OS ASSIGNMENT:

MINI PROJECT : CPU Scheduling Algorithm

For this mini project on scheduling algorithms, you can consider implementing the scheduling algorithm in cycles. This algorithm can be commonly used in operating systems to allocate CPU time to multiple processes. It works by its turn based on the priority and burst time to each process in a circular manner, allowing each process to execute for a time before it can be executed completely, it is not a pre-emptive version of this program.

Here is the complete guide for my algorithm execution:

Here are some steps to follow while entering data :

1. Enter the process name (eg. P1, P2, P3 ...)
2. Enter the priority of the process (eg. 1, 2, 3 ...)
3. Enter the burst time of the process (eg. 5, 6, 7 ...) 4 . Enter the arrival time of the process (eg. 1, 2, 3 ...)

please constrain your ranges of entering data to the following ranges

:

1. Process name : (A-Z)
2. Priority: (050) [High Number => High Priority]
3. Burst Time : (0 - 50)
4. Arrival Time: (010)

Here is my algorithm on which it schedules the process: 1.Processs will be entered in the sorted order starting from 0 2. process with arrival time 0 will run until any process satisfying the conditions for cycles

THE ALGORITHM :

CYCLE 1: ALL process with priority higher than 30 will be executed as per they entered in the queue

CYCLE 2: ALL process with priority greater than 15 and burst time les s than 10ms will be executed as per they entered in the queue

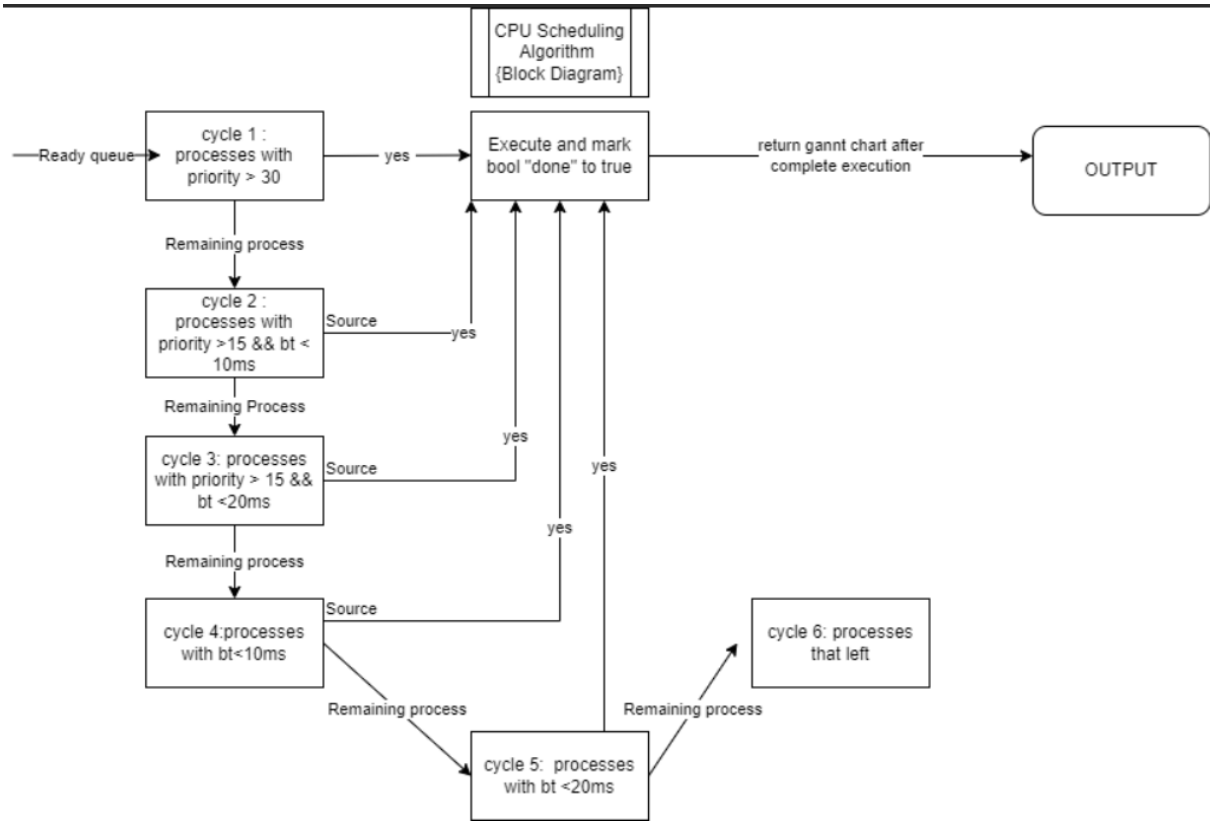
CYCLE 3: ALL process with priority greater than 15 and burst time les s than 20ms will be executed as per they entered in the queue

CYCLE 4: ALL process with burst time less than 10ms will will be exec uted as per they entered in the queue

CYCLE 5: ALL process with burst time less than 20ms will be executed as per they entered in the queue

CYCLE 6: execute all remaining process.

Block Diagram:



code for the above algorithm:

```
#include <bits/stdc++.h>
using namespace std;

struct QNode {
  string process;
  int priority;
  int burstTime;
  int arrivalTime;
  bool done;
  QNode *next;
  QNode(string process, int p, int bt, int arrt, bool done = false) {
    this->process = process;
    this->priority = p;
    this->burstTime = bt;
    this->arrivalTime = arrt;
    next = NULL;
  }
};

struct Queue {
  QNode *front, *rear;
  Queue() { front = rear = NULL; }

  void enQueue(string process, int p, int bt, int arrt) {

    // Create a new LL node
    QNode *temp = new QNode(process, p, bt, arrt);

    // If queue is empty, then
    // new node is front and rear both
```

```
    if (rear == NULL) {
      front = rear = temp;
      return;
    }

    // Add the new node at
    // the end of queue and change rear
    rear->next = temp;
    rear = temp;
  }

  // Function to remove
  // a key from given queue q
  void deQueue() {
    // If queue is empty, return NULL.
    if (front == NULL)
      return;

    // Store previous front and
    // move front one node ahead
    QNode *temp = front;
    front = front->next;

    // If front becomes NULL, then
    // change rear also as NULL
    if (front == NULL)
      rear = NULL;

    delete (temp);
  }
};
void guide() {
  cout << "Here are some steps to follow while entering data :" << endl;
  cout << "1. Enter the process name (eg. P1, P2, P3 ...)" << endl;
  cout << "2. Enter the priority of the process (eg. 1, 2, 3 ...)" << endl;
  cout << "3. Enter the burst time of the process (eg. 5, 6, 7 ...)" << endl;
  cout << "4. Enter the arrival time of the process (eg. 1, 2, 3 ...)" << endl;
  cout << "please constrain your ranges of entering data to the following "
          "ranges: "
       << endl;
  cout << "1. Process name : (A-Z)" << endl;
  cout << "2. Priority : (0 - 50) [High Number => High Priority]" << endl;
  cout << "3. Burst Time : (0 - 50)" << endl;
  cout << "4. Arrival Time : (0 - 10)" << endl;
  cout << "Here is my algorithm on which it schedules the process: " << endl;
  cout << "1.Processs will be entered in the sorted order starting from 0"
       << endl;
  cout << "2.process with arrival time 0 will run untill any process stisfying "
          "the conditions for cycles "
       << endl;
  cout << "THE ALGORITHM" << endl;
  cout << "CYCLE 1: ALL process with priority higher than 30 will be executed "
          "as per they entered in the queue"
       << endl;
  cout << "CYCLE 2: ALL process with priority greater than 15 and burst time "
          "less than 10ms  will be executed as per they entered in the queue"
       << endl;
  cout << "CYCLE 3: ALL process with priority greater than 15 and burst time "
          "less than 20ms will be executed as per they entered in the queue"
       << endl;
  cout << "CYCLE 4: ALL process with burst time less than 10ms will will be "
          "executed as per they entered in the queue"
       << endl;
  cout << "CYCLE 5: ALL process with burst time less than 20ms will be "
          "executed as per they entered in the queue"
       << endl;
  cout << "CYCLE 6: execute all remaining process" << endl << endl << endl;
}
```

```
// Driver code
int main() {
  Queue readyQueue;
  string p;
  int p1, bt, at, n;
  guide();
  cout << "Enter the number of processes: ";
  cin >> n;
  cout << endl;
  cout << "Note : Please Enter the process in increasing Arrival time and "
          "start from zero";
  cout << endl;
  for (int i = 0; i < n; i++) {
    cout << i + 1 << ". "
         << "ENTER PROCESS NAME  :  ";
    cin >> p;
    cout << endl
         << i + 1 << ". "
         << "ENTER PRIORITY  :  ";
    cin >> p1;
    cout << endl
         << i + 1 << ". "
         << "ENTER BURST TIME  :  ";
    cin >> bt;
    cout << endl
         << i + 1 << ". "
         << "ENTER ARRIVAL TIME  :  ";
    cin >> at;
    cout << endl;
    readyQueue.enQueue(p, p1, bt, at);
  }
  cout << endl << "READY QUEUE : " << endl;
  cout << "PROCESS NAME  "
       << "  "
       << "PRIORITY  "
       << "  "
       << "BURST TIME  "
       << "  "
       << "ARRIVAL TIME  " << endl;
  QNode *initial =readyQueue.front;
  while (readyQueue.front != NULL) {
    cout << endl
         << readyQueue.front->process << "                    "
         << readyQueue.front->priority << "        "
         << readyQueue.front->burstTime << "ms          "
         << readyQueue.front->arrivalTime << "ms" << endl;
    readyQueue.front = readyQueue.front->next;
  }
  int t = 0;
  readyQueue.front = initial;
  // the process with arrival time 0;
  int count = 0;
  cout << "the sequence will be as follows:" << endl;
  cout << "PROCESS NAME     TIME" << endl;
  string a = readyQueue.front->process;
  readyQueue.front = initial;
  int tpre;
  // CYCLE 1
  while (readyQueue.front != NULL) {
    if (readyQueue.front->priority > 30&&readyQueue.front->done == false) {
      if (readyQueue.front->arrivalTime > t && initial->done == false) {
        tpre = t;
        t = readyQueue.front->arrivalTime;
        initial->burstTime = initial->burstTime - t+tpre;
        cout << initial->process << "              " << t << endl;
        initial->burstTime = initial->burstTime - readyQueue.front->arrivalTime;
      }
```

```
        t = t + readyQueue.front->burstTime;
        cout << readyQueue.front->process << "               " << t << endl;
        readyQueue.front->done = true;
        readyQueue.front->burstTime = 0;

    }
    readyQueue.front = readyQueue.front->next;
  }
  readyQueue.front = initial;
  // CYCLE 2
  while (readyQueue.front != NULL) {
    if (readyQueue.front->priority > 15 && readyQueue.front->burstTime < 10&&readyQueue.front->done==false) {
      if (readyQueue.front->arrivalTime > t && initial->done == false) {
        tpre = t;
        t = readyQueue.front->arrivalTime;
        initial->burstTime = initial->burstTime - t+tpre;
        cout << initial->process << "               " << t << endl;
        readyQueue.front->done = true;

      }
        t = t + readyQueue.front->burstTime;
        cout << readyQueue.front->process << "               " << t << endl;
        readyQueue.front->done = true;
        readyQueue.front->burstTime = 0;

    }
    readyQueue.front = readyQueue.front->next;
  }
  readyQueue.front = initial;
  // CYCLE 3
  while (readyQueue.front != NULL) {
    if (readyQueue.front->priority > 15 && readyQueue.front->burstTime < 20&&readyQueue.front->done==false) {
      if (readyQueue.front->arrivalTime > t && initial->done == false) {
        tpre = t;
        t = readyQueue.front->arrivalTime;
        initial->burstTime = initial->burstTime - t+tpre;
        cout << initial->process << "               " << t << endl;
        readyQueue.front->done = true;

      }
        t = t + readyQueue.front->burstTime;
        cout << readyQueue.front->process << "               " << t << endl;
        readyQueue.front->done = true;
        readyQueue.front->burstTime = 0;

    }
    readyQueue.front = readyQueue.front->next;
  }
  readyQueue.front = initial;
  // CYCLE 4
  while (readyQueue.front != NULL ) {
    if (readyQueue.front->burstTime < 10&&readyQueue.front->done==false) {
      if (readyQueue.front->arrivalTime > t && initial->done == false) {
        tpre = t;
        t = readyQueue.front->arrivalTime;
        initial->burstTime = initial->burstTime - t+tpre;
        cout << initial->process << "                " << t << endl;
        readyQueue.front->done = true;

      }
        t = t + readyQueue.front->burstTime;
        cout << readyQueue.front->process << "               " << t << endl;
        readyQueue.front->done = true;
        readyQueue.front->burstTime = 0;
      }

    readyQueue.front = readyQueue.front->next;
  }
```

```
  readyQueue.front = initial;
  //cycle 5
  while (readyQueue.front != NULL ) {
    if (readyQueue.front->burstTime < 20&&readyQueue.front->done==false) {
      if (readyQueue.front->arrivalTime > t && initial->done == false) {
        tpre = t;
        t = readyQueue.front->arrivalTime;
        initial->burstTime = initial->burstTime - t+tpre;
        cout << initial->process << "               " << t << endl;
        readyQueue.front->done = true;

      }
        t = t + readyQueue.front->burstTime;
        cout << readyQueue.front->process << "               " << t << endl;
        readyQueue.front->done = true;
        readyQueue.front->burstTime = 0;

    }
    readyQueue.front = readyQueue.front->next;
  }
  readyQueue.front = initial;
  //cycle 6
  while (readyQueue.front != NULL ) {
    if (readyQueue.front->done==false) {

        t = t + readyQueue.front->burstTime;
        cout << readyQueue.front->process << "               " << t << endl;
        readyQueue.front->done = true;
        readyQueue.front->burstTime = 0;

    }
    readyQueue.front = readyQueue.front->next;
  }
  return 0;
}
```

Example Input:

```
Here are some steps to follow while entering data :
1. Enter the process name (eg. P1, P2, P3 ...)
2. Enter the priority of the process (eg. 1, 2, 3 ...)
3. Enter the burst time of the process (eg. 5, 6, 7 ...)
4. Enter the arrival time of the process (eg. 1, 2, 3 ...)
please constrain your ranges of entering data to the following ran
ges:
1. Process name : (A-Z)
2. Priority : (0 - 50) [High Number => High Priority]
3. Burst Time : (0 - 50)
4. Arrival Time : (0 - 10)
Here is my algorithm on which it schedules the process:
1.Processs will be entered in the sorted order starting from 0
2.process with arrival time 0 will run untill any process stisfyin
g the conditions for cycles
THE ALGORITHM
CYCLE 1: ALL process with priority higher than 30 will be executed
 as per they entered in the queue
CYCLE 2: ALL process with priority greater than 15 and burst time
less than 10ms  will be executed as per they entered in the queue
CYCLE 3: ALL process with priority greater than 15 and burst time
less than 20ms will be executed as per they entered in the queue
CYCLE 4: ALL process with burst time less than 10ms will will be e
xecuted as per they entered in the queue
CYCLE 5: ALL process with burst time less than 20ms will be execut
ed as per they entered in the queue
CYCLE 6: execute all remaining process


Enter the number of processes: 6

Note : Please Enter the process in increasing Arrival time and sta
rt from zero
1. ENTER PROCESS NAME  :  P1

1. ENTER PRIORITY  :  0
```

```
1. ENTER BURST TIME  :  12
1. ENTER ARRIVAL TIME  :  0
2. ENTER PROCESS NAME  :  P2
2. ENTER PRIORITY  :  25
2. ENTER BURST TIME  :  18
2. ENTER ARRIVAL TIME  :  3
3. ENTER PROCESS NAME  :  P3
3. ENTER PRIORITY  :  36
3. ENTER BURST TIME  :  21
3. ENTER ARRIVAL TIME  :  4
4. ENTER PROCESS NAME  :  P4
4. ENTER PRIORITY  :  21
4. ENTER BURST TIME  :  8
4. ENTER ARRIVAL TIME  :  6
5. ENTER PROCESS NAME  :  P5
5. ENTER PRIORITY  :  12
5. ENTER BURST TIME  :  21
```

```
5. ENTER ARRIVAL TIME  :  8

6. ENTER PROCESS NAME  :  P6

6. ENTER PRIORITY  :  34

6. ENTER BURST TIME  :  32

6. ENTER ARRIVAL TIME  :  9


READY QUEUE :
PROCESS NAME    PRIORITY    BURST TIME    ARRIVAL TIME

P1                    0        12ms          0ms

P2                    25        18ms          3ms

P3                    36        21ms          4ms

P4                    21        8ms          6ms

P5                    12        21ms          8ms

P6                    34        32ms          9ms
```

Output For Above Input :

```
the sequence will be as follows:
PROCESS NAME     TIME
P1              4
P3               25
P6               57
P4               65
P2               83
P1               87
P5               108
```

SUMMARY:

This document is about a mini project on CPU scheduling algorithms. The algorithm works by allocating CPU time to multiple processes based on priority and burst time. The document provides a guide on entering data and explains the algorithm's execution in cycles. It also includes a block diagram and code for the algorithm implementation.