EXPERIMENT-6

Experiment 6: Contiguous Memory Allocation Techniques

Program Statement:

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

1. Best Fit
2. First Fit
3. Worst Fit

a) Best Fit

Code;

```
#include <stdio.h> #define max 25 int main() { int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000; static int bf[max], ff[max]; printf("\nEnter the number of blocks:"); scanf("%d", &nb); printf("Enter the number of files:"); scanf("%d", &nf); printf("\nEnter the size of the blocks:-\n"); for (i = 1; i <= nb; i++) { printf("Block %d:", i); scanf("%d", &b[i]); } printf("Enter the size of the files :-\n"); for (i = 1; i <= nf; i++) { printf("File %d:", i); scanf("%d", &f[i]); } for (i = 1; i <= nf; i++) { lowest = 10000; ff[i] = 0; // Initialize to 0 (no block allocated) for (j = 1; j <= nb; j++) { if (bf[j] != 1) // If block is not allocated { temp = b[j] - f[i]; if (temp >= 0) // If file fits in block { if (lowest > temp) // If this is the best fit so far { ff[i] = j; lowest = temp; } } } } // Only allocate if a suitable block was found if (ff[i] != 0) { frag[i] = lowest; // Store fragment bf[ff[i]] = 1; // Mark block as allocated } else { frag[i] = -1; // Indicate allocation failure } } printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment"); for (i = 1; i <= nf; i++) { if (ff[i] != 0) printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]); else printf("\n%d\t\t%d\t\tNot Allocated", i, f[i]); } printf("\n"); return 0; }
```

Expected output:

b) First fit:

Code:

```
#include <stdio.h> #define MAX 25 // Function to allocate memory to blocks as per First fit algorithm void firstFit(int blockSize[], int m, int processSize[], int n) { int i, j; // Stores block id of the block allocated to a process int allocation[n]; // Create a copy of blockSize to preserve original values int availableBlock[m]; for (i = 0; i < m; i++) { availableBlock[i] = blockSize[i]; } // Initially no block is assigned to any process for (i = 0; i < n; i++) { allocation[i] = -1; } // Pick each process and find suitable blocks // according to its size and assign to it for (i = 0; i < n; i++) // n -> number of processes { for (j = 0; j < m; j++) // m -> number of blocks { if (availableBlock[j] >= processSize[i]) { // Allocating block j to the ith process allocation[i] = j; // Reduce available memory in this block availableBlock[j] -= processSize[i]; break; // Go to the next process in the queue } } } printf("\nProcess No.\tProcess Size\tBlock no.\n"); for (i = 0; i < n; i++) { printf(" %d\t\t%d KB\t\t", i + 1, processSize[i]); if (allocation[i] != -1) printf("%d\n", allocation[i] + 1); else printf("Not Allocated\n"); } } // Driver code int main() { int nb, np; int blockSize[MAX]; int processSize[MAX]; int m, n; printf("Enter number of Processes: "); scanf("%d", &np); printf("Enter number of Blocks: "); scanf("%d", &nb); for(int i = 0; i < np; i++){ printf("Enter size of Process %d: ", i + 1); scanf("%d",
```

&processSize[i]); } for(int i = 0; i < nb; i++){ printf("Enter size of Block %d: ", i + 1); scanf("%d", &blockSize[i]); } m = nb; n = np; firstFit(blockSize, m, processSize, n); return 0; }

Expected output:

c) Worst fit:

Code:
#include <stdio.h> #define MAX 25 int main(void) { int frag[MAX], b[MAX], f[MAX]; int i, j, nb, nf, temp, highest; static int bf[MAX], ff[MAX]; printf("\n\tMemory Management Scheme - Worst Fit"); printf("\nEnter the number of blocks: "); scanf("%d", &nb); printf("Enter the number of files: "); scanf("%d", &nf); // Optional: basic bounds check if (nb > MAX || nf > MAX) { printf("Error: nb and nf must each be <= %d\n", MAX); return 1; } printf("\nEnter the size of the blocks:\n"); for (i = 0; i < nb; i++) { printf("Block %d: ", i + 1); scanf("%d", &b[i]); } printf("Enter the size of the files:\n"); for (i = 0; i < nf; i++) { printf("File %d: ", i + 1); scanf("%d", &f[i]); } // Initialize bf and ff arrays for (i = 0; i < nb; i++) bf[i] = 0; // 0 = free, 1 = allocated for (i = 0; i < nf; i++) ff[i] = -1; // -1 = no block assigned // Worst Fit allocation for (i = 0; i < nf; i++) { int worst_fit_block_idx; // ✅ Declare at start of block highest = -1; // Reset highest for each file worst_fit_block_idx = -1; // No block chosen yet for (j = 0; j < nb; j++) { if (bf[j] != 1) { // Block is free temp = b[j] - f[i]; if (temp >= 0) { // Block can accommodate file if (temp > highest) { highest = temp; worst_fit_block_idx = j; } } } } if (worst_fit_block_idx != -1) { // A suitable block was found ff[i] = worst_fit_block_idx; // Assign block index to file frag[i] = highest; // Fragmentation bf[worst_fit_block_idx] = 1; // Mark block as allocated } else { frag[i] = -1; // File could not be allocated } } printf("\nFile_no:\tFile_size:\tBlock_no: \tBlock_size:\tFragment"); for (i = 0; i < nf; i++) { if (ff[i] != -1)
{ printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]); } else
{ printf("\n%d\t\t%d\t\tNot Allocated\t--\t\t--", i + 1, f[i]); } } printf("\n"); return 0; }

Expected output:

EXPERIMENT-7

Develop a C program to simulate page replacement algorithms:

a) FIFO

b) LRU

a)FIFO

#include<stdio.h> int fr[3]; void display() { int i; printf("\n"); for(i=0;i<3;i++) printf("%d\t",fr[i]); } int main() { int i,j,n,page[50]; int flag1=0,flag2=0,pf=0,frsize=3,top=0; printf("Enter number of pages (max 50): "); scanf("%d",&n); printf("Enter %d page numbers: ",n); for(i=0;i<n;i++) scanf("%d",&page[i]); for(i=0;i<3;i++) { fr[i]=-1; } for(j=0;j<n;j++) { flag1=0; flag2=0; for(i=0;i<3;i+

+) { if(fr[i]==page[j]) { flag1=1; flag2=1; break; } } if(flag1==0) { for(i=0;i<frsize;i++) { if(fr[i]==-1) { fr[i]=page[j]; flag2=1; pf++; // ← minimal bug fix break; } } } if(flag2==0) { fr[top]=page[j]; top++; pf++; if(top>=frsize) top=0; } display(); } printf("\nNumber of page faults : %d ",pf); return 0; }

Expected Output:

b) LRU

Code:

#include <stdio.h> #define FRAMES 3 #define PAGES 12 int fr[FRAMES]; void display() { int i; printf("\n"); for (i = 0; i < FRAMES; i++) printf("%d\t", fr[i]); } int main() { int page[PAGES] = {2,3,2,1,5,2,4,5,3,2,5,2}; int last_used[FRAMES]; // store "time" when each frame was last used int time = 0; int pf = 0; int i, j; // initialize frames as empty for (i = 0; i < FRAMES; i++) { fr[i] = -1; last_used[i] = -1; } for (j = 0; j < PAGES; j++) { int current = page[j]; int hit = 0; time++; // 1. Check if page is already in a frame (HIT) for (i = 0; i < FRAMES; i++) { if (fr[i] == current) { hit = 1; last_used[i] = time; // update last used time break; } } if (!hit) { // 2. Check for an empty frame first int placed = 0; for (i = 0; i < FRAMES; i++) { if (fr[i] == -1) { fr[i] = current; last_used[i] = time; pf++; placed = 1; break; } } // 3. If no empty frame, replace LRU page if (!placed) { int lru_index = 0; int min_time = last_used[0]; for (i = 1; i < FRAMES; i++) { if (last_used[i] < min_time) { min_time = last_used[i]; lru_index = i; } } fr[lru_index] = current; last_used[lru_index] = time; pf++; } } display(); } printf("\nNumber of page faults (LRU): %d\n", pf); return 0; }

Expected Output:

EXPERIMENT-8

Simulate following File Organization Techniques

A) Single level directory.

B) Two level directory.

A) Single level directory

Code;

```c
#include<stdio.h>

// Structure to represent a directory
struct Directory {
  char name[20];
  int fileCount;
  char files[20][20];
};

int main()
{
  struct Directory directories[20];
  int master;
  int i, j;

  printf("Enter number of directories: ");
  scanf("%d", &master);

  // Input directory information
  for(i = 0; i < master; i++) {
    printf("\nDirectory %d:\n", i + 1);

    printf("  Enter directory name: ");
    scanf("%s", directories[i].name);

    printf("  Enter number of files: ");
    scanf("%d", &directories[i].fileCount);

    printf("  Enter file names:\n");
    for(j = 0; j < directories[i].fileCount; j++) {
      printf("    File %d: ", j + 1);
      scanf("%s", directories[i].files[j]);
    }
  }

  // Display directory structure
  printf("\n\n");
  printf("=================================================\n");
  printf(" Directory\t\tSize\tFilenames\n");
  printf("=================================================\n");

  for(i = 0; i < master; i++)
  {
    printf("%-20s\t%2d\t", directories[i].name, directories[i].fileCount);

    for(j = 0; j < directories[i].fileCount; j++) {
      if(j == 0)
```

```
            printf("%s\n", directories[i].files[j]);
        else
            printf("\t\t\t\t%s\n", directories[i].files[j]);
    }
    printf("\n");
  }
  printf("================================================\n");

  return 0;
}
```

Expected output:

b) Two level directory

```
#include<stdio.h>
#include<conio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
void main()
{
int i,j,k,n;
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
```

```
}}}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*************************************************\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");
}
}
```

Expected Output;

EXPERIMENT-9

Develop a C Program to simulate the Linked file allocation strategies

Code;

#include<stdio.h> #include<stdlib.h> // Added for exit() function void main() { int f[50],p,i,j,k,a,st,len,n,c; for(i=0;i<50;i++) f[i]=0; printf("Enter how many blocks that are already allocated: "); scanf("%d",&p); printf("\nEnter the blocks no.s that are already allocated: "); for(i=0;i<p;i++) { scanf("%d",&a); f[a]=1; } X: printf("\nEnter the starting index block & length: "); scanf("%d%d",&st,&len); k=len; for(j=st;j<(k+st);j++) { if(f[j]==0) { f[j]=1; printf("\n%d->%d",j,f[j]); } else { printf("\n%d->file is already allocated",j); k++; } } printf("\nIf u want to enter one more file? (yes-1/no-0): "); scanf("%d",&c); if(c==1) goto X; else exit(0); // Changed from exit() to exit(0) }

Expected output:

EXPERIMENT-10

Develop a C program to simulate SCAN disk scheduling algorithm.

Code;

```
#include <stdio.h> #include <stdlib.h> #include <string.h> #define size 10 #define disk_size 200 int comp(const void * l, const void * n) { return (*(int*)l - *(int*)n); } void SCAN(int arr[], int head, char* dn){ int seek_num = 0; int dt, cur_track; int leftside[size + 1] = {0}; // Initialize array int rightside[size + 1] = {0}; // Initialize array int seek_seq[size + 2]; // Correct size int m_scan = 0, s_scan = 0; if (strcmp(dn, "leftside") == 0) leftside[m_scan++] = 0; else if (strcmp(dn, "rightside") == 0) rightside[s_scan++] = disk_size - 1; for (int p_s = 0; p_s < size; p_s++) { if (arr[p_s] < head) leftside[m_scan++] = arr[p_s]; if (arr[p_s] > head) rightside[s_scan++] = arr[p_s]; } qsort(leftside, m_scan, sizeof(int), comp); qsort(rightside, s_scan, sizeof(int), comp); int go = 2; int ind = 0; while (go--) { if (strcmp(dn, "leftside") == 0) { for (int p_s = m_scan - 1; p_s >= 0; p_s--) { cur_track = leftside[p_s]; seek_seq[ind++] = cur_track; dt = abs(cur_track - head); seek_num += dt; head = cur_track; } dn = "rightside"; } else if (strcmp(dn, "rightside") == 0) { for (int p_s = 0; p_s < s_scan; p_s++) { cur_track = rightside[p_s]; seek_seq[ind++] = cur_track; dt = abs(cur_track - head); seek_num += dt; head = cur_track; } dn = "leftside"; } } printf("Num of seek process = %d\n", seek_num); printf("Sequence is:\n"); for (int p_s = 0; p_s < ind; p_s++) { printf("%d\n", seek_seq[p_s]); } } int main(){ int arr[size] = { 126, 90, 14, 50, 25, 42, 51, 78, 102, 100 }; int head = 42; char dn[] = "leftside"; SCAN(arr, head, dn); return 0; }
```

Expected output: