CYCLE-4

Exp-1	Implement Banker's Algorithm for Dead Lock avoidance and prevention
Exp-2	Simulate the following file allocation strategies a) Sequential b) Indexed c) Linked
Exp-3	Download and install nachos operating system and experiment with it

Experiment-1:

Implement Bankers Algorithm for Dead Lock avoidance and prevention.

```
#include<stdio.h>
int main()
{
 int allocated[15][15], max[15][15], need[15][15], avail[15], tres[15], work[15],
                                                                    flag[15];
    int pno, rno, i, j, prc, count, total;
    count = 0;
    printf("\n Enter number of processes: ");
    scanf("%d", &pno);
    printf("\n Enter number of resources: ");
    scanf("%d", &rno);

    for(i = 0; i < pno; i++)
        flag[i] = 0;

    printf("\n Enter total numbers of each resource: ");
    for(i = 0; i < rno; i++)
        scanf("%d", &tres[i]);

    printf("\n Enter Max resources for each process:");
    for(i = 0; i < pno; i++)
    {
        printf("\n For process %d:", i);
        for(j = 0; j < rno; j++)
            scanf("%d", &max[i][j]);
    }

    printf("\n Enter allocated resources for each process:");
    for(i = 0; i < pno; i++)
    {
        printf("\n For process %d:", i);
        for(j = 0; j < rno; j++)
            scanf("%d", &allocated[i][j]);
    }

    printf("\n Available resources:\n");
    for(j = 0; j < rno; j++)
    {
        total = 0;
        for(i = 0; i < pno; i++)
```

```
        total += allocated[i][j];

    avail[j] = tres[j] - total;
    work[j] = avail[j];
    printf(" %d \t", work[j]);
}
do
{
    for(i = 0; i < pno; i++)
        for(j = 0; j < rno; j++)
            need[i][j] = max[i][j] - allocated[i][j];

    printf("\n Allocated matrix   Max   Need");
    for(i = 0; i < pno; i++)
    {
        printf("\n");
        for(j = 0; j < rno; j++)
            printf("%4d", allocated[i][j]);

        printf("|");
        for(j = 0; j < rno; j++)
            printf("%4d", max[i][j]);

        printf("|");
        for(j = 0; j < rno; j++)
            printf("%4d", need[i][j]);
    }

    prc = -1;
    for(i = 0; i < pno; i++)
    {
        if(flag[i] == 0)
        {
            prc = i;
            for(j = 0; j < rno; j++)
            {
                if(work[j] < need[i][j])
                {
                    prc = -1;
                    break;
                }
            }
        }
        if(prc != -1)
            break;
    }

    if(prc != -1)
    {
        printf("\n Process %d completed", prc);
```

```
            count++;
            for(j = 0; j < rno; j++)
            {
                work[j] += allocated[prc][j];
                allocated[prc][j] = 0;
                max[prc][j] = 0;
            }
            flag[prc] = 1;
        }

    } while(count != pno && prc != -1);

    printf("\n The system is %s\n", (count == pno) ? "in a safe state!" : "in an
                                                                    unsafe state!");
    return 0;
}
```

Output:

```
 Enter number of processes: 5

 Enter number of resources: 3

 Enter total numbers of each resource: 10 5 7

 Enter Max resources for each process:
 For process 0:7 5 3

 For process 1:3 2 2

 For process 2:9 0 2

 For process 3:2 2 2

 For process 4:4 3 3

 Enter allocated resources for each process:
 For process 0:0 1 0

 For process 1:3 0 2

 For process 2:3 0 2

 For process 3:2 1 1

 For process 4:0 0 2

 Available resources:
 2       3       0
```

```
Allocated matrix   Max   Need
  0   1   0|   7   5   3|   7   4   3
  3   0   2|   3   2   2|   0   2   0
  3   0   2|   9   0   2|   6   0   0
  2   1   1|   2   2   2|   0   1   1
  0   0   2|   4   3   3|   4   3   1
Process 1 completed
Allocated matrix   Max   Need
  0   1   0|   7   5   3|   7   4   3
  0   0   0|   0   0   0|   0   0   0
  3   0   2|   9   0   2|   6   0   0
  2   1   1|   2   2   2|   0   1   1
  0   0   2|   4   3   3|   4   3   1
Process 3 completed
Allocated matrix   Max   Need
  0   1   0|   7   5   3|   7   4   3
  0   0   0|   0   0   0|   0   0   0
  3   0   2|   9   0   2|   6   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   2|   4   3   3|   4   3   1
Process 0 completed
Allocated matrix   Max   Need
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  3   0   2|   9   0   2|   6   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   2|   4   3   3|   4   3   1
Process 2 completed
Allocated matrix   Max   Need
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   2|   4   3   3|   4   3   1
Process 4 completed
The system is in a safe state!
```

Experiment-2

Simulate the following file allocation strategies

a) Sequential b) Indexed c) Linked

a) Sequential File Allocation Method

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h> // Use standard boolean types

bool allocateSequentially(int startingPosition, int allocationLength, int maximumSize, int
block[])
{
    /*---------------------------------------------------------------------------------------- */
    // Bound checking
    if (startingPosition + allocationLength > maximumSize)
    {
        return false;
    }

    /*---------------------------------------------------------------------------------------- */
    // Check whether already occupied
    for (int i = startingPosition - 1; i < startingPosition + allocationLength - 1; i++)
    {
        if (block[i] == true)
        {
            return false;
        }
    }

    /*---------------------------------------------------------------------------------------- */
    // Allocate the blocks
    for (int i = startingPosition - 1; i < startingPosition + allocationLength - 1; i++)
    {
        block[i] = true;
    }
    return true;
}

int main()
{
    /*---------------------------------------------------------------------------------------- */
    // Input and initialization
    int allocationLength, startingPosition, maximumSize;
    int inputChoice;

    printf("\n\n\t\t Sequential File Allocation Method Demonstration\n\t\t ---------- ----
---------- ------");
    printf("\n\n\t Enter the maximum size available : ");
    scanf("%d", &maximumSize);

    // Dynamic memory allocation
    int *block = (int *)malloc(maximumSize * sizeof(int));
    if (block == NULL)
    {
        printf("Memory allocation failed!\n");
        return 1;
```

```
    }

    // Initialize block array to false
    for (int i = 0; i < maximumSize; i++)
    {
        block[i] = false;
    }

    /*---------------------------------------------------------------------------------------- */
    // Execute till user wants to exit
    while (1)
    {
        printf("\n\n\t\t MENU : \n\n\t\t 1. Allocate \n\n\t\t 2. Exit \n\n\t\t Choice : ");
        scanf("%d", &inputChoice);

        if (inputChoice == 2)
        {
            free(block); // Free allocated memory before exiting
            return 0;
        }

        printf("\n\n\t Enter the starting address : ");
        scanf("%d", &startingPosition);
        printf("\n\n\t Enter the length of the block : ");
        scanf("%d", &allocationLength);

        if (allocateSequentially(startingPosition, allocationLength, maximumSize, block))
        {
            printf("\n\n\t Successfully Allocated!");
        }
        else
        {
            printf("\n\n\t Space cannot be allocated!");
        }
    }

    free(block); // Free memory before program ends
    return 0;
}
```

Output:

```
Sequential File Allocation Method Demonstration
          ---------- ---- ---------- ------
     Enter the maximum size available: 10
          MENU :
          1. Allocate
          2. Exit
          Choice : 1
     Enter the starting address : 3
     Enter the length of the block : 4
     Successfully Allocated!
          MENU :
          1. Allocate
          2. Exit
          Choice : 1

     Enter the starting address : 2
     Enter the length of the block : 3
    Space cannot be allocated!

          MENU :
          1. Allocate
          2. Exit
          Choice : 1

     Enter the starting address : 7
     Enter the length of the block : 3
     Successfully Allocated!

          MENU :
          1. Allocate
          2. Exit
          Choice : 2


Final State of Memory Blocks

Index:  1  2  3  4  5  6  7  8  9 10
Block:  0  0  1  1  1  1  1  1  1  1
```

b) Indexed File Allocation Method

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_BLOCKS 100

int files[MAX_BLOCKS];
int blockSize[MAX_BLOCKS];
int numFiles = 0;

void initialize() {
    for (int i = 0; i < MAX_BLOCKS; i++) {
        files[i] = -1; // Initialize all blocks as empty (-1 represents empty)
    }
}

void allocateFile(int fileNumber, int size) {
    if (numFiles >= MAX_BLOCKS) {
        printf("Disk is full. Cannot allocate more files.\n");
        return;
    }

    if (size <= 0 || size > MAX_BLOCKS) {
        printf("Invalid file size.\n");
        return;
    }

    int startBlock = -1;
    int consecutiveBlocks = 0;

    for (int i = 0; i < MAX_BLOCKS; i++) {
        if (files[i] == -1) {
            if (consecutiveBlocks == 0) {
                startBlock = i;
            }
            consecutiveBlocks++;
        } else {
            consecutiveBlocks = 0;
            startBlock = -1;
        }

        if (consecutiveBlocks == size) {
            break;
        }
    }

    if (consecutiveBlocks == size) {
```

```
        for (int i = startBlock; i < startBlock + size; i++) {
            files[i] = fileNumber;
            blockSize[i] = size;
        }
        numFiles++;
        printf("File %d allocated starting from block %d\n", fileNumber,
startBlock);
    } else {
        printf("Not enough consecutive free blocks to allocate the file.\n");
    }
}

void deallocateFile(int fileNumber) {
    int blocksFreed = 0;

    for (int i = 0; i < MAX_BLOCKS; i++) {
        if (files[i] == fileNumber) {
            files[i] = -1;
            blocksFreed++;
        }
    }

    if (blocksFreed > 0) {
        numFiles--;
        printf("File %d deallocated. %d blocks freed.\n", fileNumber, blocksFreed);
    } else {
        printf("File %d not found on the disk.\n", fileNumber);
    }
}

void displayDiskStatus() {
    printf("\nDisk Status:\n");
    for (int i = 0; i < MAX_BLOCKS; i++) {
        if (files[i] != -1) {
            printf("Block %d: File %d (Size: %d blocks)\n", i, files[i], blockSize[i]);
        }
    }
}

int main() {
    initialize();

    while (1) {
        printf("\nFile Allocation Menu:\n");
        printf("1. Allocate a File\n");
        printf("2. Deallocate a File\n");
        printf("3. Display Disk Status\n");
        printf("4. Exit\n");

        int choice;
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (numFiles >= MAX_BLOCKS) {
                    printf("Disk is full. Cannot allocate more files.\n");
                } else {
                    int fileNumber, size;
                    printf("Enter File Number and Size: ");
                    scanf("%d %d", &fileNumber, &size);
                    allocateFile(fileNumber, size);
                }
                break;
            case 2:
                if (numFiles <= 0) {
                    printf("No files to deallocate.\n");
                } else {
                    int fileNumber;
                    printf("Enter File Number to deallocate: ");
                    scanf("%d", &fileNumber);
                    deallocateFile(fileNumber);
                }
                break;
            case 3:
                displayDiskStatus();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
File Allocation Menu:
1. Allocate a File
2. Deallocate a File
3. Display Disk Status
4. Exit
Enter your choice: 1
Enter File Number and Size: 1 5
File 1 allocated starting from block 0

File Allocation Menu:
1. Allocate a File
2. Deallocate a File
3. Display Disk Status
4. Exit
Enter your choice: 1
Enter File Number and Size: 2 3
File 2 allocated starting from block 5

File Allocation Menu:
1. Allocate a File
2. Deallocate a File
3. Display Disk Status
4. Exit
Enter your choice: 3

Disk Status:
Block 0: File 1 (Size: 5 blocks)
Block 1: File 1 (Size: 5 blocks)
Block 2: File 1 (Size: 5 blocks)
Block 3: File 1 (Size: 5 blocks)
Block 4: File 1 (Size: 5 blocks)
Block 5: File 2 (Size: 3 blocks)
Block 6: File 2 (Size: 3 blocks)
Block 7: File 2 (Size: 3 blocks)

File Allocation Menu:
1. Allocate a File
2. Deallocate a File
3. Display Disk Status
4. Exit
Enter your choice: 2
Enter File Number to deallocate: 1
File 1 deallocated. 5 blocks freed.
```

File Allocation Menu:
1. Allocate a File
2. Deallocate a File
3. Display Disk Status
4. Exit
Enter your choice: 3

Disk Status:
Block 5: File 2 (Size: 3 blocks)
Block 6: File 2 (Size: 3 blocks)
Block 7: File 2 (Size: 3 blocks)

File Allocation Menu:
1. Allocate a File
2. Deallocate a File
3. Display Disk Status
4. Exit
Enter your choice: 4

c) Linked File Allocation Method

```
#include <stdio.h>
#include <stdlib.h>

void recursiveParts(int pagesAllocation[], int size) {
    int s, length, k, c, j;

    while (1) {  // Convert recursion to a loop
        printf("Enter the beginning block's index as well as length: ");
        if (scanf("%d%d", &s, &length) != 2 || s < 0 || s >= size || length <= 0 ||
(s + length) > size) {
            printf("Error: Invalid block range.\n");
            return;
        }

        k = length;
        if (pagesAllocation[s] == 0) {
            for (j = s; j < (s + k); j++) {
                if (pagesAllocation[j] == 0) {
                    pagesAllocation[j] = 1;
                    printf("%d ------> %d\n", j, pagesAllocation[j]);
                } else {
                    printf("The block %d has already been allocated \n", j);
                }
            }
        } else {
            printf("The block %d has already been allocated \n", s);
        }

        printf("Do you want to add more files? \n");
        printf("Enter 1 for continue, Enter 0 for No: ");
        if (scanf("%d", &c) != 1 || (c != 0 && c != 1)) {
            printf("Invalid input.\n");
            return;
        }

        if (c == 0)
            break;
    }
}

int main() {
    int pagesAllocation[50] = {0};
    int p1, a1;

    printf("Enter the quantity of provided blocks: ");
    if (scanf("%d", &p1) != 1 || p1 <= 0 || p1 > 50) {
```

```
        printf("Error: Invalid block quantity.\n");
        return 1;
    }

    printf("Enter the number of allotted blocks: ");
    for (int i = 0; i < p1; i++) {
        if (scanf("%d", &a1) != 1 || a1 < 0 || a1 >= 50) {
            printf("Error: Invalid block index.\n");
            return 1;
        }
        pagesAllocation[a1] = 1;
    }

    recursiveParts(pagesAllocation, 50);

    return 0;
}
```

Output:
Case 1: Successful Allocation
Input and Output
Enter the quantity of provided blocks: 5
Enter the number of allotted blocks: 2 5 10 15 20
Enter the beginning block's index as well as length: 6 4
6 ------> 1
7 ------> 1
8 ------> 1
9 ------> 1
Do you want to add more files?
Enter 1 for continue, Enter 0 for No: 0

Case 2: Trying to Allocate an Already Allocated Block
Input and Output:
Enter the quantity of provided blocks: 4
Enter the number of allotted blocks:
2 5 10 15
Enter the beginning block's index as well as length: 5 3
The block 5 has already been allocated
Do you want to add more files?
Enter 1 for continue, Enter 0 for No: 0

Case 3: Trying to Allocate Blocks Out of Range
Input and Output:
Enter the quantity of provided blocks: 3
Enter the number of allotted blocks:
2 8 15
Enter the beginning block's index as well as length: 48 5
Error: Invalid block range.

Case 4: Invalid Input for Block Allocation
Input:
Enter the quantity of provided blocks: -3
Output:
Error: Invalid block quantity.

Case 5: Invalid Input for Allotted Blocks
Input:
Enter the quantity of provided blocks: 2
Enter the number of allotted blocks:
abc
Output:
Error: Invalid block index.

Experiment-3

Download and install nachos operating system and experiment with it.

Steps to Download, Install, and Experiment with Nachos Operating System

Nachos (Not Another Completely Heuristic Operating System) is an instructional operating system used for teaching OS concepts. Below are the steps to download, install, and run experiments with Nachos.

Step 1: Prerequisites

Ensure you have the following installed on your system:

- **Ubuntu/Linux (Recommended) or Windows with WSL**
- **GCC and Make** (for compilation)
- **Java JDK** (for Java-based Nachos versions)

On **Ubuntu/Linux**, install necessary dependencies:

bash
sudo apt update
sudo apt install build-essential git default-jdk

On **Windows**, install:

- **WSL (Windows Subsystem for Linux)**
- **GCC (via MinGW or Cygwin)**
- **JDK (from Oracle or OpenJDK)**

Step 2: Download Nachos Source Code

Clone the Nachos repository from GitHub or download it manually:

bash
git clone https://github.com/oxosz/Nachos
cd Nachos

Or download from a reliable source (if GitHub is unavailable).

Step 3: Compile Nachos

Navigate to the code directory and compile Nachos:

bash
cd code/build.linux
make clean
make

If using Windows (MinGW/Cygwin), navigate to code/build.windows and run:

bash
make clean
make

Step 4: Run Nachos Kernel

Run a simple test to check if Nachos is working:

bash
cd code/userprog
../build.linux/nachos -x ../test/halt

This should execute the halt program.

Step 5: Experiment with Nachos

1. Running User Programs

Nachos supports user programs written in C. Example:
bash
../build.linux/nachos -x ../test/exit

2. Testing Process Scheduling

Run multiple user programs to see how Nachos schedules them:
bash
../build.linux/nachos -x ../test/matmult -x ../test/sort

3. Virtual Memory and Paging

Modify machine/machine.cc to test paging mechanisms.

4. File System Operations

Create and list files in Nachos:
bash
../build.linux/nachos -f # Format Nachos filesystem
../build.linux/nachos -cp ../test/sort sort
../build.linux/nachos -ls # List files

Step 6: Modify Nachos Source Code (For Experiments)

You can modify Nachos for custom experiments:

- **Modify process scheduling in scheduler.cc**
- **Experiment with virtual memory in addrspace.cc**
- **Extend file system support in filesys.cc**

After changes, recompile Nachos:
bash
cd code/build.linux
make clean
make

Step 7: Debugging Nachos

Use GDB (GNU Debugger) to debug Nachos:
bash
gdb ../build.linux/nachos
run -x ../test/halt
Set breakpoints in progtest.cc to debug process execution.

Step 8: Cleanup and Uninstall Nachos

If you want to remove Nachos:
bash
CopyEdit
rm -rf Nachos