**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**Semester - IV**

**UCS1411 – Operating Syatems Lab**

**Academic Year: 2019-2020 Batch: 2018-2022**

**Name : Rahul Ram M**

**Class : CSE – B**

**Register Number : 185001121**

**Lab Exercise 12: File Allocation Techniques**

**CODE:**

**LINKEDLIST.h**

**typedef Block Data;**

**typedef struct Node**

**{**

**Data d;**

**struct Node \*next;**

**} Node;**

**typedef Node \*List;**

**extern void init\_block(Block \*const);**

**List createEmptyList()**

**{**

**Node \*head = (Node \*)malloc(sizeof(Node));**

**init\_block(&(head -> d));**

**head->next = NULL;**

**return head;**

**}**

**void insertLast(List head, const Data d)**

**{**

**Node \*new = (Node \*)malloc(sizeof(Node));**

**new->d = d;**

**Node \*tmp = head;**

**while (tmp->next)**

**tmp = tmp->next;**

**new->next = NULL;**

**tmp->next = new;**

**}**

**void insertFirst(List head, const Data d)**

**{**

**Node \*new = (Node \*)malloc(sizeof(Node));**

**new->d = d;**

**new->next = head->next;**

**head->next = new;**

**}**

**Data delete (List prev)**

**{**

**Data rVal;**

**if (!prev)**

**return rVal;**

**if (!prev->next)**

**return rVal;**

**Node \*tmp = prev->next;**

**rVal = tmp->d;**

**prev->next = prev->next->next;**

**free(tmp);**

**return rVal;**

**}**

**Data deleteFirst(List head)**

**{**

**Data rVal;**

**if (head->next == NULL)**

**{**

**printf(" Empty List!\n");**

**return rVal;**

**}**

**delete (head);**

**}**

**Data deleteLast(List head)**

**{**

**Data rVal;**

**if (head->next == NULL)**

**{**

**printf(" Empty List!\n");**

**return rVal;**

**}**

**Node \*tmp = head;**

**while (tmp->next->next != NULL)**

**tmp = tmp->next;**

**delete (tmp);**

**}**

**void display(List head)**

**{**

**Node \*tmp = head->next;**

**if (tmp == NULL)**

**{**

**printf(" Empty!\n");**

**return;**

**}**

**while (tmp)**

**{**

**printf(" BID: %-2d\tStatus: %d\n", tmp->d.id, tmp->d.status);**

**tmp = tmp->next;**

**}**

**}**

**int length(List head)**

**{**

**Node \*tmp = head->next;**

**if (tmp == NULL)**

**return 0;**

**int count = 0;**

**while (tmp)**

**{**

**tmp = tmp->next;**

**count++;**

**}**

**return count;**

**}**

**Node\* search(List head, const int id)**

**{**

**if (head->next == NULL)**

**return NULL;**

**Node \*tmp = head -> next;**

**while (tmp)**

**{**

**if (tmp->d.id == id)**

**return tmp;**

**tmp = tmp->next;**

**}**

**return NULL;**

**}**

**FIEALLOCATION.C**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define MAX 100

#define FREE 0

typedef struct File

{

char name[21];

int size;

int start\_block;

int end\_block;

int \*indices;

int length;

} File;

void init\_file(File \*const);

typedef struct Directory

{

File f[MAX];

int size;

} Directory;

void init\_dir(Directory \*const);

typedef struct Block

{

int id;

unsigned status : 1;

struct Block \*next\_file\_blk;

} Block;

void init\_block(Block \*const);

#include "LinkedList.h"

void contiguous(File \*const, const int, const int, const int);

void linked(File \*const, const int, const int, const int);

void indexed(File \*const, const int, const int, const int);

int main()

{

int mem\_size;

int blk\_size;

int num\_blks;

int num\_file;

int choice;

File f[MAX];

printf(" Enter the size of memory: ");

scanf("%d", &mem\_size);

printf(" Enter the size of block: ");

scanf("%d", &blk\_size);

num\_blks = mem\_size / blk\_size;

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

scanf("%d", &num\_file);

getchar();

for (int i = 0; i < num\_file; i++)

{

printf(" Enter the name of file: ");

scanf("%[^\n]", f[i].name);

printf(" Enter the size of file: ");

scanf("%d", &f[i].size);

getchar();

}

while (1)

{

printf("\t\t\tFILE ALLOCATION TECHNIQUES\n");

printf(" 1 - Contiguous\n");

printf(" 2 - Linked\n");

printf(" 3 - Indexed\n");

printf(" 0 - Exit\n");

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

printf(" Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 0:

exit(0);

case 1:

contiguous(f, num\_file, blk\_size, num\_blks);

break;

case 2:

linked(f, num\_file, blk\_size, num\_blks);

break;

case 3:

indexed(f, num\_file, blk\_size, num\_blks);

break;

default:

printf(" Invalid Input!\n");

}

}

}

void init\_file(File \*const f)

{

strcpy(f->name, "");

f->start\_block = -1;

f->end\_block = -1;

f->size = -1;

f->indices = NULL;

f->length = -1;

}

void init\_dir(Directory \*const d)

{

d->size = 0;

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

init\_file(&(d->f[i]));

}

void init\_block(Block \*const b)

{

b->status = FREE;

b->id = -1;

b->next\_file\_blk = NULL;

}

void contiguous(File \*const f, const int n\_files, const int blk\_size, const int num\_blk)

{

List list = createEmptyList();

Block b;

init\_block(&b);

Node \*ptr, \*tmp;

int blocks\_visited, flag, id, counter, blk\_req;

int start, end;

for (int i = 0; i < num\_blk; i++)

{

b.id = i;

insertLast(list, b);

}

for (int i = 0; i < n\_files; i++)

{

blocks\_visited = 0;

flag = 0;

blk\_req = f[i].size / blk\_size;

if (f[i].size % blk\_size)

blk\_req++;

while (blocks\_visited < num\_blk && !flag)

{

id = random() % num\_blk;

ptr = search(list, id);

if (ptr->d.status != FREE)

{

blocks\_visited++;

continue;

}

counter = 0;

start = ptr->d.id;

tmp = ptr;

while (tmp)

{

if (tmp->d.status == FREE)

{

counter++;

if (counter == blk\_req)

{

flag = 1;

break;

}

}

else

break;

tmp = tmp->next;

}

if (flag)

{

f[i].start\_block = start;

f[i].length = blk\_req;

tmp = ptr;

for (int i = 0; i < blk\_req; i++)

{

tmp->d.status = 1;

tmp = tmp->next;

}

}

else

blocks\_visited++;

}

if (!flag)

printf(" Unable to allocate file: %s\n!", f[i].name);

}

printf("\n\t\tDIRECTORY STRUCTURE\n");

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

printf(" | File Name | Start | Length |\n");

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

for (int i = 0; i < n\_files; i++)

if (f[i].length > 0)

printf(" | %-20s | %-5d | %-6d |\n", f[i].name, f[i].start\_block, f[i].length);

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

}

void linked(File \*const f, const int n\_files, const int blk\_size, const int num\_blk)

{

List list = createEmptyList();

Block b;

init\_block(&b);

Node \*ptr, \*tmp, \*left, \*right;

int blocks\_visited, flag, id, counter, blk\_req;

for (int i = 0; i < num\_blk; i++)

{

b.id = i;

insertLast(list, b);

}

for (int i = 0; i < n\_files; i++)

{

counter = 0;

blocks\_visited = 0;

flag = 0;

blk\_req = f[i].size / blk\_size;

if (f[i].size % blk\_size)

blk\_req++;

int \*allocated = (int \*)calloc(blk\_req, sizeof(int));

while (blocks\_visited < num\_blk && !flag)

{

id = random() % num\_blk;

ptr = search(list, id);

if (ptr->d.status != FREE)

{

blocks\_visited++;

continue;

}

ptr -> d.status = 1;

allocated[counter++] = id;

if (counter == blk\_req)

flag = 1;

}

if (!flag){

printf(" Unable to allocate file: %s\n", f[i].name);

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

ptr = search(list, allocated[i]);

ptr -> d.status = FREE;

}

free(allocated);

}

else

{

f[i].start\_block = allocated[0];

f[i].end\_block = allocated[blk\_req - 1];

f[i].length = blk\_req;

for (int i = 0; i < blk\_req - 1; i++)

{

left = search(list, allocated[i]);

right = search(list, allocated[i + 1]);

left->d.next\_file\_blk = &(right->d);

left->d.status = 1;

}

right->d.next\_file\_blk = NULL;

free(allocated);

}

}

printf("\n\t\tDIRECTORY STRUCTURE\n");

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

printf(" | File Name | Start Block | End Block |\n");

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

for (int i = 0; i < n\_files; i++)

if (f[i].end\_block >= 0)

printf(" | %-20s | %-2d | %-2d |\n",

f[i].name, f[i].start\_block, f[i].end\_block);

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

printf("\n");

for (int i = 0; i < n\_files; i++)

if (f[i].start\_block >= 0)

{

printf("\n\n File Name: %s\n ",f[i].name);

ptr = search(list, f[i].start\_block);

Block \*b = &(ptr->d);

while (b)

{

printf("%-2d ", b->id);

b = b->next\_file\_blk;

}

}

}

void indexed(File \*const f, const int n\_files, const int blk\_size, const int num\_blk)

{

List list = createEmptyList();

Block b;

init\_block(&b);

Node \*ptr, \*tmp;

int blocks\_visited, flag, id, counter, blk\_req;

int start, end;

for (int i = 0; i < num\_blk; i++)

{

b.id = i;

insertLast(list, b);

}

for (int i = 0; i < n\_files; i++)

{

blocks\_visited = 0;

flag = 0;

blk\_req = f[i].size / blk\_size;

if (f[i].size % blk\_size)

blk\_req++;

f[i].indices = (int \*)calloc(blk\_req + 1, sizeof(int));

f[i].length = blk\_req;

counter = 0;

while (blocks\_visited < num\_blk && !flag)

{

id = random() % num\_blk;

ptr = search(list, id);

if (ptr->d.status == FREE)

{

f[i].indices[counter++] = id;

if (counter == blk\_req + 1)

{

flag = 1;

break;

}

}

else

blocks\_visited++;

}

if (!flag)

{

printf(" Unable to allocate memory for file: %s\n", f[i].name);

free(f[i].indices);

f[i].indices = NULL;

}

}

printf("\n\t\tDIRECTORY STRUCTURE\n");

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

printf(" | File Name | Index Block |\n");

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

for (int i = 0; i < n\_files; i++)

if (f[i].indices)

printf(" | %-20s | %-2d |\n", f[i].name, f[i].indices[0]);

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

printf("\n\n");

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

printf(" | File Name | Blocks Indexed |\n");

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

for (int i = 0; i < n\_files; i++)

{

if (f[i].indices)

{

for (int j = 1; j <= f[i].length; j++)

printf(" | %-20s | %-2d |\n", ((j > 1) ? "" : f[i].name), f[i].indices[j]);

}

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

}

}

OUTPUT:

Enter the size of memory: 500

Enter the size of block: 10

Enter the number of files: 3

Enter the name of file: file1.txt

Enter the size of file: 54

Enter the name of file: temp.bin

Enter the size of file: 102

Enter the name of file: output.pdf

Enter the size of file: 33

FILE ALLOCATION TECHNIQUES

1 - Contiguous

2 - Linked

3 - Indexed

0 - Exit

----------------------

Enter your choice: 1

DIRECTORY STRUCTURE

+----------------------+-------+--------+

| File Name | Start | Length |

+----------------------+-------+--------+

| file1.txt | 33 | 6 |

| temp.bin | 15 | 11 |

| output.pdf | 43 | 4 |

+----------------------+-------+--------+

FILE ALLOCATION TECHNIQUES

1 - Contiguous

2 - Linked

3 - Indexed

0 - Exit

----------------------

Enter your choice: 2

DIRECTORY STRUCTURE

+----------------------+-------------+-----------+

| File Name | Start Block | End Block |

+----------------------+-------------+-----------+

| file1.txt | 35 | 12 |

| temp.bin | 27 | 32 |

| output.pdf | 30 | 8 |

+----------------------+-------------+-----------+

File Name: file1.txt

35 36 42 49 21 12

File Name: temp.bin

27 40 9 13 26 22 11 18 17 29 32

File Name: output.pdf

30 23 2 8 FILE ALLOCATION TECHNIQUES

1 - Contiguous

2 - Linked

3 - Indexed

0 - Exit

----------------------

Enter your choice: 3

DIRECTORY STRUCTURE

+----------------------+-------------+

| File Name | Index Block |

+----------------------+-------------+

| file1.txt | 19 |

| temp.bin | 23 |

| output.pdf | 30 |

+----------------------+-------------+

+----------------------+----------------+

| File Name | Blocks Indexed |

+----------------------+----------------+

| file1.txt | 17 |

| | 43 |

| | 6 |

| | 11 |

| | 42 |

| | 29 |

+----------------------+----------------+

| temp.bin | 21 |

| | 19 |

| | 34 |

| | 37 |

| | 48 |

| | 24 |

| | 15 |

| | 20 |

| | 13 |

| | 26 |

| | 41 |

+----------------------+----------------+

| output.pdf | 6 |

| | 23 |

| | 12 |

| | 20 |

+----------------------+----------------+

FILE ALLOCATION TECHNIQUES

1 - Contiguous

2 - Linked

3 - Indexed

0 - Exit

----------------------

Enter your choice: 0