

OPERATING SYSTEMS

LAB ASESSMENT – 2

­­

Name: **VIBHU KUMAR SINGH**

Reg. No: **19BCE0215**

Teacher: **Manikandan K.**

**Q12) Write a C program to kill a process by specifying its name rather**

**than its PID.**

**A12)**

**CODE:**

#define \_POSIX\_SOURCE

#include<signal.h>

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

sigset\_t sigset;

int p[2], status;

char c='z';

pid\_t pid;

if (pipe(p) !=0)

perror("pipe() error");

else

{

if ((pid = fork()) == 0)

{

sigemptyset(&sigset);

puts("child is getting parent know he's ready for signal");

write(p[1],&c,1);

puts("child is waiting for signal");

sigsuspend(&sigset);

}

puts("parent is waiting for child to say he's ready for signal");

read(p[0],&c,1);

puts("child has told parent he's ready for signal");

kill(pid, SIGTERM);

wait(&status);

if (WIFSIGNALED(status))

if (WTERMSIG(status)==SIGTERM)

puts("child was ended with a SIGTERM");

else

printf("child vas ended with a %d signal\n", WTERMSIG(status));

else puts("child was not ended with a signal");

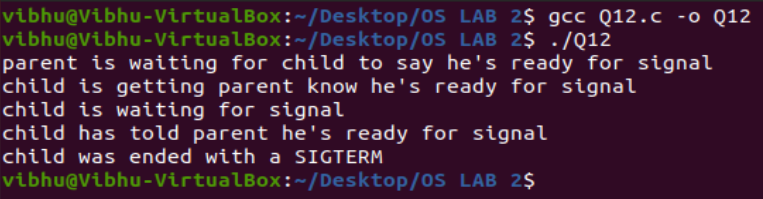
close(p[0]);

close(p[1]);

}

}

**OUTPUT:**



**Q13) Create a file with few lines, Write a C program to read the file and delete the spaces more than one in the file (use UNIX file API’s).**

**A13)**

**CODE:**

#include<stdio.h>

#include<ctype.h>

int main()

{

FILE \* pfile;

int a;

printf("\n Remove the spaces between two words :\n");

printf("-----------------------------------------\n");

pfile=fopen ("file.txt","r");

printf("The content of the file is :\n Hi! I am Vibhu.\n\n");

printf("After removing the spaces the content is : \n");

if (pfile)

{

do

{

a = fgetc (pfile);

if (isgraph(a))

putchar (a);

}while (a != EOF);

fclose (pfile);

}

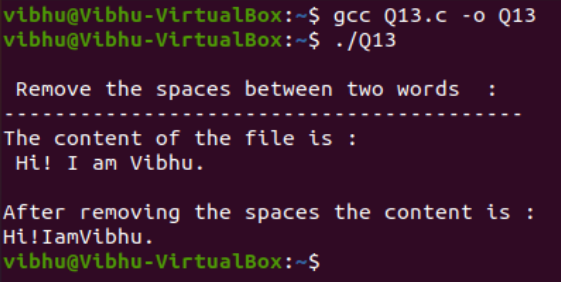
printf("\n");

printf("Spaces Removed");

return 0;

}

**OUTPUT:**



**Q14) Write a program**

**a. To create parent & child process and print their id.**

**b. To create a zombie process.**

**c. To create orphan process a).**

**A14)**

**a)**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

// Driver code

int main()

{

int pid, pid1, pid2;

// variable pid will store the

// value returned from fork() system call

pid = fork();

// If fork() returns zero then it

// means it is child process.

if (pid == 0) {

// First child needs to be printed

// later hence this process is made

// to sleep for 3 seconds.

sleep(3);

// This is first child process

// getpid() gives the process

// id and getppid() gives the

// parent id of that process.

printf("child[1] --> pid = %d and ppid = %d\n",

getpid(), getppid());

}

else {

pid1 = fork();

if (pid1 == 0) {

sleep(2);

printf("child[2] --> pid = %d and ppid = %d\n",

getpid(), getppid());

}

else {

pid2 = fork();

if (pid2 == 0) {

// This is third child which is

// needed to be printed first.

printf("child[3] --> pid = %d and ppid = %d\n",

getpid(), getppid());

}

// If value returned from fork()

// in not zero and >0 that means

// this is parent process.

else {

// This is asked to be printed at last

// hence made to sleep for 3 seconds.

sleep(3);

printf("parent --> pid = %d\n", getpid());

}

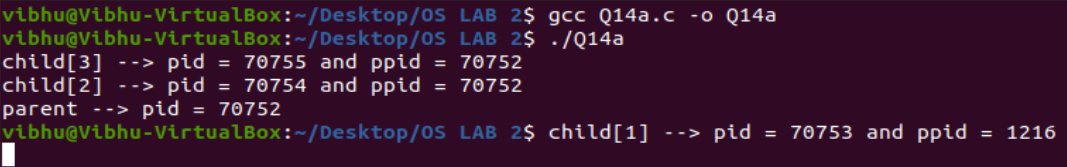
}

}

return 0;

}

**OUTPUT:**



**b)**

**CODE:**

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Fork returns process id

// in parent process

pid\_t child\_pid = fork();

// Parent process

if (child\_pid > 0)

sleep(20);

// Child process

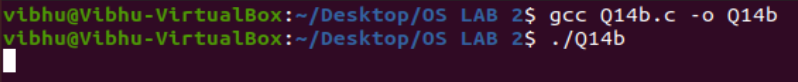
else

exit(0);

return 0;

}

**OUTPUT:**



**c)**

**CODE:**

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Create a child process

int pid = fork();

if (pid > 0)

printf("in parent process\n");

// Note that pid is 0 in child process

// and negative if fork() fails

else if (pid == 0)

{

sleep(30);

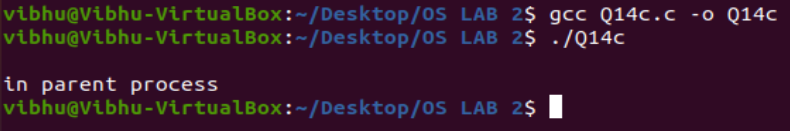
printf("in child process");

}

return 0;

}

**OUTPUT:**



**Q15. Write a program**

**a. To make the process to sleep for few seconds.**

**b. To create a background process.**

**A15)**

**a)**

**CODE:**

#include<stdlib.h>

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

void makeitsleep(int x)

{

sleep(x);

}

int main()

{

printf("How much time do you want me to sleep?\n");

int n; scanf("%d",&n);

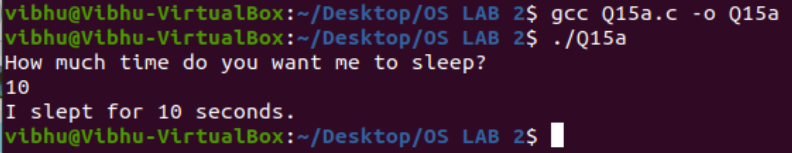
makeitsleep(n);

printf("I slept for %d seconds.\n",n);

return(0);

}

**OUTPUT:**



**b)**

**CODE:**

#include<stdlib.h>

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

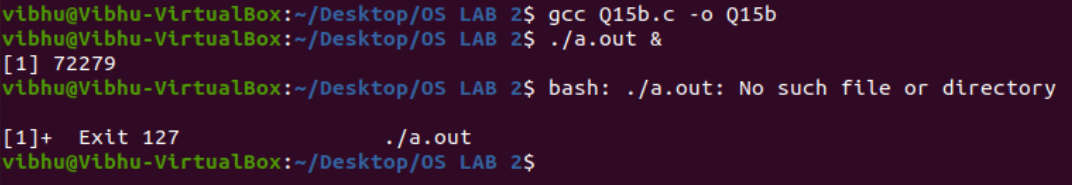
int main()

{

printf("The process ID is: %d\n",getpid());

}

**OUTPUT:**



**Q16) Write a program to create a thread and let the thread check whether**

**the given number is prime or not.**

**A16)**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void \*primeNumbers(void \*vargp)

{

int n, i, m = 0,flag = 0;

printf("Enter the number to check whether it's PRIME or not : ");

scanf("%d",&n);

m = n/2;

for(i = 2 ; i <= m ; i++)

{

if(n%i == 0)

{

printf("The number %d is not PRIME.\n",n);

flag = 1;

break;

}

}

if(flag == 0)

{

printf("The number %d is PRIME.\n",n);

}

return 0;

}

int main()

{

pthread\_t thread\_id;

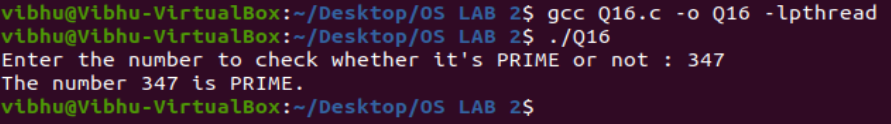
pthread\_create(&thread\_id, NULL, primeNumbers, NULL);

pthread\_join(thread\_id, NULL);

return 0;

}

**OUTPUT:**



**Q17) Design the following CPU Scheduling Algorithms to provide the**

**performance analysis among them.**

**a. FCFS**

**b. PRIORITY**

**c. ROUND ROBIN**

**d. SJF FCFS**

**A17)**

**a)**

**CODE:**

#include<stdio.h>

int main()

{

Int n,burst\_time[50],waiting\_time[50],turnaround\_time[50],avg\_waiting\_time=0,avg\_turnaround\_time=0,i,j;

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

scanf("%d",&n);

printf("\nEnter Process Burst Time: ");

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

{

printf("P[%d]:",i+1);

scanf("%d",&burst\_time[i]);

}

waiting\_time[0]=0;

for(i=1;i<n;i++)

{

waiting\_time[i]=0;

for(j=0;j<i;j++)

waiting\_time[i]+=burst\_time[j];

}

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");

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

{

turnaround\_time[i]=burst\_time[i]+waiting\_time[i];

avg\_waiting\_time+=waiting\_time[i];

avg\_turnaround\_time+=turnaround\_time[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,burst\_time[i],waiting\_time[i],turnaround\_time[i]);

}

avg\_waiting\_time/=i;

avg\_turnaround\_time/=i;

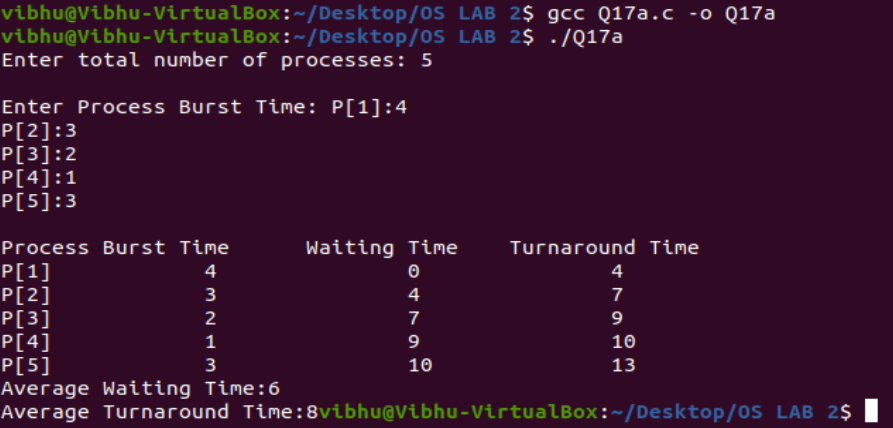
printf("\nAverage Waiting Time:%d",avg\_waiting\_time);

printf("\nAverage Turnaround Time:%d",avg\_turnaround\_time);

return 0;

}

**OUTPUT:**



**b)**

**CODE:**

#include <stdio.h>

int main()

{

int bt[20],wt[20],p[20],tat[20],priority[20];

float avwt=0,avtat=0;

int i,j,n,temp,key;

printf("\nEnter the number of the processes: ");

scanf("%d",&n);

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

{

printf("\nEnter the burst time and priority of the process P[%d]: ",i);

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

scanf("%d",&priority[i]);

p[i]=i;

}

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

{

key=i;

for(j=i+1;j<n;j++)

{

if(priority[j]<priority[key])

{

key=j;

}

}

temp=bt[i];

bt[i]=bt[key];

bt[key]=temp;

temp=priority[i];

priority[i]=priority[key];

priority[key]=temp;

temp=p[i];

p[i]=p[key];

p[key]=temp;

}

wt[0]=0;

tat[0]=bt[0];

avtat=tat[0];

for(i=1;i<n;i++)

{

wt[i]=wt[i-1]+bt[i-1];

tat[i]=tat[i-1]+bt[i];

avwt+=wt[i];

avtat+=tat[i];

}

avwt=avwt/n;

avtat=avtat/n;

printf("\n\nPROCESS\t\twaiting time\tburst time\tTurnaround time\n");

printf("\n");

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

{

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

}

printf("\n\nAverage waiting time: %.2f",avwt);

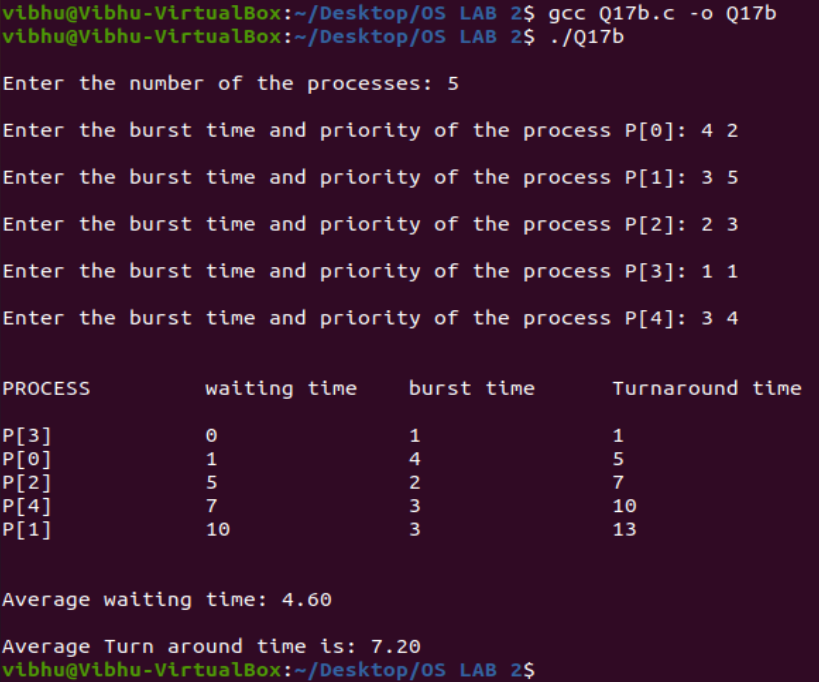
printf("\n\nAverage Turn around time is: %.2f",avtat);

printf("\n");

return 0;

}

**OUTPUT:**



**c)**

**CODE:**

#include<stdio.h>

int main()

{

int limit,count = 0,a,i,count1 = 0,time\_quantum;

int waiting\_time = 0, turnaround\_time = 0, arrival\_time[50], burst\_time[50], temp[50];

float average\_waiting\_time, average\_turnaround\_time;

printf("Enter count Number of Processes: ");

scanf("%d", &limit);

a = limit;

for(i = 0; i < limit; i++)

{

printf("\nEnter the Process[%d]", i + 1);

printf("\nArrival Time: ");

scanf("%d", &arrival\_time[i]);

printf("\nBurst Time: ");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum: ");

scanf("%d", &time\_quantum);

printf("\nProcess ID\tBurst Time\tTurnaround Time\t\tWaiting Time");

for(count = 0, i = 0; a != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

count = count + temp[i];

temp[i] = 0;

count1 = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

count = count + time\_quantum;

}

if(temp[i] == 0 && count1 == 1)

{

a--;

printf("\nProcess[%d]\t%d\t\t\t%d\t\t\t%d", i + 1, burst\_time[i], count - arrival\_time[i], count - arrival\_time[i] - burst\_time[i]);

waiting\_time = waiting\_time + count - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + count - arrival\_time[i];

count1 = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= count)

{

i++;

}

else

{

i = 0;

}

}

average\_waiting\_time = waiting\_time \* 1.0 / limit;

average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

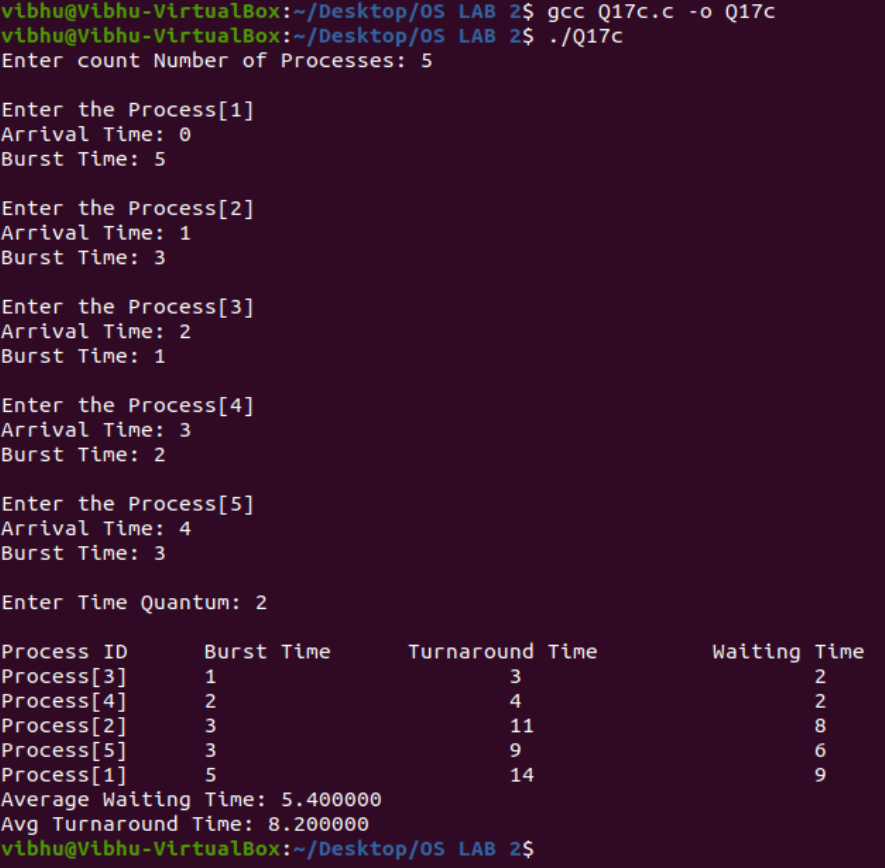
printf("\nAverage Waiting Time: %f", average\_waiting\_time);

printf("\nAvg Turnaround Time: %f", average\_turnaround\_time);

return 0;

}

**OUTPUT:**



**d)**

**CODE:**

#include<stdio.h>

int main()

{

int n,burst\_time[50],waiting\_time[50],turnaround\_time[50],avg\_waiting\_time=0,avg\_turnaround\_time=0,i,j,choice,temp,p[50];

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

scanf("%d",&n);

printf("\nEnter Process Burst Time");

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

{

printf("P[%d]:",i+1);

scanf("%d",&burst\_time[i]);

p[i]=i+1;

}

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

{

for(j=i+1;j<n;j++)

{

if(burst\_time[j]<burst\_time[i])

{

temp=burst\_time[i];

burst\_time[i]=burst\_time[j];

burst\_time[j]=temp;

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

waiting\_time[0]=0;

for(i=1;i<n;i++)

{

waiting\_time[i]=0;

for(j=0;j<i;j++)

waiting\_time[i]+=burst\_time[j];

}

printf("\nProcess Burst Time Waiting Time Turnaround Time");

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

{

turnaround\_time[i]=burst\_time[i]+waiting\_time[i];

avg\_waiting\_time+=waiting\_time[i];

avg\_turnaround\_time+=turnaround\_time[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,burst\_time[i],waiting\_time[i],turnaround\_time[i]);

}

avg\_waiting\_time/=i;

avg\_turnaround\_time/=i;

printf("\n\n Average Waiting Time:%d",avg\_waiting\_time);

printf("\nAverage Turnaround Time:%d",avg\_turnaround\_time);

return 0;

}

**OUTPUT:**

