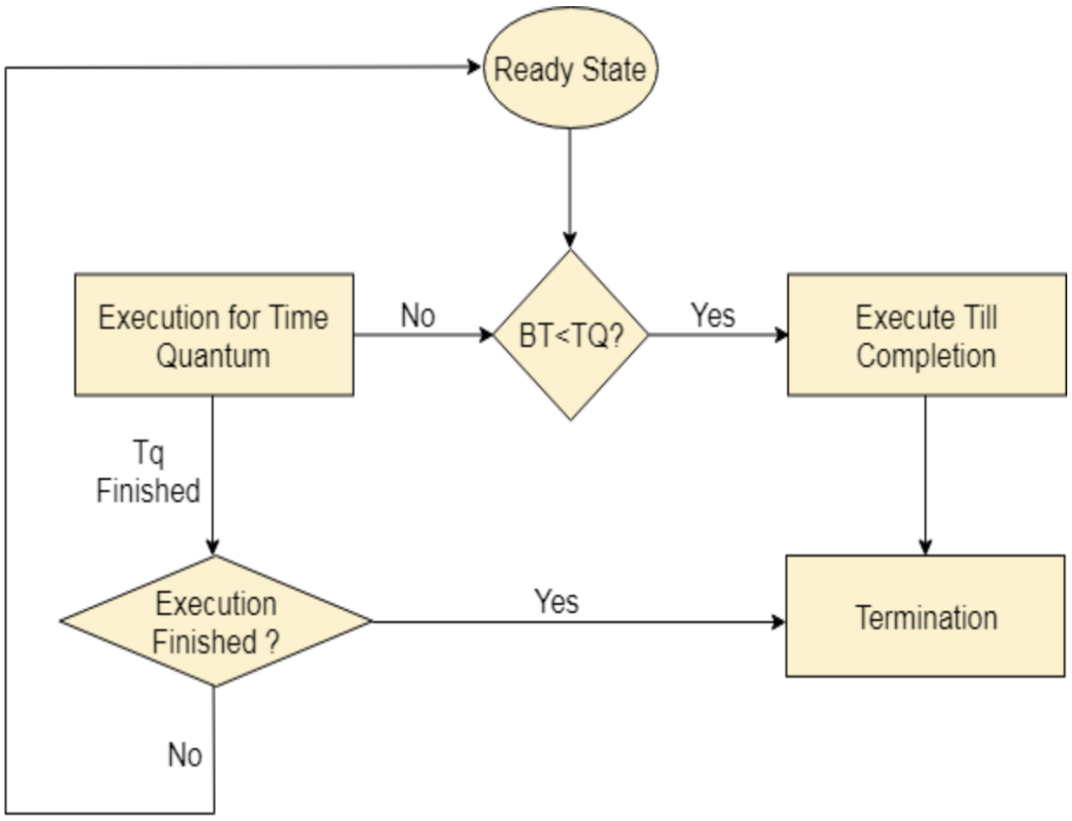
2.PROBLEM STATEMENT

Taylor is a Linux expert who wants to have an online system where she can handle student queries. Since there can be multiple requests at any time she wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of her time. She will log into the system from 10am to 12am only. She wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time she spent on handling queries and average query time.

The given problem is scheduling problem. The problem can be solved by Round Robin algorithm.

3.DESIGN

3.1. METHODOLOGY



3.2.PROGRAM

```
#include<stdio.h>
struct Query {
    char QueryID[3];
    int ArrivalTime;
    int BurstTime;
    int CompletionTime;
    int TotalTime;
}Faculty[120], Student[120], Mix[120];

// Initializing required variables (globally):
int TimeQuantum=0, FacultyCount=0, StudentCount=0, MixCount=0, TotalQueries=0,
Burst=120;
int TQ=0, WaitTime=0, TATime=0, counter=0, total, CTarr[120], maximumCT=0;

// Function to take Required inputs for a query:
// Time complexity = O(TotalQueries), TotalQueries is a limited int.
void InputsForProcess() {
    int QueryType, AT=1000, BT=0;
    ValidQuery:
    printf("\nEnter total number of Queries: ");
    scanf("%d", &TotalQueries);
    // Check whether entered query number is <0 or >120
    if(TotalQueries<=0 || TotalQueries>120) {
        printf("\nQueries cannot be <0 or >120!\n");
        goto ValidQuery;
    }
    else {
        TQ = TotalQueries;  // for RoundRobin() function
        printf("\nEnter Time Quantum for each query: ");
        scanf("%d", &TimeQuantum);
```

```
        // Taking inputs for all the queries
        for(int i=0; i<TotalQueries; i++) { //Time complexity = O(TotalQueries)
            TryQuery:
            printf("\nType of Query (1 for Faculty, 2 for Student): ");
            scanf("%d", &QueryType);

            // Query Processing For Faculty
            if(QueryType == 1) {
                printf("\nEnter Query ID: ");
                scanf("%s", &Faculty[FacultyCount].QueryID[0]);
                FTime:
                printf("Enter Query Arrival Time: ");
                scanf("%d", &AT);
                // Check Time constraint
                if(AT<1000 || AT>1200 || (AT<1100 && AT>1059) || (AT<1200 && AT>1159)) {
                    printf("\nEnter Correct Time!\n");
                    goto FTime;
                }
                else {  // Simplifying ArrivalTime for further calculations
                    if (AT>=1000 && AT<1100) {
                        Faculty[FacultyCount].ArrivalTime = AT-1000;
                    }
                    else {
                        Faculty[FacultyCount].ArrivalTime = AT-1040;
                    }
                }
                FBTime:
                printf("Enter Burst Time: ");
                scanf("%d", &BT);
                if(Burst - BT < 0 || BT <= 0 || Faculty[FacultyCount].ArrivalTime + BT >= 120) {
// initially Burst=120
                    if(BT<=0) {
                    printf("\nBurst Time cannot be less than 0\n"); }
                    else {
```

```
                if (Burst-BT<=0) {
                    int choice;
                    printf("\nSudesh Sharma will not have enough time to handle this Query
because of high BurstTime."
                    "\nWant to change BurstTime? (1 : Yes; Else : No) ");
                    scanf("%d", &choice);
                    if(choice==1) { goto FBTime; }
                    else {
                        printf("\nOK. This query's all data will be lost\n");
                        goto TryQuery;
                    }
                }
                else {
                    printf("\nInvalid Burst time for corresponding Arrival Time\n");
                }
            }
            printf("Please enter valid Burst Time\n");
            goto FBTime;
        }
        else {
            Faculty[FacultyCount].BurstTime = BT;
        }
        Burst -= BT;    // Updates Total Remaining Burst time
        Faculty[FacultyCount].TotalTime = Faculty[FacultyCount].BurstTime;
        FacultyCount++;
    }

    // Query Processing For Student
    else if(QueryType == 2) {
        printf("\nEnter Query ID: ");
        scanf("%s", &Student[StudentCount].QueryID[0]);
        STime:
        printf("Enter Query Arrival Time: ");
        scanf("%d", &AT);
```

```
            // Check Time constraint
            if(AT<1000 || AT>1200 || (AT<1100 && AT>1060) || (AT<1200 && AT>1160)) {
               printf("\nEnter valid Time!\n");
               goto STime;
            }
            else {
               if (AT>=1000 && AT<1100) {
                  Student[StudentCount].ArrivalTime = AT-1000;
               }
               else {
                  Student[StudentCount].ArrivalTime = AT-1040;
               }
            }
            SBTime:
            printf("Enter Burst Time: ");
            scanf("%d", &BT);
            if(Burst - BT < 0 || BT <= 0 || Student[StudentCount].ArrivalTime + BT >= 120) {
// initially Burst=120
               if(BT<=0) {
               printf("\nBurst Time cannot be less than 0\n"); }
               else {
                  if (Burst-BT<=0) {
                     int choice;
                     printf("\nSudesh Sharma won't have enough time to handle this Query
because of high BurstTime."
                        "\nWant to change BurstTime? (1 : Yes; Else : No) ");
                     scanf("%d", &choice);
                     if(choice==1) {
                        goto FBTime;
                     }
                     else {
                        printf("\nOK. This query's all data will be lost\n");
                        goto TryQuery;
                     }
```

```
                }
                else {
                    printf("\nInvalid Burst time for corresponding Arrival Time\n");
                }
            }
            printf("Please enter valid Burst Time\n");
            goto SBTime;
        }
        else {
            Student[StudentCount].BurstTime = BT;   // Updates Total Remaining Burst time
        }
        Burst -= BT;
        Student[StudentCount].TotalTime = Student[StudentCount].BurstTime;
        StudentCount++;
    }
    else {  // In case any other wrong input
        printf("\nInvalid Input. Please try again.\n");
        goto TryQuery;
    }
  }
 }
}
// Sorting Faculties and Students Queries according to Arrival Time using QuickSort
algorithm:
// Time complexity of Faculty QuickSort = O(nlog(n)), n=no. of Faculty queries to sort
(limited)
int Fpartition(int low, int high) {
   int pivot = Faculty[high].ArrivalTime;
   int i = (low - 1);
   for (int j=low; j<=high; j++) {
      if (Faculty[j].ArrivalTime < pivot) {
         i++;
         Faculty[FacultyCount] = Faculty[i];
         Faculty[i] = Faculty[j];
```

```
            Faculty[j] = Faculty[FacultyCount];
        }
    }
    Faculty[FacultyCount] = Faculty[i+1];
    Faculty[i+1] = Faculty[high];
    Faculty[high] = Faculty[FacultyCount];
    return(i+1);
}
void FacultySort(int low, int high) {
    if(low < high) {
        int pi = Fpartition(low, high);
        FacultySort(low, pi-1);
        FacultySort(pi+1, high);
    }
}
// Time complexity of Student QuickSort = O(mlog(m)), m=no. of Student queries to sort (limited)
int Spartition(int low, int high) {
    int pivot = Student[high].ArrivalTime;
    int i = (low - 1);
    for (int j=low; j<=high; j++) {
        if (Student[j].ArrivalTime < pivot) {
            i++;
            Student[StudentCount] = Student[i];
            Student[i] = Student[j];
            Student[j] = Student[StudentCount];
        }
    }
    Student[StudentCount] = Student[i+1];
    Student[i+1] = Student[high];
    Student[high] = Student[StudentCount];
    return(i+1);
}
void StudentSort(int low, int high) {
```

```
    if(low < high) {
        int pi = Spartition(low, high);
        StudentSort(low, pi-1);
        StudentSort(pi+1, high);
    }
}
// function to merge Faculty and Student's queries into one variable of structure (Mix):
// Time complexity = O(FacultyCount + StudentCount)
void MergeQueries() {
    int iSC=0, iFC=0;   // Counting variables to keep count of added queries into Mix variable
    if(FacultyCount !=0  && StudentCount !=0) {      // got entries for both
                while(iSC < StudentCount && iFC < FacultyCount) {
                        if(Faculty[iFC].ArrivalTime == Student[iSC].ArrivalTime) {          //
both entries arrives at same time
                                Mix[MixCount] = Faculty[iFC];        // priority to faculty
                                MixCount++;
                                iFC++;
                                Mix[MixCount] = Student[iSC];        // and then student
                                MixCount++;
                                iSC++;
                        }
                        else if(Faculty[iFC].ArrivalTime < Student[iSC].ArrivalTime) {     //
faculty entry came before
                                Mix[MixCount] = Faculty[iFC];
                                MixCount++;
                                iFC++;
                        }
                        else if(Faculty[iFC].ArrivalTime > Student[iSC].ArrivalTime) {     //
student entry came first
                                Mix[MixCount] = Student[iSC];
                                MixCount++;
                                iSC++;
                        }
                }
```

```
                if(MixCount != (FacultyCount + StudentCount)) {    // in case there's any
unadded query (which most probably will occur)
                        if(FacultyCount != iFC) {   // Adding remained Faculty Queries
                                while(iFC != FacultyCount) {
                                        Mix[MixCount] = Faculty[iFC];
                                        MixCount++;
                                        iFC++;
                                }
                        }
                        else if(StudentCount != iSC) {  // Adding remained Student Queries
                                while(iSC != StudentCount) {
                                        Mix[MixCount] = Student[iSC];
                                        MixCount++;
                                        iSC++;
                                }
                        }
                }
        }
        else if(FacultyCount == 0) {  //got entries for student only
                while(iSC != StudentCount) {
                        Mix[MixCount] = Student[iSC];
                        MixCount++;
                        iSC++;
                }
        }
        else if(StudentCount == 0) {  //got entries for faculty only
                while(iFC != FacultyCount) {
                        Mix[MixCount] = Faculty[iFC];
                        MixCount++;
                        iFC++;
                }
        }
}
// Function to apply RoundRobin operation on Mix variable's queries:
```

```
// Time complexity of Round Robin = O(1)
void RoundRobin() {
   total = Mix[0].ArrivalTime;
   printf("\n==> Time is in minutes for all calculations\n");
   printf("\nQuery
ID\tArrivalTime\tBurstTime\tWaitingTime\tTurnAroundTime\tCompletionTime\n");
   for(int i = 0; TQ != 0;) {
      if(Mix[i].TotalTime <= TimeQuantum && Mix[i].TotalTime > 0) {   // (First if) Process
will complete without any preemption
         total = total + Mix[i].TotalTime;
         Mix[i].TotalTime = 0;
         counter = 1;
      }
      else if(Mix[i].TotalTime > 0) { // Process will preempt according to TimeQuantum
         Mix[i].TotalTime -= TimeQuantum;
         total = total + TimeQuantum;
      }
      if(Mix[i].TotalTime == 0 && counter == 1) {     // continue after first if
         TQ--;
         int ATCalc = Mix[i].ArrivalTime+1000;
         int CTCalc = total+1000;
         CTarr[i] = CTCalc;
         if(ATCalc>1059) {
            ATCalc += 40;
         }
         if(CTCalc>1059) {
            CTCalc += 40;
         }
         printf("\n%s\t\t%d hh:mm\t%d minutes\t%d minutes\t%d minutes\t%d hh:mm",
            Mix[i].QueryID, ATCalc, Mix[i].BurstTime,
            total-Mix[i].ArrivalTime-Mix[i].BurstTime, total-Mix[i].ArrivalTime, CTCalc);
         WaitTime += total - Mix[i].ArrivalTime - Mix[i].BurstTime;
         TATime += total - Mix[i].ArrivalTime;
         counter = 0;
```

```
        }
        if(i == TotalQueries - 1) {
            i = 0;
        }
        else if(Mix[i+1].ArrivalTime <= total) {
            i++;
        }
        else {
            i = 0;
        }
    }
}
// Function to find maximum Completion Time:
// Time complexity = O(1) bcoz MixCount is limited int value
void MaxCT() {
    maximumCT = CTarr[0];
    for(int i=1; i<MixCount; i++) {
        if(maximumCT < CTarr[i]) {
            maximumCT = CTarr[i];
        }
    }
}
// Function to print Final Result of program:
// Time complexity = O(1)
void PrintResult() {
    MaxCT(); total = Mix[0].ArrivalTime;
    printf("\n\nSummary of Execution: \n\n");
    printf("Total Time Spent on handling Queries: %d minutes\n", maximumCT-total-1000);
    float avgWaitTime = WaitTime * 1.0 / TotalQueries;
    float avgTATime = TATime * 1.0 / TotalQueries;
    printf("Average TurnAround Time : %.2f minutes\n", avgTATime);
    printf("Average Waiting Time : %.2f minutes", avgWaitTime);
    printf("\n\nProgram Execution Completed!\n\n");
}
```

```
// Main function:
// Overall Time Complexity = 2*O(n + m) + O(nlog(n)) + O(mlog(m))  + 2*O(1) =
O(nlog(n)) + O(mlog(m))
int main() {
    /* Program execution sequence:
    1. Taking inputs of queries from user
    2. Sorting all queries according to ArrivalTime
    3. Merging all queries (initial priority to Faculty's query)
    4. Applying RoundRobin algorithm on merged queries
    5. Print the results */
    printf("\nWelcome to the OS Project made by Dhiraj Kelhe.\n\n"
        "Please follow these instructions to execute the program:\n"
        "1. Enter number of queries between 0 & 120\n"
        "2. Make sure to keep value of TimeQuantum minimum for convinience\n"
        "3. Enter Query Arrival Time in the format of HHMM\n"
        "    Example: 10:25 should be entered as 1025\n"
        "4. Next Query's ArrivalTime must be less than previous Query's CompletionTime
(ArrivalTime + BurstTime)\n"
        "5. BurstTime must be entered such that (ArrivalTime + BurstTime) < 120\n");
    InputsForProcess(); //Time Complexity = O(TotalQueries)
    FacultySort(0, FacultyCount-1); // Time Complexity = O(nlog(n)); n=FacultyCount
    StudentSort(0, StudentCount-1); // Time Complexity = O(mlog(m)); m=StudentCount
    MergeQueries(); // Time Complexity = O(TotalQueries)
    RoundRobin();   // Time Complexity = O(1)
    PrintResult();  // Time Complexity = O(1)
}
```

4.OUTPUT SCREENSHOTS

```
Please follow these instructions to execute the program:
1. Enter number of queries between 0 & 120
2. Make sure to keep value of TimeQuantum minimum for convinience
3. Enter Query Arrival Time in the format of HHMM
    Example: 10:25 should be entered as 1025
4. Next Query's ArrivalTime must be less than previous Query's CompletionTime (ArrivalTime + BurstTime)
5. BurstTime must be entered such that (ArrivalTime + BurstTime) < 120

Enter total number of Queries: 3

Enter Time Quantum for each query: 3

Type of Query (1 for Faculty, 2 for Student): 1

Enter Query ID: f1
Enter Query Arrival Time: 1030
Enter Burst Time: 15

Type of Query (1 for Faculty, 2 for Student): 2

Enter Query ID: s1
Enter Query Arrival Time: 1035
Enter Burst Time: 25

Type of Query (1 for Faculty, 2 for Student): 2

Enter Query ID: s2
Enter Query Arrival Time: 1040
Enter Burst Time: 5

==> Time is in minutes for all calculations

Query ID        ArrivalTime     BurstTime       WaitingTime     TurnAroundTime  CompletionTime

s2              1040 hh:mm      5 minutes       11 minutes      16 minutes      1056 hh:mm
f1              1030 hh:mm      15 minutes      14 minutes      29 minutes      1059 hh:mm
s1              1035 hh:mm      25 minutes      15 minutes      40 minutes      1115 hh:mm

Summary of Execution:

Total Time Spent on handling Queries: 45 minutes
Average TurnAround Time : 28.33 minutes
Average Waiting Time : 13.33 minutes

Program Execution Completed!
```

5.CONCLUSION

- The name of this algorithm comes from the round-robin principle, where every person gets an equal share of something turn by turn.
- Every process gets executed in a cyclic way and the processing is done in FIFO order.
- The execution of the Round Robin scheduling algorithm mainly depends on the value of the time quantum.
- As the time quantum value decreases
    - *The response time decreases*.
    - *And, The number of context switches increases.*
- As the time quantum value increases
    - *The response time increases*.
    - *And, The number of context switches decreases.*
- We can conclude that a good scheduling algorithm for a real-time and time-sharingsystem must possess the following characteristics:
    - Minimum context switches.
    - Maximum CPU utilization.

6.REFERENCES

[1] Abdulrahim A., Abdullahi S. E. and Sahalu J. B. (2014) A new improved round robin (NIRR) CPU scheduling algorithm International Journal of Computer Applications 90(4).

[2] Singh P., Singh V. and Pandey A. (2014) Analysis and comparison of CPU scheduling algorithms International Journal of Emerging Technology and Advanced Engineering 4(1) 91-95.

[3] Joshi R. and Tyagi S. B. (2015) Smart optimized round robin (SORR) CPU scheduling algorithm International Journal of Advanced Research in Computer Science and Software Engineering 5(7) 568-574

[4] Rajput I. S., and Gupta D. (2012) A priority based round robin CPU scheduling algorithm for real-time systems International Journal of Innovations in Engineering and Technology 1(3) 1-11.

[5] Singh A., Goyal P. and Batra S. (2010) An optimized round robin scheduling algorithm for CPU scheduling International Journal on Computer Science and Engineering 2(07) 2383-2385.