UCS1411 Operating Systems Lab
AY: 2021-22

Name: Krithika Swaminathan
Roll No.: 205001057

Assignment 12 – File Allocation Techniques: Sequential, Linked and Indexed Allocation

Date: 30/05/2022

Aim:

To develop a C program to implement the various file allocation techniques.

Algorithm:

1. Start
2. Get Main memory size and block size as input.
3. Create a Main memory with ‘n’ number of blocks of equal size.
4. Main memory is maintained as Linked List with structure containing block id, Free/Filename, Link to next Memory block , Link to Next File block (only for Linked Allocation), File block table (integer array to hold block numbers only for Indexed Allocation)
5. Get the number of files and their size as input.
6. Calculate the no. of blocks needed for each file.
7. Select the Allocation Algorithm – For every algorithm display Directory information and File information.
8. For Contiguous Allocation - For each file do the following:
    1. Generate a random number between 1 to ‘n’
    2. Check for a continuous number of needed file free blocks starting from that random block no.
    3. If free then allot that file in those continuous blocks and update the directory structure.
    4. Else, repeat step 1.
    5. If no continuous blocks are free then ‘no enough memory error’
    6. The Directory Structure should contain Filename, Starting Block, length (no. of blocks)
9. For Linked Allocation- For each file do the following:
    1. Generate a random number between 1 to ‘n’ blocks.
    2. Check if that block is free or not.
    3. If free then allot it for file. Repeat steps 1 to 3 for the needed number of blocks for the file and create a linked list in Main memory using the field “Link to Next File block”.
    4. Update the Directory entry which contains Filename, Start block number, Ending Block Number.

5. Display the file blocks starting from start block number in Directory upto ending block number by traversing the Main memory Linked list using the field “Link to Next File block”.

10. For Indexed Allocation - For each file do the following:
    1. Generate a random number between 1 to ‘n’ blocks for index block.
    2. Check if it is free. If not, repeat index block selection.
    3. Generate needed number of free blocks in random order for the file and store those block numbers in index block as array in File block table array.
    4. Display the Directory structure which contains the filename and index block number. Display the File Details by showing the index block number’s File Block table.
11. Stop

Code:

```
//Linked list ADT for file allocation techniques
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;
}

//Program to implement file allocation techniques
#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);
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
int main()
{
   printf("\n\t\t\tFILE ALLOCATION TECHNIQUES\n");

   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("\n\nMENU:\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");
    }
}
```

Name: Krithika Swaminathan
Roll No.: 205001057

Output:

```
                        FILE ALLOCATION TECHNIQUES
 Enter the size of memory: 100
 Enter the size of block: 5
 Enter the number of files: 3
 Enter the name of file: f1
 Enter the size of file: 15
 Enter the name of file: f2
 Enter the size of file: 20
 Enter the name of file: f3
 Enter the size of file: 10


MENU:
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ---------------------
 Enter your choice: 1

                DIRECTORY STRUCTURE
 +----------------------+-------+--------+
 |      File Name       | Start | Length |
 +----------------------+-------+--------+
 | f1                   | 3     | 3      |
 | f2                   | 6     | 4      |
 | f3                   | 17    | 2      |
 +----------------------+-------+--------+


MENU:
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ---------------------
```

```
 Enter your choice: 2

                DIRECTORY STRUCTURE
 +----------------------+-------------+-----------+
 |      File Name       | Start Block | End Block |
 +----------------------+-------------+-----------+
 | f1                   |     15      |    6      |
 | f2                   |     12      |    2      |
 | f3                   |     7       |    10     |
 +----------------------+-------------+-----------+



 File Name: f1
 15 13 6

 File Name: f2
 12 9  1  2

 File Name: f3
 7  10
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
MENU:
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ----------------------
 Enter your choice: 3

                DIRECTORY STRUCTURE
 +----------------------+-------------+
 |      File Name       | Index Block |
 +----------------------+-------------+
 | f1                   |     19      |
 | f2                   |     6       |
 | f3                   |     7       |
 +----------------------+-------------+

 +----------------------+----------------+
 |      File Name       | Blocks Indexed |
 +----------------------+----------------+
 | f1                   |       3        |
 |                      |       6        |
 |                      |       0        |
 +----------------------+----------------+
 | f2                   |       12       |
 |                      |       16       |
 |                      |       11       |
 |                      |       8        |
 +----------------------+----------------+
 | f3                   |       9        |
 |                      |       2        |
 +----------------------+----------------+


MENU:
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ----------------------
 Enter your choice: 0
root@hadoop-slave-3:~/krith#
```

Name: Krithika Swaminathan
Roll No.: 205001057

Learning outcomes:

- File allocation techniques were understood and implemented.
- Files were allocated in the main memory according to the sequential, linked and indexed allocation techniques.
