|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| Operation System Lab  SCHEDULING |
| |  |  |  | | --- | --- | --- | | MFY | 1/6/22 | OS lab | |

# First Come First Serve:

## Code:

#include<stdio.h>

void findWaitingTime(int processes[], int n, bt[], int wt[]){

    wt[0] = 0;

    for (int  i = 1; i < n ; i++ )

        wt[i] =  bt[i-1] + wt[i-1] ;

}

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[]){

    for (int  i = 0; i < n ; i++)

        tat[i] = bt[i] + wt[i];

}

void findavgTime( int processes[], int n, int bt[]){

    int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

    findWaitingTime(processes, n, bt, wt);

    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Processes   Burst time   Waiting time   Turn around time\n");

    for (int  i=0; i<n; i++){

        total\_wt = total\_wt + wt[i];

        total\_tat = total\_tat + tat[i];

        printf("   %d\t\t%d\t\t%d\t\t%d\n",(i+1), bt[i] ,wt[i] ,tat[i] );

    }

    int s=(float)total\_wt / (float)n;

    int t=(float)total\_tat / (float)n;

    printf("\n\nAverage waiting time. \t (%d / %d) = %d",total\_wt, n, s);

    printf("\nAverage turn around time. (%d / %d) = %d",total\_tat, n,t);

}

int main(){

    int processes[] = { 1, 2, 3,8,5,8};

    int  burst\_time[] = {10, 5, 8,8,5,8};

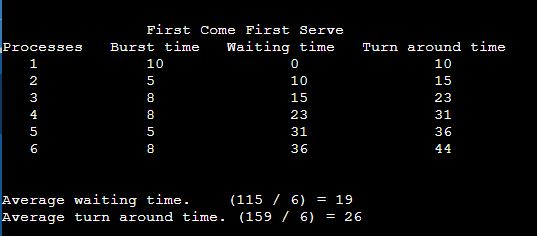
    int n = sizeof processes / sizeof processes[0];

    findavgTime(processes, n,  burst\_time);

    return 0;

}

## Output:



# Shortest Job First:

## Code:

#include <stdio.h>

void main()

{

  printf("\n\n\t\tShortest Job First\n");

  int bt[] = {10, 5, 8, 7, 2, 1},

      p[] = {1, 2, 3, 4, 5, 6},

      n = 6,

      wt[n], tat[n], total = 0,

      // loop varaibles

      i, j, pos, temp;

  float avg\_wt, avg\_tat;

  n = 6;

  //sorting burst time in ascending order using selection sort

  for (i = 0; i < n; i++)

  {

    pos = i;

    for (j = i + 1; j < n; j++)

    {

      if (bt[j] < bt[pos])

        pos = j;

    }

    temp = bt[i];

    bt[i] = bt[pos];

    bt[pos] = temp;

    temp = p[i];

    p[i] = p[pos];

    p[pos] = temp;

  }

  wt[0] = 0;

  //calculate waiting time

  for (i = 1; i < n; i++)

  {

    wt[i] = 0;

    for (j = 0; j < i; j++)

      wt[i] += bt[j];

    total += wt[i];

  }

  avg\_wt = (float)total / n;

  int total\_wt = total;

  total = 0;

  printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");

  for (i = 0; i < n; i++)

  {

    tat[i] = bt[i] + wt[i];

    total += tat[i];

    printf("\np%d\t\t  %d\t\t    %d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);

  }

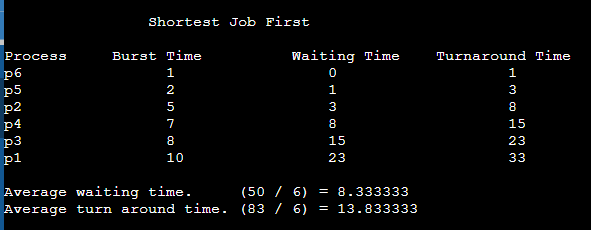
  avg\_tat = (float)total / n;

  printf("\n\nAverage waiting time.  \t  (%d / %d) = %f",total\_wt, n, avg\_wt);

  printf("\nAverage turn around time. (%d / %d) = %f",total, n,avg\_tat);

}

## Output:



# Shortest Job Remaining:

## Code:

#include <stdio.h>

struct process

{

  int WT, AT, BT, TAT;

};

struct process a[10] = {};

int main(){

  printf("\n\n\t\tShortest Job Remaining\n");

  int n = 6, temp[10];

  int count = 0, t = 0, short\_P;

  float total\_WT = 0, total\_TAT = 0, Avg\_WT, Avg\_TAT;

  int bt[] = {10, 5, 8, 7, 2, 1},

      p[] = {0, 0, 3, 4, 0, 6};

  printf("\n");

  for (int i = 0; i < n; i++){

    a[i].BT = bt[i];

    a[i].AT = p[i];

    temp[i] = a[i].BT;

  }

  a[9].BT = 10000; // temp for max

  for (t = 0; count != n; t++){

    short\_P = 9;

    for (int i = 0; i < n; i++){

      if (a[i].BT < a[short\_P].BT && (a[i].AT <= t && a[i].BT > 0))

        short\_P = i;

    }

    a[short\_P].BT = a[short\_P].BT - 1;

    // if any process is completed

    if (a[short\_P].BT == 0){

      // one process complete

      count++;

      a[short\_P].WT = t + 1 - a[short\_P].AT - temp[short\_P];

      a[short\_P].TAT = t + 1 - a[short\_P].AT;

      // total calculation

      total\_WT = total\_WT + a[short\_P].WT;

      total\_TAT = total\_TAT + a[short\_P].TAT;

    }

  }

  Avg\_WT = total\_WT / n;

  Avg\_TAT = total\_TAT / n;

  // printing of the answer

  printf("Id.\tAT.\tBT.\tWT.\tTAT.\n");

  for (int i = 0; i < n; i++){

    printf(" %d\t%d\t%d\t%d\t%d\n", (i + 1), a[i].AT,

           (a[i].TAT - a[i].WT), a[i].WT, a[i].TAT);

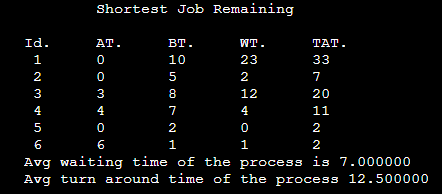
  }

  printf("Avg waiting time of the process is %f\n", Avg\_WT);

  printf("Avg turn around time of the process %f\n", Avg\_TAT);

}

## Output:



# ROUND Robin [pre-emptive]:

## Code:

#include<stdio.h>

#include<conio.h>

#include <time.h>

#include <stdlib.h>

int getNumber(int max){

   return (rand() % (max - 1 + 1)) + 1;

}

int main(){

    srand(time(0));

    int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0;

    float avg\_wt, avg\_tat;

    printf("Total number of process in the system: ");

    y = NOP = getNumber(8);

    printf("%d \n",y);

    int at[y], bt[y], temp[y];

    for(i=0; i<NOP; i++){

        at[i] = getNumber(3);

        bt[i] = getNumber(6);

        temp[i] = bt[i];

    }

    printf("Enter the Time Quantum for the process: \t");

    scanf("%d", &quant);

    printf("\n| Process No | Arrial Time | Burst Time |\t TAT \t|  Waiting Time |");

    for(sum=0, i = 0; y!=0; ){

        if(temp[i] <= quant && temp[i] > 0){

        sum = sum + temp[i];

        temp[i] = 0;

        count=1;

        }

        else if(temp[i] > 0){

            temp[i] = temp[i] - quant;

            sum = sum + quant;

        }

        if(temp[i]==0 && count==1){

            y--;

            printf("\n|      %d     |      %d      |      %d     |\t%d\t|\t%d\t|", (i+1),at[i], bt[i], sum-at[i], (bt[i]-sum-at[i]));

            wt = wt+sum-at[i]-bt[i];

            tat = tat+sum-at[i];

            count =0;

        }

        if(i==NOP-1){

            i=0;

        }

        else if(at[i+1]<=sum){

            i++;

        }

        else{

            i=0;

        }

    }

    avg\_wt = wt \* 1.0/NOP;

    avg\_tat = tat \* 1.0/NOP;

    printf("\n\nAverage Turn Around Time: \t%f", avg\_wt);

    printf("\nAverage Waiting Time: \t%f", avg\_tat);

    getch();

}

## Output:

# ROUND Robin [pre-emptive]:

## Code: