DAY-1:

OPERATING SYSTEM

Date:2-11-2023

1. Create a new process by invoking the appropriate system call. Get the process identifier of the

currently running process and its respective parent using system calls and display the same

using a C program.

#include<stdio.h>

#include<unistd.h>

int main()

{

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

printf("parent process id:%d\n",getpid());

printf("child process id:%d\n",getpid());

return 0;

}

2. Identify the system calls to copy the content of one file to another and illustrate the same using

a C program.

#include<stdio.h

#include<stdio.h>

#include<stdlib.h>

int main()

{

FILE \*fptr1,\*fptr2;

char filename[100],c;

printf("enter the filename to open for reading\n");

scanf("%s",filename);

fptr1=fopen("D:\1.txt","r");

if(fptr1==NULL)

{

printf("cannot open file %s\n",filename);

exit(0);

}

printf("enter the filename to open for writing\n");

scanf("%s",filename);

fptr2=fopen(filename,"w");

if(fptr2==NULL)

{

printf("cannot open file %s \n",filename);

exit(0);

}

c=fgetc(fptr1);

while(c!=EOF)

{

fputc(c,fptr2);

c=fgetc(fptr1);

}

printf("\n Contents copied to %s",filename);

fclose(fptr1);

fclose(fptr2);

return 0;

}

3. Design a CPU scheduling program with C using First Come First Served technique with the

following considerations.

a. All processes are activated at time 0.

b. Assume that no process waits on I/O devices.

#include <stdio.h>

int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of process: ");

scanf("%d", &n);

printf("Enter Burst Time:\n");

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

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

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

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

index = i;

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

if (A[j][1] < A[index][1])

index = j;

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

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

A[i][2] = 0;

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

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

printf("P BT WT TAT\n");

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

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],A[i][1], A[i][2], A[i][3]);

}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt);

printf("\nAverage Turnaround Time= %f", avg\_tat);

}

4. Construct a scheduling program with C that selects the waiting process with the smallest

execution time to execute next.

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("enter number of process:");

scanf("%d",&n);

printf("nenter burst time:n");

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

{

printf("p%d:",i+1);

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

p[i]=i+1;

}

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];

p[pos]=temp;

}

wt[0]=0;

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;

total=0;

printf("nProcesst Burst Time tWaiting TimetTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

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

}

avg\_tat=(float)total/n;

printf("nnAverage Waiting time=%f",avg\_wt);

printf("nAverage Turnaround Time=%fn",avg\_tat);

}

5. Construct a scheduling program with C that selects the waiting process with the highest

priority to execute next.

#include<stdio.h>

struct priority\_scheduling

{

char process\_name;

int burst\_time;

int waiting\_time;

int turn\_around\_time;

int priority;

};

int main()

{

int number\_of\_process;

int total=0;

struct priority\_scheduling temp\_process;

int ASCII\_number=65;

int position;

float average\_waiting\_time;

float average\_turnaround\_time;

printf("enter the total number of processes: ");

scanf("%d",&number\_of\_process);

struct priority\_scheduling process[number\_of\_process];

printf("\nplease enter the Burst Time and Priority of each process:\n");

for(int i=0;i<number\_of\_process;i++)

{

process[i].process\_name=(char) ASCII\_number;

printf("\nEnter the details of the process %c\n",process[i].process\_name);

printf("enter the burst time: ");

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

printf("enter the priority: ");

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

ASCII\_number++;

}

for(int i=0;i<number\_of\_process;i++)

{

position=i;

for(int j=i+1;j<number\_of\_process;i++)

{

if(process[j].priority>process[position].priority)

position=j;

}

temp\_process=process[i];

process[i]=process[position];

process[position]=temp\_process;

}

process[0].waiting\_time=0;

for(int i=1;i<number\_of\_process;i++)

{

process[i].waiting\_time=0;

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

{

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

}

total+=process[i].waiting\_time;

}

average\_waiting\_time=(float)total/(float)number\_of\_process;

total=0;

printf("\n\nprocess\_name\tburst time \t waiting time \t turnaround time\n");

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

for(int i=0;i<number\_of\_process;i++)

{

process[i].turn\_around\_time=process[i].burst\_time+process[i].waiting\_time;

total+=process[i].turn\_around\_time;

printf("\t %c \t\t %d \t\t %d \t\t %d",process[i].process\_name,process[i].burst\_time,

process[i].waiting\_time,process[i].turn\_around\_time);

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

}

average\_turnaround\_time=(float)total/(float)number\_of\_process;

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

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

return 0;

}

6. Construct a C program to implement pre-emptive priority scheduling algorithm.

#include<conio.h>

int main()

{

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

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

scanf("%d", &NOP);

y = NOP;

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

printf(" Arrival time is: \t");

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

printf(" \nBurst time is: \t");

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

temp[i] = bt[i];

}

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

scanf("%d", &quant);

printf("\n Process No \t\t Burst Time \t\t TAT \t\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("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[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 Average Turn Around Time: \t%f", avg\_wt);

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

getch();

}

7. Construct a C program to implement non-preemptive SJF algorithm.

#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);

}

8. Construct a C program to simulate Round Robin scheduling algorithm with C.

#include <stdio.h>

#define MAX\_PROCESS 10

int main() {

int n, quantum, burst\_time[MAX\_PROCESS], remaining\_burst\_time[MAX\_PROCESS];

int waiting\_time = 0, turnaround\_time = 0, completion\_time[MAX\_PROCESS], arrival\_time[MAX\_PROCESS];

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

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

scanf("%d", &n);

printf("Enter the burst time for each process:\n");

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

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

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

remaining\_burst\_time[i] = burst\_time[i];

}

printf("Enter the time quantum: ");

scanf("%d", &quantum);

printf("Enter the arrival time for each process:\n");

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

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

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

}

int time = 0, completed\_processes = 0;

while (completed\_processes < n) {

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

if (remaining\_burst\_time[i] > 0) {

if (remaining\_burst\_time[i] <= quantum) {

time += remaining\_burst\_time[i];

remaining\_burst\_time[i] = 0;

completion\_time[i] = time;

completed\_processes++;

} else {

time += quantum;

remaining\_burst\_time[i] -= quantum;

}

if (remaining\_burst\_time[i] == 0) {

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

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

}

}

}

}

average\_waiting\_time = (float) waiting\_time / n;

average\_turnaround\_time = (float) turnaround\_time / n;

printf("\nAverage waiting time: %.2f ms\n", average\_waiting\_time);

printf("Average turnaround time: %.2f ms\n", average\_turnaround\_time);

return 0;

}

9. Illustrate the concept of inter-process communication using shared memory with a C program.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main() {

key\_t key = ftok("shared\_memory\_example", 65); // Generate a unique key

int shmid = shmget(key, 1024, 0666 | IPC\_CREAT); // Create a shared memory segment

if (shmid == -1) {

perror("shmget");

exit(1);

}

char \*shared\_memory = (char\*)shmat(shmid, (void\*)0, 0); // Attach the shared memory

if (shared\_memory == (char\*)(-1)) {

perror("shmat");

exit(1);

}

// Write a message to shared memory

char message[] = "Hello, shared memory!";

strcpy(shared\_memory, message);

printf("Message written to shared memory: %s\n", message);

shmdt(shared\_memory); // Detach shared memory

return 0;

}

10. Illustrate the concept of inter-process communication using message queue with a C

program.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct msg\_buffer

{

long msg\_type;

char msg\_text[100];

};

int main() {

key\_t key;

int msgid;

key = ftok("msg\_queue\_example", 65);

if (key == -1) {

perror("ftok");

exit(1);

}

msgid = msgget(key, 0666 | IPC\_CREAT);

if (msgid == -1) {

perror("msgget");

exit(1);

}

struct msg\_buffer message;

message.msg\_type = 1;

strcpy(message.msg\_text, "Hello, message queue!");

if (msgsnd(msgid, &message, sizeof(message), 0) == -1) {

perror("msgsnd");

exit(1);

}

printf("Message sent to the queue: %s\n", message.msg\_text);

return 0;

}