**QUESTION 1: Write a program to create a child process using system call fork().**

**CODE:**

#include<unistd.h>

#include<stdio.h>

int main()

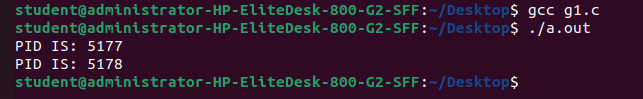
{

   int pid = fork();

   printf("PID IS: %d \n" , getpid());

}

**Output:**

****

**QUESTION 2: Write a program to printf process id’s of parent and child process i.e parent should print its own and child process id while child process should print its own and its prent process id.(use getpid and getppid).**

**CODE:**

#include<unistd.h>

#include<stdio.h>

**int** main()

{

**int** pid = fork();

  if (pid==0)

  {

    sleep(3000);

   printf("this is child process %d \n",getppid());

  }

  else if(pid>0)

  {

    printf("this process for is parent");

    printf(" %d  \n" , getpid());

  }

  else

  {

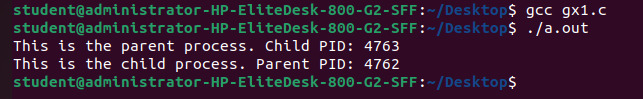
   printf("procss not created");

  }

  return 0;

}

**Output :**

****

**QUESTION 3 : Write a program to create a child process which will list all the files present in your system.make sure that parent process waits until child has not completeits execution.(use wait() , exit()).**

**CODE:**

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

int main()

{

  int pid = fork();

  if (pid==0)

  {

    execlp("ls" , "ls"  , "-l" ,(char \*) NULL);

   printf("this is child process %d \n",getpid());

   exit(0);

  }

  else if(pid>0)

  {

     sleep(5);

     wait(NULL);

    printf("this process for is parent");

    printf(" %d  \n" , getpid());

     int pid = fork();

     if (pid==0)

     {

*// execlp("ls" , "ls"  , "-l" ,(char \*) NULL);*

       printf("this is child process %d \n",getpid());

       exit(0);

     }

     else if(pid>0)

  {

     wait(NULL);

    printf("this process for is parent");

    printf(" %d  \n" , getpid());

  }

  else

  {

   printf("procss not created");

  }

 }

  else

  {

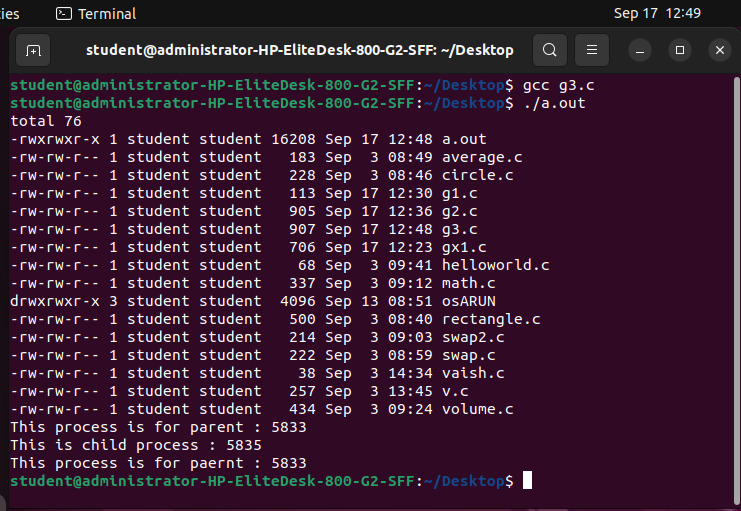
   printf("procss not created");

  }

  return 0;

}

**Output:**

****

**QUESTION 3 : Write a code for FCFS.**

**CODE:**

#include<stdio.h>

#include <stdlib.h>

struct process\_struct

{

int pid;

int at;

int bt;

int ct,wt,tat,rt,start\_time;

}ps[100];

int findmax(int a, int b)

{

return a>b?a:b;

}

int comparatorAT(const void \* a, const void \*b)

{

int x =((struct process\_struct \*)a) -> at;

int y =((struct process\_struct \*)b) -> at;

if(x<y)

return -1;

else if( x>=y)

return 1;

}

int comparatorPID(const void \* a, const void \*b)

{

int x =((struct process\_struct \*)a) -> pid;

int y =((struct process\_struct \*)b) -> pid;

if(x<y)

return -1; // No sorting

else if( x>=y)

return 1;

}

int main()

{

int n;

printf("Enter total number of processes: ");

scanf("%d",&n);

float sum\_tat=0,sum\_wt=0,sum\_rt=0;

int length\_cycle,total\_idle\_time=0;

float cpu\_utilization;

for(int i=0;i<n;i++)

{

printf("\nEnter Process %d Arrival Time: ",i);

scanf("%d",&ps[i].at);

ps[i].pid = i ;

}

for(int i=0;i<n;i++)

{

printf("\nEnter Process %d Burst Time: ",i);

scanf("%d",&ps[i].bt);

}

qsort((void \*)ps,n, sizeof(struct process\_struct),comparatorAT);

for(int i=0;i<n;i++)

{

ps[i].start\_time = (i==0) ? ps[i].at : findmax(ps[i].at, ps[i-1].ct);

ps[i].ct = ps[i].start\_time + ps[i].bt;

ps[i].tat = ps[i].ct-ps[i].at;

ps[i].wt = ps[i].tat-ps[i].bt;

ps[i].rt=ps[i].wt;

sum\_tat += ps[i].tat;

sum\_wt += ps[i].wt;

sum\_rt += ps[i].rt;

total\_idle\_time += (i==0) ? 0 : (ps[i].start\_time - ps[i-1].ct);

}

length\_cycle = ps[n-1].ct - ps[0].start\_time;

qsort((void \*)ps,n, sizeof(struct process\_struct),comparatorPID);

printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n");

for(int i=0;i<n;i++)

printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n",ps[i].pid,ps[i].at,ps[i].bt,ps[i].ct,ps[i].tat,ps[i].wt

,ps[i].rt);

printf("\n");

cpu\_utilization = (float)(length\_cycle - total\_idle\_time)/ length\_cycle;

printf("\nAverage Turn Around time= %f ",sum\_tat/n);

printf("\nAverage Waiting Time= %f ",sum\_wt/n);

printf("\nAverage Response Time= %f ",sum\_rt/n);

printf("\nThroughput= %f",n/(float)length\_cycle);

printf("\nCPU Utilization(Percentage)= %f",cpu\_utilization\*100);

printf("\n");

return 0;

}

**OUTPUT:**

**A screenshot of a computer

Description automatically generated**

**QUESTION 4 : Write a code for SRTF.**

**CODE:**

#include<stdio.h>

#include<stdbool.h>

#include<limits.h>

struct process\_struct

{

int pid;

int at;

int bt;

int ct,wt,tat,rt,start\_time;

}ps[100];

int findmax(int a, int b)

{

return a>b?a:b;

}

int findmin(int a, int b)

{

return a<b?a:b;

}

int main()

{

int n;

float bt\_remaining[100];

bool is\_completed[100]={false},is\_first\_process=true;

int current\_time = 0;

int completed = 0;;

float sum\_tat=0,sum\_wt=0,sum\_rt=0,total\_idle\_time=0,length\_cycle,prev=0;

float cpu\_utilization;

int max\_completion\_time,min\_arrival\_time;

printf("Enter total number of processes: ");

scanf("%d",&n);

for(int i=0;i<n;i++)

{

printf("\nEnter Process %d Arrival Time: ",i);

scanf("%d",&ps[i].at);

ps[i].pid = i ;

}

for(int i=0;i<n;i++)

{

printf("\nEnter Process %d Burst Time: ",i);

scanf("%d",&ps[i].bt);

bt\_remaining[i]= ps[i].bt;

}

while(completed!=n)

{

int min\_index = -1;

int minimum = INT\_MAX;

for(int i = 0; i < n; i++) {

if(ps[i].at <= current\_time && is\_completed[i] == false) {

if(bt\_remaining[i] < minimum) {

minimum = bt\_remaining[i];

min\_index = i;

}

if(bt\_remaining[i]== minimum) {

if(ps[i].at < ps[min\_index].at) {

minimum= bt\_remaining[i];

min\_index = i;

}

}

}

}

if(min\_index==-1)

{

current\_time++;

}

else

{

if(bt\_remaining[min\_index] == ps[min\_index].bt)

{

ps[min\_index].start\_time = current\_time;

total\_idle\_time += (is\_first\_process==true) ? 0 : (ps[min\_index].start\_time -

prev);

is\_first\_process=false;

}

bt\_remaining[min\_index] -= 1;

current\_time++;

prev=current\_time;

if(bt\_remaining[min\_index] == 0)

{

ps[min\_index].ct = current\_time;

ps[min\_index].tat = ps[min\_index].ct - ps[min\_index].at;

ps[min\_index].wt= ps[min\_index].tat - ps[min\_index].bt;

ps[min\_index].rt = ps[min\_index].start\_time - ps[min\_index].at;

completed++;

is\_completed[min\_index]=true;

}

}

}

max\_completion\_time = INT\_MIN;

min\_arrival\_time = INT\_MAX;

for(int i=0;i<n;i++)

{

max\_completion\_time = findmax(max\_completion\_time,ps[i].ct);

min\_arrival\_time = findmin(min\_arrival\_time,ps[i].at);

}

length\_cycle = max\_completion\_time - min\_arrival\_time;

printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n");

for(int i=0;i<n;i++)

printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n",ps[i].pid,ps[i].at,ps[i].bt,ps[i].ct,ps[i].tat,ps[i].wt

,ps[i].rt);

printf("\n");

cpu\_utilization = (float)(length\_cycle - total\_idle\_time)/ length\_cycle;

printf("\nAverage Turn Around time= %f ",(float)sum\_tat/n);

printf("\nAverage Waiting Time= %f ",(float)sum\_wt/n);

printf("\nAverage Response Time= %f ",(float)sum\_rt/n);

printf("\nThroughput= %f",n/(float)length\_cycle);

printf("\nCPU Utilization(Percentage)= %f",cpu\_utilization\*100);

return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**QUESTION 5 : Write a code for SJFS.**

**CODE:**

#include<stdio.h>

#include<stdbool.h>

#include<limits.h>

struct process\_struct

{

int pid;

int at;

int bt;

int ct,wt,tat,rt,start\_time;

}ps[100];

int findmax(int a, int b)

{

return a>b?a:b;

}

int findmin(int a, int b)

{

return a<b?a:b;

}

int main()

{

int n;

bool is\_completed[100]={false},is\_first\_process=true;

int current\_time = 0;

int completed = 0;;

printf("Enter total number of processes: ");

scanf("%d",&n);

int sum\_tat=0,sum\_wt=0,sum\_rt=0,total\_idle\_time=0,prev=0,length\_cycle;

float cpu\_utilization;

int max\_completion\_time,min\_arrival\_time;

for(int i=0;i<n;i++)

{

printf("\nEnter Process %d Arrival Time: ",i);

scanf("%d",&ps[i].at);

ps[i].pid = i ;

}

for(int i=0;i<n;i++)

{

printf("\nEnter Process %d Burst Time: ",i);

scanf("%d",&ps[i].bt);

}

while(completed!=n)

{

int min\_index = -1;

int minimum = INT\_MAX;

for(int i = 0; i < n; i++) {

if(ps[i].at <= current\_time && is\_completed[i] == false) {

if(ps[i].bt < minimum) {

minimum = ps[i].bt;

min\_index = i;

}

if(ps[i].bt== minimum) {

if(ps[i].at < ps[min\_index].at) {

minimum= ps[i].bt;

min\_index = i;

}

}

}

}

if(min\_index==-1)

{

current\_time++;

}

else

{

ps[min\_index].start\_time = current\_time;

ps[min\_index].ct = ps[min\_index].start\_time + ps[min\_index].bt;

ps[min\_index].tat = ps[min\_index].ct - ps[min\_index].at;

ps[min\_index].wt = ps[min\_index].tat - ps[min\_index].bt;

ps[min\_index].rt = ps[min\_index].wt;

sum\_tat +=ps[min\_index].tat;

sum\_wt += ps[min\_index].wt;

sum\_rt += ps[min\_index].rt;

total\_idle\_time += (is\_first\_process==true) ? 0 : (ps[min\_index].start\_time - prev);

completed++;

is\_completed[min\_index]=true;

current\_time = ps[min\_index].ct;

prev= current\_time;

is\_first\_process = false;

}

}

max\_completion\_time = INT\_MIN;

min\_arrival\_time = INT\_MAX;

for(int i=0;i<n;i++)

{

max\_completion\_time = findmax(max\_completion\_time,ps[i].ct);

min\_arrival\_time = findmin(min\_arrival\_time,ps[i].at);

}

length\_cycle = max\_completion\_time - min\_arrival\_time;

printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n");

for(int i=0;i<n;i++)

printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n",ps[i].pid,ps[i].at,ps[i].bt,ps[i].ct,ps[i].tat,ps[i].wt

,ps[i].rt);

printf("\n");

cpu\_utilization = (float)(length\_cycle - total\_idle\_time)/ length\_cycle;

printf("\nAverage Turn Around time= %f ",(float)sum\_tat/n);

printf("\nAverage Waiting Time= %f ",(float)sum\_wt/n);

printf("\nAverage Response Time= %f ",(float)sum\_rt/n);

printf("\nThroughput= %f",n/(float)length\_cycle);

printf("\nCPU Utilization(Percentage)= %f",cpu\_utilization\*100);

return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**QUESTION 6 : Write a code for LJFS.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Process {

    int pid, at, bt, ct, tat, wt;

};

int comparatorArrival(const void \*a, const void \*b) {

    return ((struct Process \*)a)->at - ((struct Process \*)b)->at;

}

void ljfs(struct Process \*processes, int n) {

    qsort(processes, n, sizeof(struct Process), comparatorArrival);

    int time = 0, completed = 0;

    while (completed < n) {

        int index = -1;

        for (int i = 0; i < n; i++) {

            if (processes[i].at <= time && processes[i].ct == 0) {

                if (index == -1 || processes[i].bt > processes[index].bt) {

                    index = i;

                }

            }

        }

        if (index != -1) {

            time += processes[index].bt;

            processes[index].ct = time;

            processes[index].tat = processes[index].ct - processes[index].at;

            processes[index].wt = processes[index].tat - processes[index].bt;

            completed++;

        } else {

            time++;

        }

    }

}

void display(struct Process \*processes, int n) {

    printf("\nProcess No.\tAT\tBT\tCT\tTAT\tWT\n");

    for (int i = 0; i < n; i++) {

        printf("%d\t\t%d\t%d\t%d\t%d\t%d\n",

               processes[i].pid,

               processes[i].at,

               processes[i].bt,

               processes[i].ct,

               processes[i].tat,

               processes[i].wt);

    }

    float total\_wt = 0, total\_tat = 0;

    for (int i = 0; i < n; i++) {

        total\_wt += processes[i].wt;

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

    }

    float avg\_wt = total\_wt / n;

    float avg\_tat = total\_tat / n;

    int total\_time = processes[n - 1].ct;

    float throughput = (float)n / total\_time;

    float cpu\_utilization = (total\_time - (time - total\_time)) / (float)total\_time \* 100;

    printf("\nAverage Waiting Time: %.2f\n", avg\_wt);

    printf("Average Turnaround Time: %.2f\n", avg\_tat);

    printf("Throughput: %.2f processes/unit time\n", throughput);

    printf("CPU Utilization: %.2f%%\n", cpu\_utilization);

}

int main() {

    int n;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {

        processes[i].pid = i + 1;

        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);

        scanf("%d %d", &processes[i].at, &processes[i].bt);

        processes[i].ct = 0;

    }

    ljfs(processes, n);

    display(processes, n);

    return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**QUESTION 7 : Write a code for LRTF.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Process {

    int pid,at,bt,rt,ct,tat,wt;

};

int comparatorArrival(const void \*a, const void \*b) {

    return ((struct Process \*)a)->at - ((struct Process \*)b)->at;

}

void lrtf(struct Process \*processes, int n, int \*total\_idle\_time) {

    qsort(processes, n, sizeof(struct Process), comparatorArrival);

    int time = 0, completed = 0;

    int last\_completed = -1;

    for (int i = 0; i < n; i++) {

        processes[i].rt = processes[i].bt;

    }

    while (completed < n) {

        int index = -1;

        for (int i = 0; i < n; i++) {

            if (processes[i].at <= time && processes[i].rt > 0) {

                if (index == -1 || processes[i].rt > processes[index].rt) {

                    index = i;

                }

            }

        }

        if (index != -1) {

            processes[index].rt--;

            time++;

            if (processes[index].rt == 0) {

                processes[index].ct = time;

                processes[index].tat = processes[index].ct - processes[index].at;

                processes[index].wt = processes[index].tat - processes[index].bt;

                completed++;

            }

        }

        else {

            time++;

            (\*total\_idle\_time)++;

        }

    }

}

void display(struct Process \*processes, int n, int total\_idle\_time) {

    printf("\nProcess No.\tAT\tBT\tCT\tTAT\tWT\n");

    for (int i = 0; i < n; i++) {

        printf("%d\t\t%d\t%d\t%d\t%d\t%d\n",

               processes[i].pid,

               processes[i].at,

               processes[i].bt,

               processes[i].ct,

               processes[i].tat,

               processes[i].wt);

    }

    float total\_wt = 0, total\_tat = 0;

    for (int i = 0; i < n; i++) {

        total\_wt += processes[i].wt;

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

    }

    float avg\_wt = total\_wt / n;

    float avg\_tat = total\_tat / n;

    int total\_time = processes[n - 1].ct;

    float throughput = (float)n / total\_time;

    float cpu\_utilization = (total\_time - total\_idle\_time) / (float)total\_time \* 100;

    printf("\nAverage Waiting Time: %.2f\n", avg\_wt);

    printf("Average Turnaround Time: %.2f\n", avg\_tat);

    printf("Throughput: %.2f processes/unit time\n", throughput);

    printf("CPU Utilization: %.2f%%\n", cpu\_utilization);

}

int main() {

    int n;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {

        processes[i].pid = i + 1;

        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);

        scanf("%d %d", &processes[i].at, &processes[i].bt);

        processes[i].ct = 0;

    }

    int total\_idle\_time = 0;

    lrtf(processes, n, &total\_idle\_time);

    display(processes, n, total\_idle\_time);

    return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**QUESTION 8 : Write a code for Round Robin Algorithm.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Process {

    int pid,at,bt,ct,tat,wt,rt;

};

void roundRobin(struct Process \*processes, int n, int quantum) {

    int time = 0, completed = 0;

    float total\_wt = 0, total\_tat = 0;

    int total\_idle\_time = 0;

    for (int i = 0; i < n; i++) {

        processes[i].rt = processes[i].bt;

    }

    while (completed < n) {

        int all\_done = 1;

        for (int i = 0; i < n; i++) {

            if (processes[i].at <= time && processes[i].rt > 0) {

                all\_done = 0;

                if (processes[i].rt > quantum) {

                    time += quantum;

                    processes[i].rt -= quantum;

                }

                else {

                    time += processes[i].rt;

                    processes[i].ct = time;

                    processes[i].tat = processes[i].ct - processes[i].at;

                    processes[i].wt = processes[i].tat - processes[i].bt;

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

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

                    processes[i].rt = 0;

                    completed++;

                }

            }

        }

        if (all\_done) {

            time++;

        }

    }

    int total\_time = processes[n - 1].ct;

    float throughput = (float)n / total\_time;

    float cpu\_utilization = ((float)(total\_time - total\_idle\_time) / total\_time) \* 100;

    printf("\nAverage Waiting Time: %.2f\n", total\_wt / n);

    printf("Average Turnaround Time: %.2f\n", total\_tat / n);

    printf("Throughput: %.2f processes/unit time\n", throughput);

    printf("CPU Utilization: %.2f%%\n", cpu\_utilization);

}

void display(struct Process \*processes, int n) {

    printf("\nProcess No.\tAT\tBT\tCT\tTAT\tWT\n");

    for (int i = 0; i < n; i++) {

        printf("%d\t\t%d\t%d\t%d\t%d\t%d\n",

               processes[i].pid,

               processes[i].at,

               processes[i].bt,

               processes[i].ct,

               processes[i].tat,

               processes[i].wt);

    }

}

int main() {

    int n, quantum;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {

        processes[i].pid = i + 1;

        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);

        scanf("%d %d", &processes[i].at, &processes[i].bt);

    }

    printf("Enter Time Quantum: ");

    scanf("%d", &quantum);

    roundRobin(processes, n, quantum);

    display(processes, n);

    return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**QUESTION 8 : Write a code for Priority Scheduling Algorithm.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Process {

    int pid,at,bt,ct,tat,wt,priority;

};

int comparatorPriority(const void \*a, const void \*b) {

    return ((struct Process \*)a)->priority - ((struct Process \*)b)->priority;

}

void priorityScheduling(struct Process \*processes, int n) {

    int time = 0, completed = 0;

    float total\_wt = 0, total\_tat = 0;

    qsort(processes, n, sizeof(struct Process), comparatorPriority);

    while (completed < n) {

        int index = -1;

        for (int i = 0; i < n; i++) {

            if (processes[i].at <= time && processes[i].bt > 0) {

                if (index == -1 || processes[i].priority < processes[index].priority) {

                    index = i;

                }

            }

        }

        if (index != -1) {

            time += processes[index].bt;

            processes[index].ct = time;

            processes[index].tat = processes[index].ct - processes[index].at;

            processes[index].wt = processes[index].tat - processes[index].bt;

            total\_wt += processes[index].wt;

            total\_tat += processes[index].tat;

            processes[index].bt = 0;

            completed++;

        } else {

            time++;

        }

    }

    printf("\nAverage Waiting Time: %.2f\n", total\_wt / n);

    printf("Average Turnaround Time: %.2f\n", total\_tat / n);

}

void display(struct Process \*processes, int n) {

    printf("\nProcess No.\tAT\tBT\tCT\tTAT\tWT\tPriority\n");

    for (int i = 0; i < n; i++) {

        printf("%d\t\t%d\t%d\t%d\t%d\t%d\t%d\n",

               processes[i].pid,

               processes[i].at,

               processes[i].bt,

               processes[i].ct,

               processes[i].tat,

               processes[i].wt,

               processes[i].priority);

    }

}

int main() {

    int n;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {

        processes[i].pid = i + 1;

        printf("Enter Arrival Time, Burst Time, and Priority for Process %d: ", i + 1);

        scanf("%d %d %d", &processes[i].at, &processes[i].bt, &processes[i].priority);

    }

    priorityScheduling(processes, n);

    display(processes, n);

    return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated