7) Write a C program to kill a process by specifying its name rather than its PID.

CODE

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

int main() {

printf("\nList of processes:\n");

system("ps -aux");

char command[]="pkill -9 ";

char process\_name[100];

printf("\nEnter the name of the process to be killed:");

scanf("%s",&process\_name);

char P[100];

strcpy (P,process\_name);

strcat(command,process\_name);

printf("\n");

system (command);

printf("\nList of processes after killing '%s' process.\n",P);

system("ps -aux");

return 0;

}

OUTPUT

8) 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).

CODE

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

void remSpace(char \*str,char \*retstr){

int i;

int counter=0;

for (i=0;str[i];i++){

if (str[i]!=' '){

retstr[counter]=str[i];

counter++;

}

else if (str[i]==' '){

retstr[counter]=' ';

while(str[i+1]==' '){

i++;

}

counter++;

}

}

}

int main() {

int fd;

char buffer[80];

char retbuffer[80];

fd=open("test.txt",O\_RDWR);

printf("fd=%d",fd);

if (fd!=-1){

printf("\ntest.txt opened with read and write access.\n");

lseek(fd,0,SEEK\_SET);

read(fd,buffer,50);

printf("\nText inside the Document is:%s",buffer);

remSpace(buffer,retbuffer);

lseek(fd,0,SEEK\_SET);

write(fd,retbuffer,sizeof(retbuffer));

lseek(fd,0,SEEK\_SET);

read(fd,buffer,sizeof(retbuffer));

printf("\nReading text from the file after removing more than one spaces: %s",buffer);

printf("\n\n");

close(fd);

}

return 0;

}

9) Write a program

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

CODE

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

pid\_t pid;

pid=fork();

if (pid==0){

printf("\n(i)\n\n");

printf("\nChild process created with Process ID: %d",getpid());

printf("\nChild process created with parent process ID: %d",getppid());

exit(0);

}

else if(pid>0) {

wait(0);

printf("\nParent process created with Process ID: %d",getpid());

printf("\n");

}

return 0;

}

b) . To create a zombie process.

CODE

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

pid\_t pid;

pid=fork();

if (pid==0){

printf("\n(ii)\n\n");

printf("\nThis is Child Process with process ID: %d",getpid());

printf("\nParent Process ID of this child process is: %d",getppid());

printf("\nThe child process is now converted to a zombie process because its parent process is alive.");

exit(0);

}

else if(pid>0){

sleep(1);

printf("\nParent process id is:%d",getpid());

}

return 0;

}

OUTPUT

c) To create orphan process

CODE

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

pid\_t pid;

pid=fork();

if (pid==0){

sleep(1);

printf("\nChild process created with Process ID: %d",getpid());

printf("\nChild process created with parent process ID: %d",getppid());

printf("\nThis is an Orphan process now");

}

else {

printf("\n(iii)\n\n");

printf("\nParent process has Process ID: %d",getpid());

}

return 0;

}

OUTPUT

10) Write a program

a) To make the process to sleep for a few seconds

CODE

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

int main() {

pid\_t pid;

int n;

pid=getpid();

printf("\nThe Process ID is:%d",pid);

printf("\nEnter the time in seconds for which you want to hibernate this process:");

scanf("%d",&n);

printf("\nThe Process will now sleep for %d second(s)",n);

sleep(n);

char str[3];

sprintf(str,"%d",pid);

char addr[]="cat /proc/";

char ess[]="/status";

strcat(addr,str);

int r = system(addr);

sleep(1);

return 0;

}

OUTPUT

b) To create background process

CODE

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

int main() {

pid\_t pid;

pid=fork();

if (pid==0){

printf("\nPID of background process:%d",getpid());

printf("\nBackground Process is now killed");

sleep(10);

exit(0);

}

else{

printf("\nThis is Parent process working in foreground.\nChild Process is working in background for 10s.");

printf("\nForeground Process Killed");

sleep(1);

kill(getpid());

}

return 0;

}

OUTPUT

11) Implement the program to pass messages using pipes.

CODE

#include<stdio.h>

#include<unistd.h>

int main()

{

int pipefds[2];

int returnstatus;

char writemessages[2][20]={"Hello", "World"};

char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n");

return 1;

}

printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);

write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Reading from pipe – Message 1 is %s\n", readmessage);

printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);

write(pipefds[1], writemessages[1], sizeof(writemessages[0]));

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Reading from pipe – Message 2 is %s\n", readmessage);

return 0;

}

OUTPUT

12) Write a program to demonstrate the implementation of Inter Process Communication (IPC) using shared memory.

CODE (Writer)

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void \*shared\_memory;

char buff[100];

int shmid;

shmid=shmget((key\_t)2345, 1024, 0666|IPC\_CREAT);

printf("Key of shared memory is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0);

printf("Process attached at %p\n",shared\_memory);

printf("Enter some data to write to shared memory\n");

read(0,buff,100);

strcpy(shared\_memory,buff);

printf("You wrote : %s\n",(char \*)shared\_memory);

}

OUTPUT (Writer)

CODE (Reader)

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void \*shared\_memory;

char buff[100];

int shmid;

shmid=shmget((key\_t)2345, 1024, 0666);

printf("Key of shared memory is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0);

printf("Process attached at %p\n",shared\_memory);

printf("Data read from shared memory is : %s\n",(char \*)shared\_memory);

}

OUTPUT (Reader)

13) Write a program to create a thread and let the thread check whether the given number is prime or not.

CODE

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

void \*Prime(void\* var){

printf("\nThis is thread process with pid:%d",getpid());

int flag=0;

int n=(int \*)var;

for(int i=2;i<=n/2;i++){

if(n%i==0){

flag =1;

break;

}

}

if (n==1) {

printf("\n1 is neither prime nor composite");

}

else {

if (flag==0) {

printf("\n%d is Prime.\n",n);

}

else {

printf("\n%d is Composite.\n",n);

}

}

}

int main() {

int n;

printf("\nThis is parent process with pid:%d",getpid());

printf("\nBefore Thread Creation.\nThread will now be Created...");

printf("\nEnter the number which u want to check for prime: ");

scanf("%d",&n);

pthread\_t tid;

pthread\_create (&tid,NULL,Prime,(void\*)n);

pthread\_join(tid,NULL);

pthread\_exit(NULL);

exit(0);

}

OUTPUT

14) Design the following CPU Scheduling Algorithms to provide the performance analysis among them.

a) FCFS

CODE

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int CT;

int TAT;

int WT;

};

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Pno);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

}

}

void calcCT(struct Process Process\_Array[],int n){

int timeline=0;

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

if (timeline<Process\_Array[i].AT) {

timeline=Process\_Array[i].AT;

}

timeline = timeline + Process\_Array[i].BT;

Process\_Array[i].CT = timeline;

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f\n", (sumWT/n));

}

int main(){

int n;

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

scanf ("%d",&n);

struct Process Process\_Array[100];

getStats(Process\_Array, n);

calcCT(Process\_Array, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

calcAvgTAT(Process\_Array, n);

calcAvgWT(Process\_Array, n);

return 0;

}

OUTPUT

b) PRIORITY

CODE

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int rem\_BT;

int CT;

int TAT;

int WT;

int Priority;

};

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Pno);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("Priority=%d\n",Process\_Array[i].Priority);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

Process\_Array[i].rem\_BT=Process\_Array[i].BT;

printf("Enter Priority of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].Priority);

}

}

int readyQueueManagement(struct Process Process\_Array[], struct Process Ready[],int timeline, int n){

int i=0, j=0;

while (Process\_Array[i].AT<=timeline && i<n){

if (Process\_Array[i].rem\_BT !=0){

Ready[j]=Process\_Array[i];

j++;

}

i++;

}

return j;

}

int MaxPriority(struct Process Ready[], int j){

int max=0;

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

if (Ready[i].Priority>Ready[max].Priority){

max=i;

}

}

return max;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[],int n){

int timeline=0;

int counter = 0;

while (counter !=n) {

int j=readyQueueManagement(Process\_Array, Ready, timeline, n);

if (j==0){

timeline++;

}

else{

int max= MaxPriority(Ready, j);

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

if (Ready[max].p\_id==Process\_Array[i].p\_id){

Process\_Array[i].rem\_BT=Process\_Array[i].rem\_BT-1;

timeline=timeline+1;

if (Process\_Array[i].rem\_BT==0){

Process\_Array[i].CT=timeline;

counter++;

}

}

}

}

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));

}

int main(){

int n;

printf("\nEnter the number of Processes in the system:");

scanf ("%d",&n);

struct Process Process\_Array[100];

struct Process Ready[100];

getStats(Process\_Array, n);

calcCT(Process\_Array,Ready, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

calcAvgTAT(Process\_Array, n);

calcAvgWT(Process\_Array, n);

return 0;

}

OUTPUT

c) ROUND ROBIN

CODE

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int rem\_BT;

int CT;

int TAT;

int WT;

}buffer;

void SeT(struct Process Ready[], int n){

buffer.p\_id=-1;

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

Ready[i].p\_id=-1;

Ready[i].AT=-1;

Ready[i].BT=-1;

Ready[i].rem\_BT=-1;

Ready[i].CT=-1;

Ready[i].TAT=-1;

Ready[i].WT=-1;

}

}

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Process\_Array[i].p\_id);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

printf("Rem\_BT=%d\n",Process\_Array[i].rem\_BT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

Process\_Array[i].rem\_BT=Process\_Array[i].BT;

}

}

void queue(struct Process Ready[], struct Process Process\_Array){

int i=0;

while(Ready[i].p\_id!=-1){

i++;

}

Ready[i]=Process\_Array;

}

struct Process dequeue(struct Process Ready[]){

struct Process temp=Ready[0];

int i=0;

while(Ready[i].p\_id!=-1){

i++;

}

int j=0;

for (j; j<i+1;j++){

Ready[j]=Ready[j+1];

}

return temp;

}

int Ready\_No(struct Process Ready[]){

int i=0;

while(Ready[i].p\_id!=-1){

i++;

}

return i;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[], int Time\_Quantum, int n){

int timeline=0;

int counter=0;

int Process\_counter=0;

while(counter !=n){

for(int i=Process\_counter;i<n;i++){

if(Process\_Array[i].AT<=timeline){

queue(Ready, Process\_Array[i]);

Process\_counter++;

}

}

int ReadyQueueCheck=Ready\_No(Ready);

if(ReadyQueueCheck==0){

timeline++;

}

else{

struct Process temp=dequeue(Ready);

if(temp.rem\_BT<=Time\_Quantum){

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

if(Process\_Array[i].p\_id==temp.p\_id){

timeline=timeline+Process\_Array[i].rem\_BT;

Process\_Array[i].rem\_BT=0;

Process\_Array[i].CT=timeline;

counter++;

}

}

}

else{

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

if(Process\_Array[i].p\_id==temp.p\_id){

timeline=timeline+Time\_Quantum;

for(int i=Process\_counter;i<n;i++){

if(Process\_Array[i].AT<=timeline){

queue(Ready, Process\_Array[i]);

Process\_counter++;

}

}

Process\_Array[i].rem\_BT=Process\_Array[i].rem\_BT-Time\_Quantum;

queue(Ready, Process\_Array[i]);

}

}

}

}

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));

}

int main(){

int n, Time\_Quantum;

printf("\nEnter Number of Processes in the system: ");

scanf ("%d",&n);

printf("Enter Time Quantum: ");

scanf("%d",&Time\_Quantum);

struct Process Process\_Array[100];

struct Process Ready[100];

SeT(Ready, 100);

struct Process temp=dequeue(Ready);

getStats(Process\_Array, n);

calcCT(Process\_Array,Ready, Time\_Quantum, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

calcAvgTAT(Process\_Array, n);

calcAvgWT(Process\_Array, n);

return 0;

}

OUTPUT

d) SJF

CODE

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int CT;

int TAT;

int WT;

int flag;

};

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Pno);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

Process\_Array[i].flag=0;

}

}

int MinBT(struct Process Process\_Array[],int n){

int min=0;

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

if (Process\_Array[i].BT<Process\_Array[min].BT && Process\_Array[i].flag==0){

min=i;

}

}

return min;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[], int n){

int timeline=0;

int counter=0;

int min=0;

int j=0;

while (counter!=n){

j=0;

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

if(Process\_Array[i].AT<=timeline&& Process\_Array[i].flag==0){

Ready[j]=Process\_Array[i];

j++;

}

}

if (j==0){

timeline++;

}

else{

min=MinBT(Ready,j);

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

if (Process\_Array[i].p\_id==Ready[min].p\_id){

timeline=timeline+Process\_Array[i].BT;

Process\_Array[i].CT=timeline;

Process\_Array[i].flag=1;

counter++;

}

}

}

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));

}

int main(){

int n;

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

scanf ("%d",&n);

struct Process Process\_Array[100];

struct Process Ready[100];

getStats(Process\_Array, n);

calcCT(Process\_Array, Ready, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

calcAvgTAT(Process\_Array, n);

calcAvgWT(Process\_Array, n);

return 0;

}

OUTPUT