Experiment-6

Program Statement:

Develop a C program to simulate the following contiguous memory allocation Techniques:

a) Best fit b) First fit c) worst fit

a) Best fit

ALGORITHM:

Step 1- Input memory blocks and processes with sizes.

Step 2- Initialize all memory blocks as free.

Step 3- Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current],

Step 4: if found then assign it to the current process.

Step 5- If not then leave that process and keep checking the further processes.

EXPECTED OUTPUT:

<<leave space>>

b) First fit:

ALGORITHM:

Step 1: Input memory blocks with size and processes with size.

Step 2: Initialize all memory blocks as free.

Step 3: Start by picking each process and check if it can be assigned to current block.

Step 4: If size-of-process <= size-of-block if yes then assign and check for next process.

Step 5: If not then keep checking the further blocks.

EXPECTED OUTPUT:

<<Leave Space>>

c) Worst Fit:

ALGORITHM:

Step 1: Input memory blocks and processes with sizes.

Step 2: Initialize all memory blocks as free.

Step 3: Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current],

Step 4: if found then assign it to the current process.

Step 5: If not then leave that process and keep checking the further processes.

EXPECTED OUTPUT:

<<LEAVE SPACE>>

EXPERIMENT-7

Develop a C Program to simulate page replacement algorithms:

a) FIFO

b) LRU

a) FIFO

ALGORITHM:

Step-1. Start the process

Step-2. Read number of pages n

Step-3. Read page numbers into an array a[i]

Step-4. Initialize avail[i]=0 .to check page hit

Step-5. Replace the page with circular queue, while re-placing check page availability in the frame
Place avail[i]=1 if page is placed in the frame Count page faults

Step-6. Print the results.

Step-7. Stop the process.

Expected Output:

<<Leave Space>>

b) LRU

ALGORITHM:

Step-1. Start the process

Step-2. Declare the size

Step-3. Get the number of pages to be inserted

Step-4. Get the value

Step-5. Declare counter and stack

Step-6. Select the least recently used page by counter value

Step-7. Stack them according the selection.

Step-8. Display the values

Step-9. Stop the process

Expected Output:

<<Leave Space>>

EXPERIMENT-8

Develop a C program simulate the following File Organization Techniques:

a) Single level directory

b) Two level directory

a) Single-level directory

ALGORITHM:

Step 1: Input number of directories

Display: "Enter number of directories:"

Read value into master.

Step 2: Input directory details (name and files)

Repeat for each directory i from 0 to master - 1:

Display: "Directory (i+1):"

Display: "Enter directory name:"

Read a string into directories[i].name

Display: "Enter number of files:"

Read an integer into directories[i].fileCount

Display: "Enter file names:"

Repeat for each file j from 0 to directories[i].fileCount - 1:

Display: "File (j+1):"

Read a string into directories[i].files[j]

Step 3: Display header

Print a separator line (e.g., "==========")

Print column headings: "Directory Size Filenames"

Print another separator line.

Step 4: Display directory structure

Repeat for each directory i from 0 to master - 1:

Print:

directories[i].name

directories[i].fileCount

First file name directories[i].files[0] on the same line

For remaining files j from 1 to directories[i].fileCount - 1:

Print the file name directories[i].files[j] on a new line, with indentation under the “Filenames” column.

Print a blank line after each directory for readability.

Step 5: End

Print final separator line (optional).

Terminate the program.

Expected Output:

<<Leave Space>>

b) Two-level Directory file organization techniques

ALGORITHM:

Step-1: Start

Step-2: Declare data structures

I. Define a structure st with:

- dname → directory name (string)
- sdname[10] → array of subdirectory names (strings)
- fname[10][10] → 2D array of file names (files inside each subdirectory)
- ds → number of subdirectories in this directory (integer)
- sds[10] → array storing number of files in each subdirectory

II. Declare an array dir[10] of type struct st to hold up to 10 directories.

III. Declare integers n, i, j, k.

Step-3: Input total number of directories

I. Print: "enter number of directories:"

II. Read n.

Step-4: For each directory (loop i from 0 to n-1)

I. Print: "enter directory i+1 names:"

II. Read dir[i].dname.

III. Print: "enter size of directories:"

- Here, “size” means number of subdirectories in this directory.

IV. Read dir[i].ds.

V. For each subdirectory in this directory (loop j from 0 to dir[i].ds - 1)

- Print: "enter subdirectory name and size:"
- Read subdirectory name into dir[i].sdname[j].
- Read number of files in this subdirectory into dir[i].sds[j].
- For each file in this subdirectory (loop k from 0 to dir[i].sds[j] - 1)
  - Print: "enter file name:"
  - Read file name into dir[i].fname[j][k].

Step-5: Display header

I. Print: "dirname\t\tsize\tsubdirname\tsize\tfiles"

II. Print a separator line.

Step-6: Display stored directory structure

I. For each directory (loop i from 0 to n-1):

- Print directory name dir[i].dname and number of subdirectories dir[i].ds.
- For each subdirectory in that directory (loop j from 0 to dir[i].ds - 1):
  - Print subdirectory name dir[i].sdname[j] and number of files dir[i].sds[j].
  - For each file in that subdirectory (loop k from 0 to dir[i].sds[j] - 1):
    - Print file name dir[i].fname[j][k].
  - Move to the next line with some indentation.
- Print a newline after finishing all subdirectories of this directory.

End

EXPECTED OUTPUT:

<<EXPECTED OUTPUT>>

EXPERIMENT-9

Develop a C program to simulate the Linked file allocation strategies.

ALGORITHM:

Step-1: Start the program.

Step-2: Get the number of files.

Step-3: Get the memory requirement of each file.

Step-4: Allocate the required locations by selecting a location randomly q= random(100);

a) Check whether the selected location is free .

b) If the location is free allocate and set flag=1 to the allocated locations. While allocating next location address to attach it to previous location

for(i=0;i0) { } } p=r[i][j-1]; b[p].next=q;

}

Step-5: Print the results file no, length, Blocks allocated.

Step-6: Stop the program

EXPECTED OUTPUT:

<<Leave space>>

EXPERIMENT-10

Develop a C program to simulate SCAN disk scheduling algorithm.

ALGORITHM:

Step 1 − The array has the elements that have requested resources from different processes in ascending order based on arrival time.

Step 2 − The disk arm starts from one end of the disk and moves towards the other end.

Step 3 − When the disk moves in this direction, it will service all tracks one by one based on the request.

Step 4 − Then calculate the total distance traveled from the head.

Step 5 − The new position is identified by the currently serviced request

Step 6 − Then step 3 is repeated when it reaches one end of the disk.

Step 7 − When the endpoint is reached, it will reverse the direction and go back to step 2 when the entire request has been serviced.

EXPECTED OUTPUT:

<<EXPECTED OUTPUT>>