INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **SL.NO** | **DATE** | **PROGRAM NAME** | **PAGE NO** |
|  | **10-06-22** | **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.** |  |
|  | **10-06-22** | **Identify the system calls to copy the content of one file to another and illustrate the same using a C program.** |  |
|  | **11-06-22** | **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.** |  |
|  | **13 -06-22** | **Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.** |  |
|  | **13 -06-22** | **Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.** |  |
|  | **13-06-22** | **Construct a C program to simulate producer-consumer problem using semaphores.** |  |
|  | **15-06-22** | **Construct a program to simulate the First in First Out paging technique of memory management. When a page must be replaced, the oldest page is chosen.** |  |
|  | **15-06-22** | **Construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen** |  |
|  | **15-06-22** | **Construct a program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future** |  |
|  | **17-06-22** | **Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a program to simulate the file allocation strategy.** |  |
|  | **17-06-22** | **Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy.** |  |
|  | **17-06-22** | **With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy.** |  |
|  | **20-06-22** | **Construct a C program to simulate the First Come First Served disk scheduling algorithm.** |  |
|  | **20-06-22** | **Illustrate the various File Access Permission and different types users in Linux** |  |
|  | **20-06-22** | **Study the features and deployment processes of Xen and VMware on Linux.** |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| EXPERIMENT 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 |

AIM:

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.

ALGORITHM:

Step 1 : Start program.

Step 2 : An existing process can create a new one by calling the fork() function.

Step 3 : The new process created by fork() is called the child process.

Step 4 : We are using her getpid()to get the process id.

Step 5 : Stop program..

PROGRAM:

#include<stdio.h>

#include<unistd.h>

int main()

{

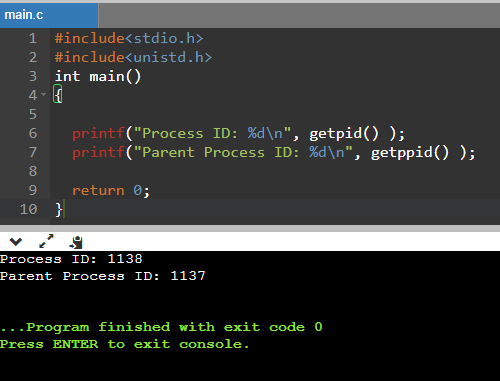
printf("Process ID: %d\n", getpid() );

printf("Parent Process ID: %d\n", getppid() );

return 0;

}

INPUT AND OUTPUT:



RESULT:

Thus the program has been successfully implemented using system call

|  |
| --- |
| EXPERIMENT 2:  Identify the system calls to copy the content of one file to another and illustrate the same using a C program. |

Aim:

To identify the system calls to copy the content of one file to another and illustrate the same using a C program.

ALGORITHM:

Step 1 : Start program.

Step 2 :type the file name that file is readable

Step 3 : type the file name that the content should be copied in the other file

Step 4 : the coping of the content is done

Step 5 : Stop program.

Program:

#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(filename, "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("\nContents copied to %s", filename);

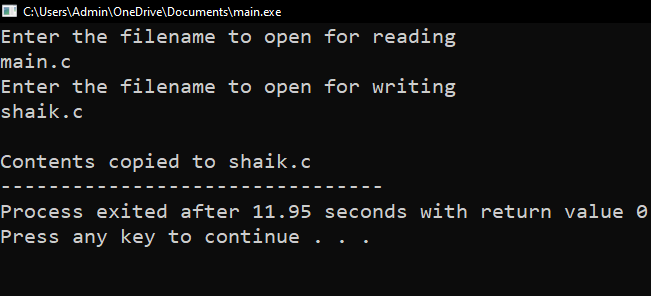
fclose(fptr1);

fclose(fptr2);

return 0;

}

INPUT AND OUTPUT:



Result: Thus the program copying one file to another has been successfully implemented using system calls.

|  |
| --- |
| EXPERIMENT 3:  Desig n 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. |

AIM:

ToDesign 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.

AlGORITHM:

Step 1: start program  
Step 2: Inside the structure declare the variables.

Step 3: Declare the variable i, j as integer, to time and to time is equal to zero.

Step 4: Get the value of „n‟ assign pid as I and get the value of p[i].b time.

Step 5: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.

Step 6: Calculate total wait time and total turnaround time by dividing by total number of process. Step 7: Print total wait time and total turnaround time.

Step 8: Stop the program

PROGRAM:

#include<stdio.h>

void main()

{

int n,bt[20],wt[20],tat[20],i,j; float avwt=0,avtat=0;printf("Enter total number of processes(maximum 20):");scanf("%d",&n);

printf("\nEnter Process Burst Time\n");for(i=0;i<n;i++)

{

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

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

} wt[0]=0;

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

{ wt[i]=0;for(j=0;j<i;j++)

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time"); for(i=0;i<n;i++)

{

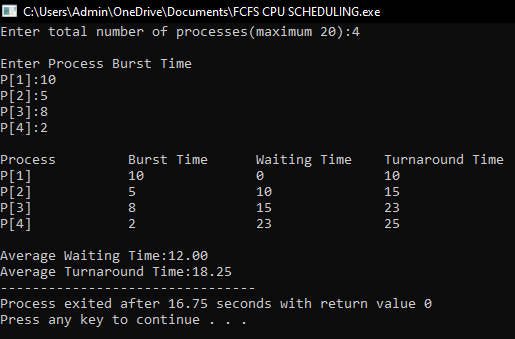
tat[i]=bt[i]+wt[i]; avwt+=wt[i]; avtat+=tat[i];printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

} avwt/=i; avtat/=i;printf("\n\nAverage Waiting Time:%.2f",avwt);

printf("\nAverage Turnaround Time:%.2f",avtat);

}

INPUT AND OUTPUT:



RESULT:

Thus the program of cpu scheduling by using “frist come first serve”technique is implemented successfully..

|  |
| --- |
| **EXPERIMENT 4:**  **Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.** |

AIM:

**Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.**

ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i, j as integer, totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] w time as zero and tot time as btime and inside the loop calculate wait time and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of process. Step 6: Print total wait time and total turnaround time.

Step 7: Stop the program.

PROGRAM:

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp,floatavg\_wt,avg\_tat,avg\_wt;

printf("Enter number of process:");

scanf("%d",&n);

printf("\n Enter 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]; bt[pos]=temp; temp=p[i]; p[i]=p[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("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");for(i=0;i<n;i++)

{

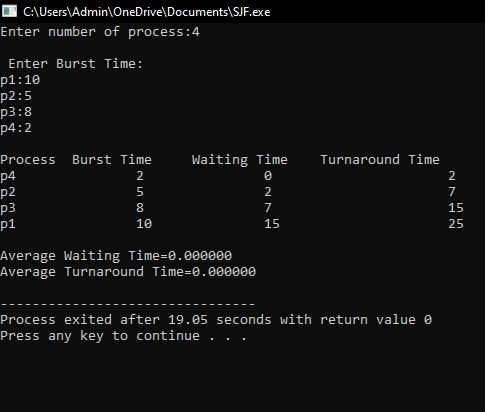
tat[i]=bt[i]+wt[i]; total+=tat[i];printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}avg\_tat=(float)total/n;printf("\n\nAverage Waiting Time=%f",avg\_wt);

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

}

OUTPUT:



RESULT: Thus the c program for **scheduling program with C that selects the waiting process with the smallest execution time to execute next successfully implemented.**

|  |
| --- |
| **EXPERIMENT 5:**  **Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.** |

AIM: To **illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.**

**ALGORITHM:**

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and available matrix.

Step-4: Compare each and every process using the banker‟s algorithm.

Step-5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process

Step-6: produce the result of state of process

Step-7: Stop the program.

PROGRAM:

**#include<stdio.h>**

**int main()**

**{**

**int count = 0, m, n, process, temp, resource; int allocation\_table[5] = {0, 0, 0, 0, 0}; int available[5], current[5][5], maximum\_claim[5][5]; int maximum\_resources[5], running[5], safe\_state = 0; printf("\nEnter The Total Number Of Processes:\t"); scanf("%d", &process); for(m=0;m<process;m++)**

**{ running[m]=1; count++;**

**}**

**printf("\nEnter The Total Number Of Resources To Allocate:\t"); scanf("%d",&resource);**

**printf("\nEnter The Claim Vector:\t"); for(m=0;m<resource;m++)**

**{**

**scanf("%d",&maximum\_resources[m]);**

**}**

**printf("\nEnter Allocated Resource Table:\n"); for(m=0;m<process;m++)**

**{**

**for(n=0;n<resource;n++)**

**{**

**scanf("%d",&current[m][n]);**

**}**

**}**

**printf("\nEnter The Maximum Claim Table:\n");for(m=0;m<process;m++)**

**{**

**for(n=0;n<resource;n++)**

**{**

**scanf("%d",&maximum\_claim[m][n]);**

**} }**

**printf("\nThe Claim Vector \n");**

**for(m=0;m<resource;m++)**

**{printf("\t%d ",maximum\_resources[m]);**

**}**

**printf("\n The Allocated Resource Table\n"); for(m=0;m<process;m++)**

**{for(n=0;n<resource;n++)**

**{printf("\t%d",current[m][n]);**

**} printf("\n");**

**}printf("\nThe Maximum Claim Table \n"); for(m=0;m<process;m++)**

**{for(n=0;n<resource;n++)**

**{printf("\t%d",maximum\_claim[m][n]);**

**} printf("\n");**

**}for(m=0;m<process;m++)**

**{for(n=0;n<resource;n++)**

**{allocation\_table[n]=allocation\_table[n]+current[m][n];**

**}}**

**printf("\nAllocated Resources \n"); for(m=0;m<resource;m++)**

**{printf("\t%d",allocation\_table[m]);**

**}for(m=0;m<resource;m++)**

**{**

**available[m]=maximum\_resources[m]-allocation\_table[m];**

**}**

**printf("\nAvailable Resources:");**

**for(m=0;m<resource;m++)**

**{**

**printf("\t%d",available[m]);**

**} printf("\n"); while(count!=0)**

**{ safe\_state=0;**

**for(m=0;m<process;m++)**

**{**

**if(running[m])**

**{ temp=1;**

**for(n=0;n<resource;n++)**

**{**

**if(maximum\_claim[m][n]-current[m][n]>available[n])**

**{ temp=0; break;**

**}**

**} if(temp)**

**{**

**printf("\nProcess %d Is In Execution \n", m + 1); running[m]=0; count--; safe\_state=1;**

**for(n=0;n<resource;n++)**

**{available[n]=available[n]+current[m][n];**

**} break;**

**}}}if(!safe\_state)**

**{printf("\nThe Processes Are In An Unsafe State \n"); break; } else**

**{printf("\nThe Process Is In A Safe State \n"); printf("\nAvailable Vector\n");**

**for(m=0;m<resource;m++)**

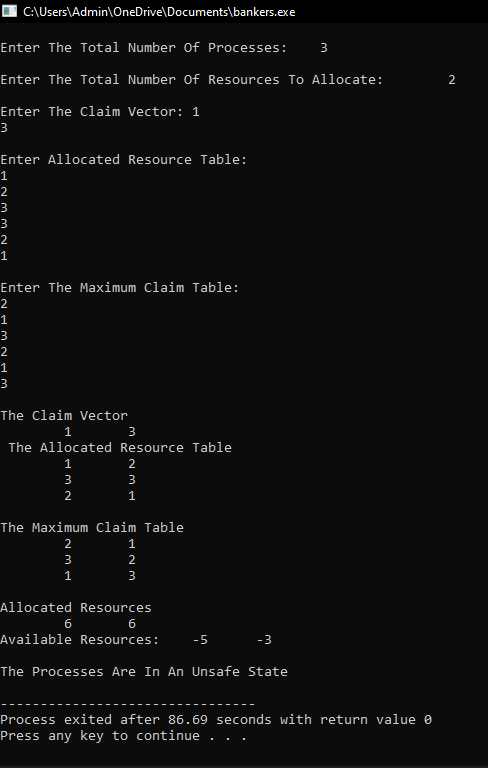
**{printf("\t%d",available[m]);**

**} printf("\n");**

**}}**

**}**

**OUTPUT:**



**RESULT:**

**Thus the program to illustrate the deadlock avoidance concept by simulating Banker’s algorithm is successfully implemented.**

|  |
| --- |
| **EXPERIMENT 6:**  **Construct a C program to simulate producer-consumer problem using semaphores.** |

**AIM:**

**To Construct a C program to simulate producer-consumer problem using semaphores**

**ALGORITHM:**

**PROGRAM:**

**#include<stdio.h>**

**#include<stdlib.h>**

**int mutex=1,full=0,empty=3,x=0;**

**int main()**

**{**

**int n;**

**void producer();**

**void consumer();**

**int wait(int);**

**int signal(int);**

**printf("\n1.Producer\n2.Consumer\n3.Exit");**

**while(1)**

**{**

**printf("\nEnter your choice:");**

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

**switch(n)**

**{**

**case 1: if((mutex==1)&&(empty!=0))**

**producer();**

**else**

**printf("Buffer is full!!");**

**break;**

**case 2: if((mutex==1)&&(full!=0))**

**consumer();**

**else**

**printf("Buffer is empty!!");**

**break;**

**case 3:**

**exit(0);**

**break;**

**}**

**}**

**return 0;**

**}**

**int wait(int s)**

**{**

**return (--s);**

**}**

**int signal(int s)**

**{ return(++s);**

**}void producer()**

**{ mutex=wait(mutex);**

**full=signal(full);**

**empty=wait(empty);**

**x++;**

**printf("\nProducer produces the item %d",x);**

**mutex=signal(mutex);**

**}void consumer()**

**{**

**mutex=wait(mutex);**

**full=wait(full);**

**empty=signal(empty);**

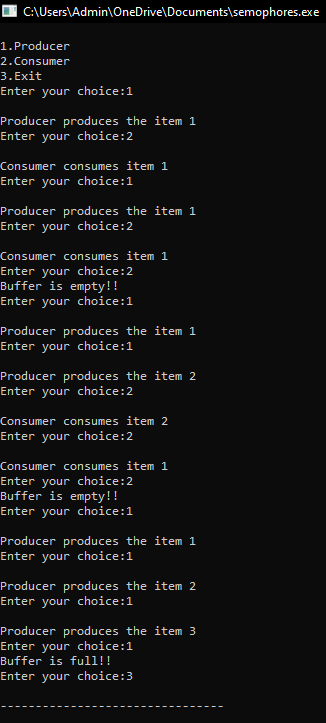
**printf("\nConsumer consumes item %d",x);**

**x--;**

**mutex=signal(mutex);**

**}**

**OUTPUT:**



**RESULT:**

**Thus the C program to simulate producer-consumer problem using semaphores is successfully implemented.**

|  |
| --- |
| **EXPERIMENT 7:**  **Construct a program to simulate the First in First Out paging technique of memory management. When a page must be replaced, the oldest page is chosen.** |

**AIM:  
 To construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen.**

**ALGORITHM:**

Step 1: Start the program.

Step 2: Declare the necessary variables.

Step 3: Enter the number of frames.

Step 4: Enter the reference string end with zero.

Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen. Step 6: When a page is brought into memory, it is inserted at the tail of the queue.

Step 7: Initially all the three frames are empty.

Step 8: The page fault range increases as the no of allocated frames also increases.

Step 9: Print the total number of page faults.

Step 10: Stop the program.

PROGRAM:

#include<stdio.h>

Int main ()

{

int i,j,n,a[50],frame[10],nf,k,avail,count=0;

printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

printf("\n ENTER THE PAGE

NUMBER:\n");for(i=1;i<=n;i++)scanf("%d",&a[i]);

printf("\n ENTER THE NUMBER OF FRAMES :");

scanf("%d",&nf);for(i=0;i<nf;i++)frame[i]= -1;j=0;

printf("\tref string\t page frames\n");for(i=1;i<=n;i++)

{

printf("%d\t\t",a[i]);avail=0;for(k=0;

k<nf;k++)

if(frame[k]==a[i])avail=1;

if (avail==0)

{

frame[j]=a[i];j=(j+1)%nf;c

ount++;for(k=0;k<nf;k++)

printf("%d\t",frame[k]);

}

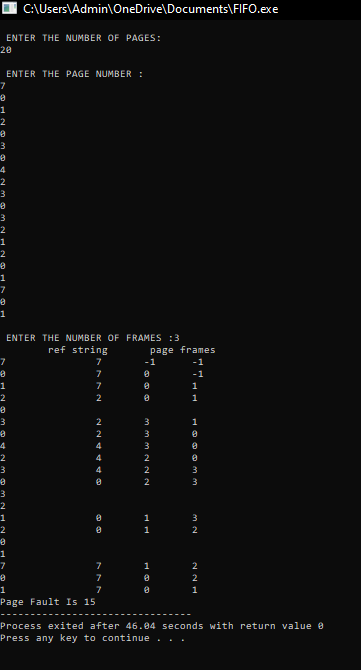
printf("\n");

}

printf("Page Fault Is %d",count);

}

OUTPUT:



RESULT:

Thus the c program to simulate the First in First Out paging technique of memory management. When a page must be replaced, the oldest page is chosen is successfully implemented.

|  |
| --- |
| **EXPERIMENT 8:**  **Construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen** |

AIM:

To Construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the necessary variables.

Step 3: Enter the number of frames.

Step 4: Enter the reference string end with zero.

Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen. Step 6: When a page is brought into memory, it is inserted at the tail of the queue.

Step 7: Initially all the three frames are empty.

Step 8: The page fault range increases as the no of allocated frames also increases.

Step 9: Print the total number of page faults.

Step 10: Stop the program.

**PROGRAM:**

#include<stdio.h>

int findLRU(int time[], int n){int i, minimum = time[0], pos = 0;for(i = 1; i < n; ++i){if(time[i] < minimum){minimum = time[i];pos = i;

}}

return pos;

}

int main()

{int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j,pos, faults = 0;printf("Enter number of frames: ");scanf("%d", &no\_of\_frames);printf("Enter number of pages: ");scanf("%d", &no\_of\_pages);printf("Enter reference string: ");

for(i = 0; i < no\_of\_pages; ++i){scanf("%d", &pages[i]);

}for(i = 0; i < no\_of\_frames; ++i){frames[i] = -1;

}for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;for(j = 0; j < no\_of\_frames; ++j){if(frames[j] == pages[i]){counter++;time[j] = counter;flag1 = flag2 = 1;

break;

}

}if(flag1 == 0){for(j = 0; j < no\_of\_frames; ++j){ if(frames[j] == -1){ counter++; faults++; frames[j] = pages[i]; time[j] = counter; flag2 = 1; break;

}}}if(flag2 == 0){ pos = findLRU(time, no\_of\_frames); counter++; faults++; frames[pos] = pages[i];

time[pos] = counter;

} printf("\n"); for(j = 0; j < no\_of\_frames; ++j){

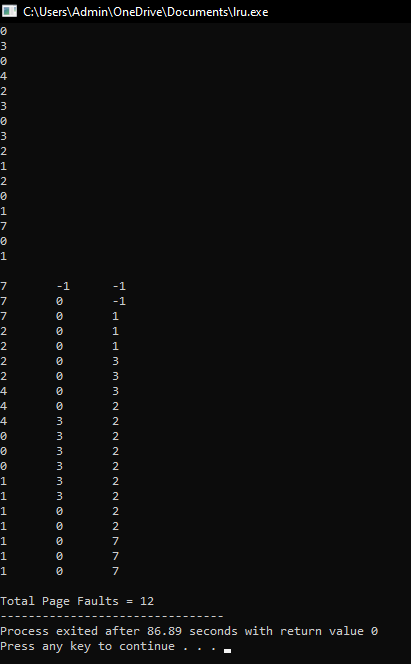
printf("%d\t", frames[j]);

}}

printf("\n\nTotal Page Faults = %d", faults); return 0;

}

OUTPUT:



RESULT: Thus the c program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen is successfully implemented.

|  |
| --- |
| **EXPERIMENT 9:**  **Construct a program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future** |

AIM:

To Construct a program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the necessary variables.

Step 3: Enter the number of frames.

Step 4: Enter the reference string end with zero.

Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen. Step 6: When a page is brought into memory, it is inserted at the tail of the queue.

Step 7: Initially all the three frames are empty.

Step 8: The page fault range increases as the no of allocated frames also increases.

Step 9: Print the total number of page faults.

Step 10: Stop the program.

**PROGRAM:**

#include<stdio.h>

void main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k,pos, max, faults = 0;

printf("Enter number of frames: "); scanf("%d", &no\_of\_frames);

printf("Enter number of pages: "); scanf("%d", &no\_of\_pages);

printf("Enter page reference string: ");

for(i = 0; i < no\_of\_pages; ++i)

{

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

}

for(i = 0; i < no\_of\_frames; ++i){ frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){ flag1 = flag2 = 0; for(j = 0; j < no\_of\_frames; ++j){ if(frames[j] == pages[i]){ flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j)

{

if(frames[j] == -1){faults++;

frames[j] = pages[i];flag2 = 1;

break;

}

}

}

if(flag2 == 0){ flag3 =0;

for(j = 0; j < no\_of\_frames; ++j){ temp[j] = -1; for(k = i + 1; k < no\_of\_pages; ++k){ if(frames[j] == pages[k]){ temp[j] = k; break;

}

}

}

for(j = 0; j < no\_of\_frames; ++j){ if(temp[j] == -1){ pos = j; flag3 = 1; break;

}

} if(flag3 ==0){ max = temp[0];

pos = 0;

for(j = 1; j < no\_of\_frames; ++j){ if(temp[j] > max)

{ max = temp[j]; pos = j;

}

}

}

frames[pos] = pages[i];faults++; } printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

printf("%d\t", frames[j]);

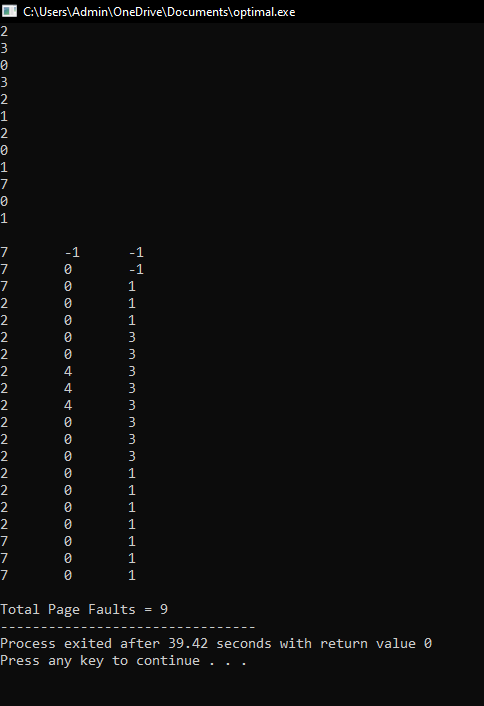
}

}

printf("\n\nTotal Page Faults = %d", faults);

}

OUTPUT:



RESULT:  
 Thus the c program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future is successfully implemented.

|  |
| --- |
| **EXPERIMENT 10:**  **Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a program to simulate the file allocation strategy.** |

AIM:

To Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a program to simulate the file allocation strategy.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Gather information about the number of files.

STEP 3: Gather the memory requirement of each file.

STEP 4: Allocate the memory to the file in a sequential manner.

STEP 5: Select any random location from the available location.

STEP 6: Check if the location that is selected is free or not.

STEP 7: If the location is allocated set the flag = 1.

STEP 8: Print the file number, length, and the block allocated.

STEP 9: Gather information if more files have to be stored.

STEP 10: If yes, then go to STEP 2.

STEP 11: If no, Stop the program.

PROGRAM:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void recurse(int files[]){

int flag = 0, startBlock, len, j, k, ch;

printf("Enter the starting block and the length of the files: ");

scanf("%d%d", &startBlock, &len);

for (j=startBlock; j<(startBlock+len); j++){

if (files[j] == 0)

flag++;

}

if(len == flag){

for (int k=startBlock; k<(startBlock+len); k++){

if (files[k] == 0){

files[k] = 1;

printf("%d\t%d\n", k, files[k]);

}

}

if (k != (startBlock+len-1))

printf("The file is allocated to the disk\n");

}

else

printf("The file is not allocated to the disk\n");

printf("Do you want to enter more files?\n");

printf("Press 1 for YES, 0 for NO: ");

scanf("%d", &ch);

if (ch == 1)

recurse(files);

else

exit(0);

return;

}

int main()

{

int files[50];

for(int i=0;i<50;i++)

files[i]=0;

printf("Files Allocated are :\n");

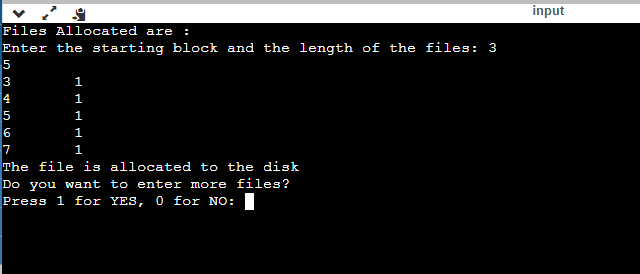
recurse(files);

getch();

return 0;

}

OUTPUT:



RESULT: Thus the c program to file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records.Design a program to simulate the file allocation strategy is successfully implemented.

|  |
| --- |
| **EXPERIMENT 11:**  **Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy.** |

AIM:

To Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Get information about the number of files.

STEP 3: Get the memory requirement of each file.

STEP 4: Allocate the memory to the file by selecting random locations.

STEP 5: Check if the location that is selected is free or not.

STEP 6: If the location is allocated set the flag = 1, and if free set flag = 0.

STEP 7: Print the file number, length, and the block allocated.

STEP 8: Gather information if more files have to be stored.

STEP 9: If yes, then go to STEP 2.

STEP 10: If no, Stop the program.

PROGRAM:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int files[50], indexBlock[50], indBlock, n;

void recurse1();

void recurse2();

void recurse1(){

printf("Enter the index block: ");

scanf("%d", &indBlock);

if (files[indBlock] != 1){

printf("Enter the number of blocks and the number of files needed for the index %d on the disk: ", indBlock);

scanf("%d", &n);

}

else{

printf("%d is already allocated\n", indBlock);

recurse1();

}

recurse2();

}

void recurse2(){

int ch;

int flag = 0;

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

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

if (files[indexBlock[i]] == 0)

flag++;

}

if (flag == n){

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

files[indexBlock[j]] = 1;

}

printf("Allocated\n");

printf("File Indexed\n");

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

{

printf("%d ------> %d : %d\n", indBlock, indexBlock[k], files[indexBlock[k]]);

}

}

else{

printf("File in the index is already allocated\n");

printf("Enter another indexed file\n");

recurse2();

}

printf("Do you want to enter more files?\n");

printf("Enter 1 for Yes, Enter 0 for No: ");

scanf("%d", &ch);

if (ch == 1)

recurse1();

else

exit(0);

return;

}

int main()

{

for(int i=0;i<50;i++)

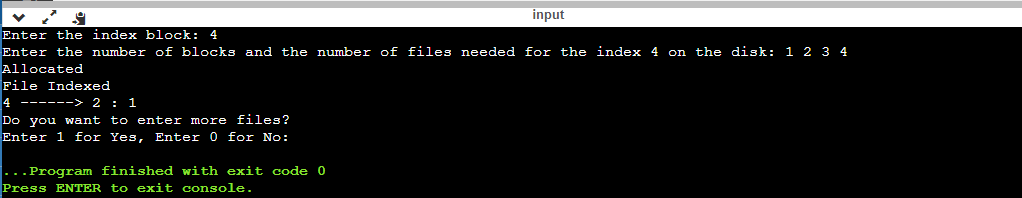
files[i]=0;

recurse1();

return 0;

}

OUTPUT:



RESULT:

Thus the c program to file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy is successfully implemented.

|  |
| --- |
| **EXPERIMENT 12:**  **With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy.** |

AIM:

To do With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Gather information about the number of files.

STEP 3: Allocate random locations to the files.

STEP 4: Check if the location that is selected is free or not.

STEP 5: If the location is free set the flag=0 a location is allocated set the flag .

STEP 6: Print the file number, length, and the block allocated.

STEP 7: Gather information if more files have to be stored.

STEP 8: If yes, then go to STEP 2.

STEP 9: If no, Stop the program.

PROGRAM:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void recursivePart(int pages[]){

int st, len, k, c, j;

printf("Enter the index of the starting block and its length: ");

scanf("%d%d", &st, &len);

k = len;

if (pages[st] == 0){

for (j = st; j < (st + k); j++){

if (pages[j] == 0){

pages[j] = 1;

printf("%d------>%d\n", j, pages[j]);

}

else {

printf("The block %d is already allocated \n", j);

k++;

}

}

}

else

printf("The block %d is already allocated \n", st);

printf("Do you want to enter more files? \n");

printf("Enter 1 for Yes, Enter 0 for No: ");

scanf("%d", &c);

if (c==1)

recursivePart(pages);

else

exit(0);

return;

}

int main(){

int pages[50], p, a;

for (int i = 0; i < 50; i++)

pages[i] = 0;

printf("Enter the number of blocks already allocated: ");

scanf("%d", &p);

printf("Enter the blocks already allocated: ");

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

scanf("%d", &a);

pages[a] = 1;

}

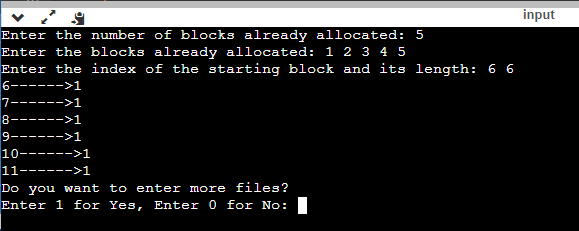
recursivePart(pages);

getch();

return 0;

}

OUTPUT:



RESULT:

Thus the c program to do With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy is successfully implemented.

|  |
| --- |
| **EXPERIMENT 13:**  **Construct a C program to simulate the First Come First Served disk scheduling algorithm.** |

AIM:

To Construct a C program to simulate the First Come First Served disk scheduling algorithm.

ALGORITHM:

Step 1 : start the program.

Step 2 : Let request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arraival “head “ is position of disk head.

Step 3: Let us one by one take the tacks in default order and calculate the absolute distance of the track from the head.

Step 4 : increment the total seek count with this distance.

Step 5:Currently serviced track position now becomes the new head position.

Step 6:Go to step 2 until all tracks in request array have not been serviced.

Step 7:Stop the program.

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

// logic for FCFS disk scheduling

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

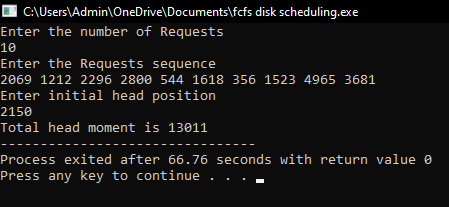
}

printf("Total head moment is %d",TotalHeadMoment);

return 0;

}

OUTPUT:



RESULT: Thus the C program to simulate the First Come First Served disk scheduling algorithm is successfully implemented.

|  |
| --- |
| **EXPERIMENT 14:**  **Illustrate the various File Access Permission and different types users in Linux** |

AIM:

To illustrate the various File Access Permission and different types users in Linux.

1. C program to writing into a file.

AIM:

To write a simple c program to write contents into a file.

ALGORITHM:

Step 1: Start.

Step 2: Open an input file in write mode.

Step 3: Get the contents from the user.

Step 4: Write the contents into the file.

Step 5: Close the file.

Step 6: Stop.

PROGRAM:

# include <stdio.h>

# include <string.h>

void main( )

{ FILE \*fp ;

char data[50];

printf( "Opening the file test.c in write mode" ) ;

fp = fopen("test.txt", "w") ;

if ( fp == NULL )

{

printf( "Could not open file test.txt" ) ;

} printf( "\n Enter some text from keyboard to write in the file test.txt" ) ; while ( strlen ( gets( data ) ) > 0 )

{ fputs(data, fp) ;

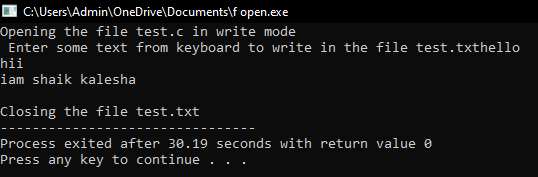
fputs("\n", fp) ;

} printf("Closing the file test.txt") ;

fclose(fp) ;

}

OUTPUT:



2. C program to read the file.

AIM:

To write a simple c program to read the contents of a file and display the same.

ALGORITHM:

Step 1: Start.

Step 2: Open an input file in read mode.

Step 3: Read the contents from the file.

Step 4: Display the contents in the screen.

Step 5: Close the file.

Step 6: Stop.

PROGRAM:

# include <stdio.h>

void main( )

{ FILE \*fp ;

char data[50] ;

printf( "Opening the file test.txt in read mode" ) ;

fp = fopen( "test.txt", "r" ) ;

if ( fp == NULL )

{ printf( "Could not open file test.txt" ) ;

} printf( "Reading the file test.txt" ) ;

while( fgets ( data, 50, fp ) != NULL )

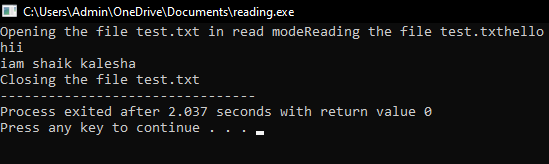
printf( "%s" , data ) ;

printf("Closing the file test.txt") ;

fclose(fp) ;

}

OUT PUT:



3. C program to reversing a file.

AIM:

To write a simple c program for reversing the file contents.

ALGORITHM:

STEP 1:START

STEP 2:Create a file named reverse.c

STEP 3:Get the data from the user.

STEP 4:Store the data in the file.

STEP 5:Reverse the data.

STEP 6:STOP.

PROGRAM:

#include <stdio.h>

void main()

{

FILE \*fp;

char ch;

int i,pos;

fp=fopen("test.txt","r");

if(fp==NULL)

{

printf("file doesnot exist \n");

}

fseek(fp,0,SEEK\_END);

pos=ftell(fp);

i=0;

while(i<pos)

{

i++;

fseek(fp,-i,SEEK\_END);

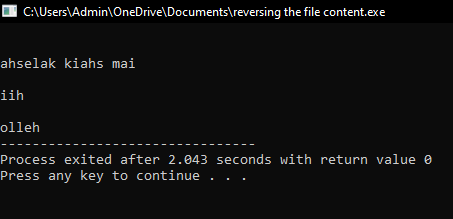
ch=fgetc(fp);

printf("%c",ch);

}

}

OUT PUT:



RESULT: Thus the illustration of the various file access permission and different types users in linux is successfully implemented.

|  |
| --- |
| **EXPERIMENT 15:**  **Study the features and deployment processes of Xen and VMware on Linux.** |

AIM:

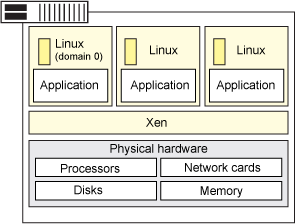
To **Study the features and deployment processes of Xen and VMware on Linux**

XEN:

Xen is a type 1 hypervisor that creates logical pools of system resources so that many virtual machines can share the same physical resources.

Xen is a hypervisor that runs directly on the system hardware. Xen inserts a virtualization layer between the system hardware and the virtual machines, turning the system hardware into a pool of logical computing resources that Xen can dynamically allocate to any guest operating system. The operating systems running in virtual machines interact with the virtual resources as if they were physical resources.

FIGURE 1. THE XEN ARCHITECTURE



Xen is running three virtual machines. Each virtual machine is running a guest operating system and applications independent of other virtual machines while sharing the same physical resources.

Features:

The following are key concepts of the Xen architecture:

• Full virtualization.

• Xen can run multiple guest OS, each in its on VM.

• Instead of a driver, lots of great stuff happens in the Xen daemon, x end.

FULL VIRTUALIZATION

Most hypervisors are based on full virtualization which means that they completely emulate all hardware devices to the virtual machines. Guest operating systems do not require any modification and behave as if they each have exclusive access to the entire system.

Full virtualization often includes performance drawbacks because complete emulation usually demands more processing resources (and more overhead) from the hypervisor. Xen is based on paravirtualization; it requires that the guest operating systems be modified to support the Xen operating environment. However, the user space applications and libraries do not require modification.

Operating system modifications are necessary for reasons like:

• So that Xen can replace the operating system as the most privileged software.

• So that Xen can use more efficient interfaces (such as virtual block devices and virtual network interfaces) to emulate devices — this increases performance.

Xen can run multiple guest OS each in its on VM

Xen can run several guest operating systems each running in its own virtual machine or domain. When Xen is first installed, it automatically creates the first domain, Domain 0 (or dom0).

Domain 0 is the management domain and is responsible for managing the system. It performs tasks like building additional domains (or virtual machines), managing the virtual devices for each virtual machine, suspending virtual machines, resuming virtual machines, and migrating virtual machines. Domain 0 runs a guest operating system and is responsible for the hardware devices.

Instead of a driver, lots of great stuff happens in the Xen daemon

The Xen daemon, x end, is a Python program that runs in dom0. It is the central point of control for managing virtual resources across all the virtual machines running on the Xen hypervisor. Most of the command parsing, validation, and sequencing happens in user space in x end and not in a driver.

IBM supports the SUSE Linux Enterprise Edition (SLES) 10 version of Xen which supports the following configuration:

• Four virtual machines per processor and up to 64 virtual machines per physical system.

• SLES 10 guest operating systems (para virtualized only).

DEPLOYING VIRTUALIZATION

To deploy virtualization for Xen:

• Install Xen on the system.

• Create and configure virtual machines (this includes the guest operating system).

Install the Xen software using one of the following methods:

• Interactive install: Use this procedure to install directly on dedicated virtual machine on the Xen server. This dedicated virtual machine is referred to as the client computer in the install procedure.

• Install from CommCell console: Use this procedure to install remotely on a dedicated virtual machine on the Xen server.

Managing your virtual machines

There are several virtual machine managers available including:

• Open source mangers: Open Xen Manager, an open source clone of Citrix's Xen Server Xen Centre and manages both XCP and Citrix's Xen Server. Xen Cloud Control System (XCCS) is a lightweight front end package for the excellent Xen Cloud Platform cloud computing system. Zentific , a web-based management interface for the effective control of virtual machines running upon the Xen hypervisor.

• Commercial managers: Con virture: Convert is a centralized management solution that lets you provision, monitor, and manage the complete life cycle of your Xen deployment. Citrix Xen Centre is a Windows-native graphical user interface for managing Citrix Xen Server and XCP. Versiera is a web-based Internet technology designed to securely manage and monitor both cloud environments and enterprises with support for Linux, FreeBSD, OpenBSD, NetBSD, OS X, Windows, Solaris, Open WRT, and DD-WRT.

CHOOSING XEN

On the pro side:

• The Xen server is built on the open source Xen hypervisor and uses a combination of paravirtualization and hardware-assisted virtualization. This collaboration between the OS and the virtualization platform enables the development of a simpler hypervisor that delivers highly optimized performance.

• Xen provides sophisticated workload balancing that captures CPU, memory, disk I/O, and network I/O data; it offers two optimization modes: one for performance and another for density.

• The Xen server takes advantage of a unique storage integration feature called the Citrix Storage Link. With it, the sysadmin can directly leverage features of arrays from such companies as HP, Dell Equal Logic, NetApp, EMC, and others.

• The Xen server includes multicore processor support, live migration, physical-server-to-virtual-machine conversion (P2V) and virtual-to-virtual conversion (V2V) tools, centralized multi server management, real-time performance monitoring, and speedy performance for Windows and Linux.

On the con side:

• Xen has a relatively large footprint and relies on Linux in dom0.

• Xen relies on third-party solutions for hardware device drivers, storage, backup and recovery, and fault tolerance.

• Xen gets bogged down with anything with a high I/O rate or anything that sucks up resources and starves other VMs.

• Xen's integration can be problematic; it could become a burden on your Linux kernel over time.

• Xen Server 5 is missing 802.1Q virtual local area network (VLAN) trunking; as for security, it doesn't offer directory services integration, role-based access controls, or security logging and auditing or administrative actions.

VMWARE:

VMware is a virtualization and cloud computing software provider based in Palo Alto, Calif. Founded in 1998, VMware is a subsidiary of Dell Technologies. EMC Corporation originally acquired VMware in 2004; EMC was later acquired by Dell Technologies in 2016. VMware bases its virtualization technologies on its bare-metal hypervisor ESX/ESXI in x86 architecture.

With VMware server virtualization, a hypervisor is installed on the physical server to allow for multiple virtual machines (VMs) to run on the same physical server. Each VM can run its own operating system (OS), which means multiple OSes can run on one physical server. All the VMs on the same physical server share resources, such as networking and RAM. In 2019, VMware added support to its hypervisor to run containerized workloads in a Kubernetes cluster in a similar way. These types of workloads can be managed by the infrastructure team in the same way as virtual machines and the DevOps teams can deploy containers as they were used to.

Diane Greene, Scott Devine, Mendel Rosenblum, Edward Wang and Edouard Bug neon founded VMware, which launched its first product -- VMware Workstation -- in 1999. The company released its second product, VMware ESX in 2001.

VMware's current CEO is Patrick Gel singer, appointed in 2012.

VMWARE PRODUCTS

VMware products include virtualization, networking and security management tools, software-defined data centre software and storage software.

Data centre and cloud infrastructure

VMware vSphere is VMware's suite of virtualization products. VMware vSphere, known as VMware Infrastructure prior to 2009, includes the following:

• ESXI

• vCenter Server

• vSphere Client

• v Motion

As of April 2018, the most current version is vSphere 6.7, which is available in three editions: Standard, Enterprise Plus and Platinum. There are also two three-server kits targeted toward small and medium-sized businesses named vSphere Essentials and Essentials Plus.

With VMware Cloud on AWS, customers can run a cluster of vSphere hosts with v SAN and NSX in an Amazon data centre and run their workloads there while in the meantime manage them with their well-known VMware tools and skills.

NETWORKING AND SECURITY

VMware NSX is a virtual networking and security software offering created when VMware acquired Nicera in 2012. NSX allows an admin to virtualize network components, enabling them to develop, deploy and configure virtual networks and switches through software rather than hardware. A software layer sits on top of the hypervisor to allow an administrator to divide a physical network into multiple virtual networks.

With the latest release of the product, NSX-T Data Centre, network virtualization can be added to both ESXI and KVM as hypervisors, as well as to bare-metal servers. Also containerized workloads in a Kubernetes cluster can be virtualized and protected. NSX-T Data Centre also offers Network Function Virtualization, with which functions such as a firewall, load balancer and VPN, can be run in the virtualization software stack.

VMware v Realize Network Insight is a network operations management tool that enables an admin to plan micro segmentation and check on the health of VMware NSX. V Realize Network Insight relies on technology from VMware's acquisition of Arkin in 2016. V Realize Network Insight collects information from the NSX Manager. It also displays errors in its user interface, which helps troubleshoot an NSX environment.