OPERATING SYSTEMS
Mini Project

OS FILE SYSTEM CONTROL AND DEADLOCK MANAGEMENT WITH MULTI THREADING

Configuring a Shell - File Management tool and Implementing Deadlock Recovery Mechanisms in Multithreaded Environments

Project done by ;

Ithikash R.	**2022510023**
Mirsha A. K.	**2022510044**

ABSTRACT

The first part of this project addresses the crucial challenge of **managing file systems** efficiently by implementing a comprehensive set of functionalities for manipulation and maintenance. It encompasses tasks such as **file creation, viewing, modification, size retrieval, details extraction, clearing, and deletion**. These operations are vital for ensuring effective file system management and user interaction. Key techniques employed include dynamic content modification, file size determination, and detailed file analysis. The project leverages system resources and programming constructs to offer a seamless **file management experience, enhancing user productivity and system reliability.**

The second part of the project tackles the critical challenge of **ensuring system reliability and stability in multithreading environments** through the implementation of **robust deadlock recovery mechanisms**. It covers strategies such as deadlock detection using algorithms like Banker's algorithm, emphasizing controlled resource allocation to prevent deadlocks. The core focus lies in the dynamic implementation of recovery mechanisms, including preemptive actions such as resource preemption or thread state rollback upon deadlock detection. Thread synchronization techniques using mutexes, semaphores, and condition variables facilitate coordinated resource access among threads, mitigating deadlock risks. Rigorous testing and debugging validate the effectiveness of these mechanisms under diverse scenarios, recognizing the **inherent complexity of deadlock recovery in varying system contexts**.

OBJECTIVE

The First objective of this Project is to develop an user-friendly file system management tool which is capable of performing various operations on files.

The primary goals include:

- Implementing functionalities for creating, viewing, modifying, and deleting files to facilitate efficient file manipulation.
- Providing features to obtain essential file details such as size, creation date, and modification date to aid in file analysis.
- Offering mechanisms for clearing file content to maintain data privacy and security.
- Ensuring user interaction through an intuitive menu-driven interface for easy navigation and operation.
- Leveraging system resources and programming constructs to optimize file management tasks and enhance overall system reliability.

The second objective of this project is to develop a robust deadlock recovery mechanism for multithreaded environments. This involves implementing strategies such as **deadlock detection, resource allocation, and recovery mechanisms** to ensure system reliability and stability in the face of deadlock scenarios.

This project aims to:

- Implement a deadlock detection algorithm to periodically check for deadlock occurrences and identify the threads involved.
- Ensure consistent and controlled resource allocation to prevent deadlock situations, utilizing methods such as resource hierarchies and request protocols.
- Develop a recovery mechanism to resolve deadlock scenarios, which may involve preempting resources from threads or rolling back thread states to a safe state.
- Employ synchronization primitives like mutexes, semaphores, and condition variables to coordinate resource access among multiple threads effectively.
- Conduct thorough testing and debugging to validate the correctness and effectiveness of the implemented deadlock recovery mechanisms under various scenarios.
- By achieving these objectives, the project aims to enhance the reliability and stability of multithreaded systems by providing efficient deadlock recovery solutions.

SCOPE

1. ***The scope of first part of the project*** involves the development of a versatile file system management tool capable of performing essential file manipulation tasks.

The project will cover the following key aspects:
-Design and implement functionalities for creating, viewing, modifying, and deleting files to facilitate efficient file manipulation.
-Develop mechanisms for users to interact with files through an intuitive menu-driven interface.

1.File Details Extraction:
-Implement features to obtain essential file details such as size, creation date, and modification date to provide users with comprehensive file information.

2.File Clearing:
-Provide a mechanism for clearing file content to enable users to maintain data privacy and security by securely erasing file contents.

3.Error Handling and User Feedback:
-Incorporate robust error handling mechanisms to handle file operation failures gracefully and provide informative error messages to users.
-Ensure clear and concise user feedback throughout the file manipulation process to enhance user experience and usability.

4.System Resource Optimization:
-Leverage system resources efficiently to optimize file management tasks and enhance overall system performance.
-Implement mechanisms to minimize resource consumption and maximize system responsiveness during file operations.

2. The scope of the second part of this project encompasses the design, implementation, and testing of a deadlock recovery mechanism for multithreaded environments. The project will focus on the following key aspects:

1. **Deadlock Detection**:
- Design and implement a deadlock detection algorithm to periodically check for deadlock occurrences in the system.
- Develop mechanisms to identify the threads involved in deadlock situations.

2. **Resource Allocation**:
   - Implement resource allocation methods to ensure resources are allocated in a consistent and controlled manner.
   - Explore techniques such as resource hierarchies, request protocols, and preemption to avoid deadlock situations.

3. **Recovery Mechanism**:
   - Design and implement a robust recovery mechanism to resolve deadlock scenarios effectively.
   - Develop algorithms to preempt resources from threads or roll back thread states to mitigate deadlocks.

4. **Thread Synchronization**:
   - Utilize synchronization primitives such as mutexes, semaphores, and condition variables to coordinate resource access among multiple threads.
   - Ensure threads acquire and release resources in a synchronized manner to prevent deadlock situations.

5. **Testing and Validation**:
   - Conduct extensive testing to validate the correctness and effectiveness of the deadlock recovery mechanisms.
   - Test the system under various scenarios and edge cases to ensure robustness and reliability.

The project will focus on providing a comprehensive solution for deadlock recovery in multithreaded environments while considering the complexities and challenges inherent in such systems.

Limitations

1. The File Management tool has the following limitations ;

1. Can operate only on files with “ Terminal - Shell like Interface “ . No Sophisticated Graphical User Interfaces involved.
2. File Security features like authentication and password protect encryption features are not fused into this mini - project yet.

3. All the files are considered to be functional only in the offline mode of use, and non-operational online.

1. The Deadlock Recovery and Management tool has the following limitations ;

1. The Deadlock Avoidance, Preventive Manager and Recovery Module has been implemented with only maximum of ;

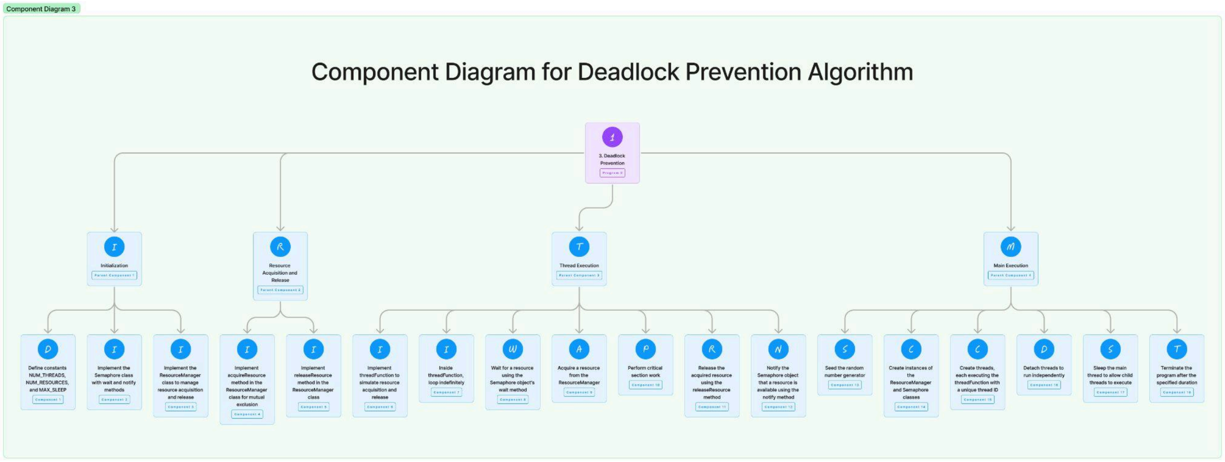
- 5 Threads
- 3 Resources

for the simplicity aspect and Accuracy of the Recovery and Management Program involved.

2. For more real- world problems on Deadlock Challenges and Problems, complicated Recovery Mechanisms and Tools need to be deployed.
3. “ The Banker’s Algorithm “ implemented using the C ++ program and multi threading libraries are only suitable for processes and threads with static Resource / Memory Requirements.

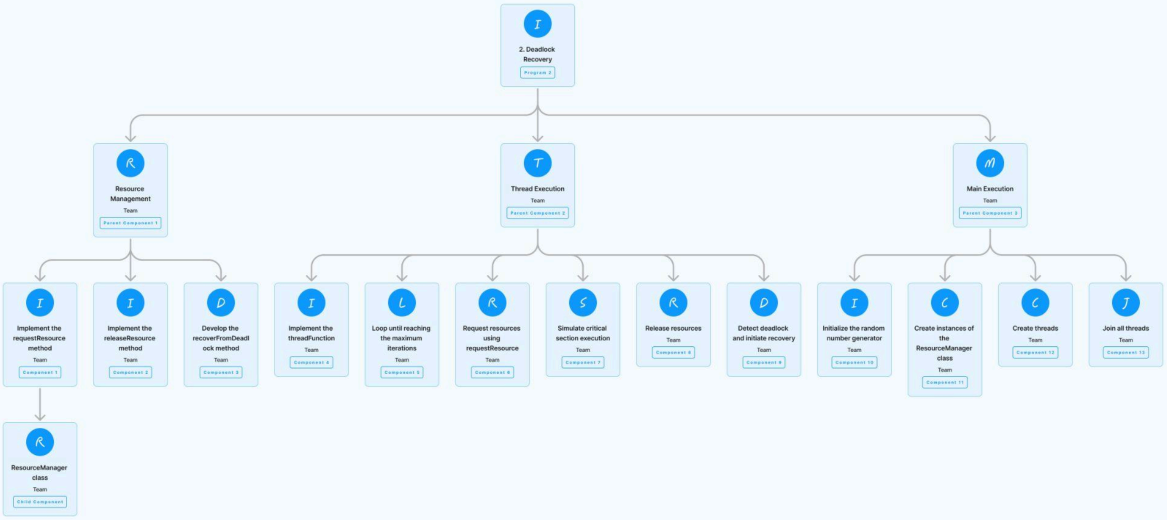
DIAGRAM

1. Deadlock Management Component Diagrams


Component Diagram 3
Component Diagram for Deadlock Prevention Algorithm
1
3. Deadlock Prevention
Program 3
I
Initialization
Parent Component 1
R
Resource Acquisition and Release
Parent Component 2
T
Thread Execution
Parent Component 3
M
Main Execution
Parent Component 4
D
Define constants NUM_THREADS, NUM_RESOURCES, and MAX_SLEEP
Component 1
I
Implement the Semaphore class with wait and notify methods
Component 2
I
Implement the ResourceManager class to manage resource acquisition and release
Component 3
I
Implement acquireResource method in the ResourceManager class for mutual exclusion
Component 4
I
Implement releaseResource method in the ResourceManager class
Component 5
I
Implement threadFunction to simulate resource acquisition and release
Component 6
I
Inside threadFunction, loop indefinitely
Component 7
W
Wait for a resource using the Semaphore object's wait method
Component 8
A
Acquire a resource from the ResourceManager
Component 9
P
Perform critical section work
Component 10
R
Release the acquired resource using the releaseResource method
Component 11
N
Notify the Semaphore object that a resource is available using the notify method
Component 12
S
Seed the random number generator
Component 13
C
Create instances of the ResourceManager and Semaphore classes
Component 14
C
Create threads, each executing the threadFunction with a unique thread ID
Component 15
D
Detach threads to run independently
Component 16
S
Sleep the main thread to allow child threads to execute
Component 17
T
Terminate the program after the specified duration
Component 18

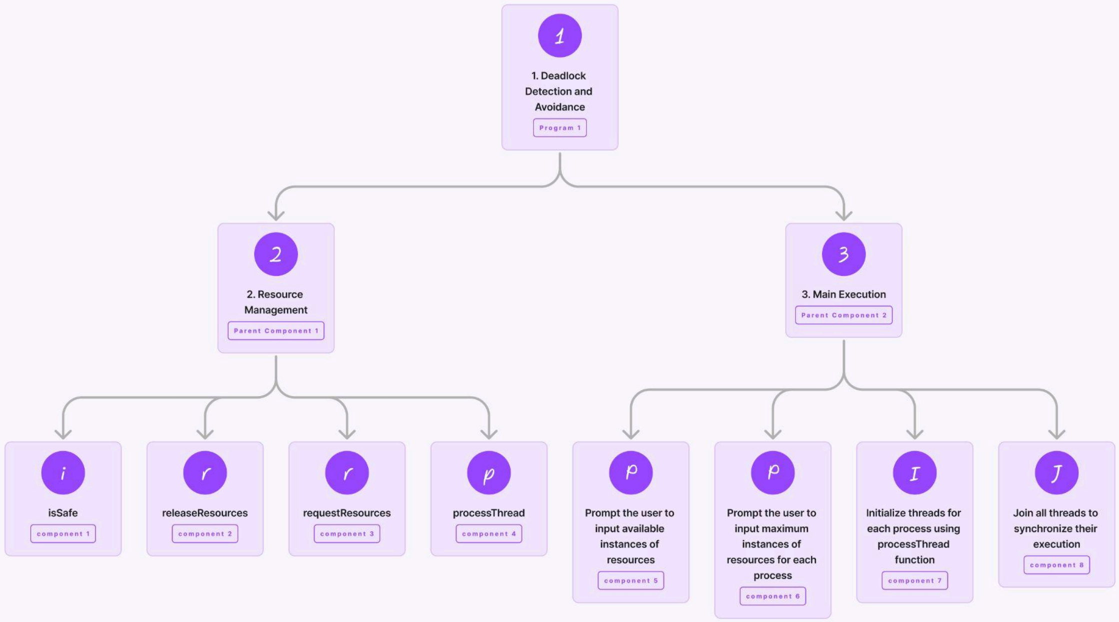
Component Diagram 2

Component Diagram for Deadlock Recovery Algorithm

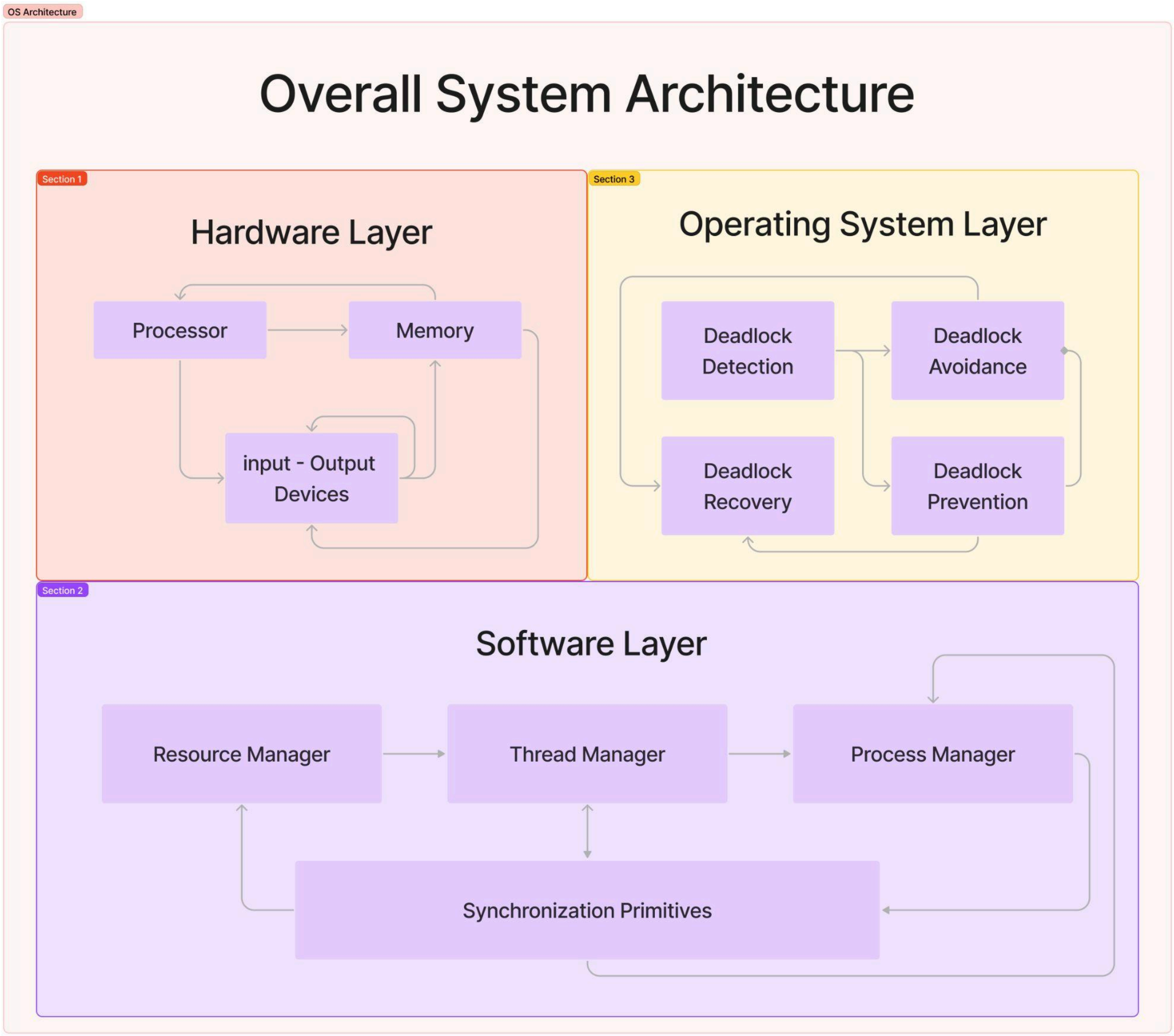


Component Diagram 1

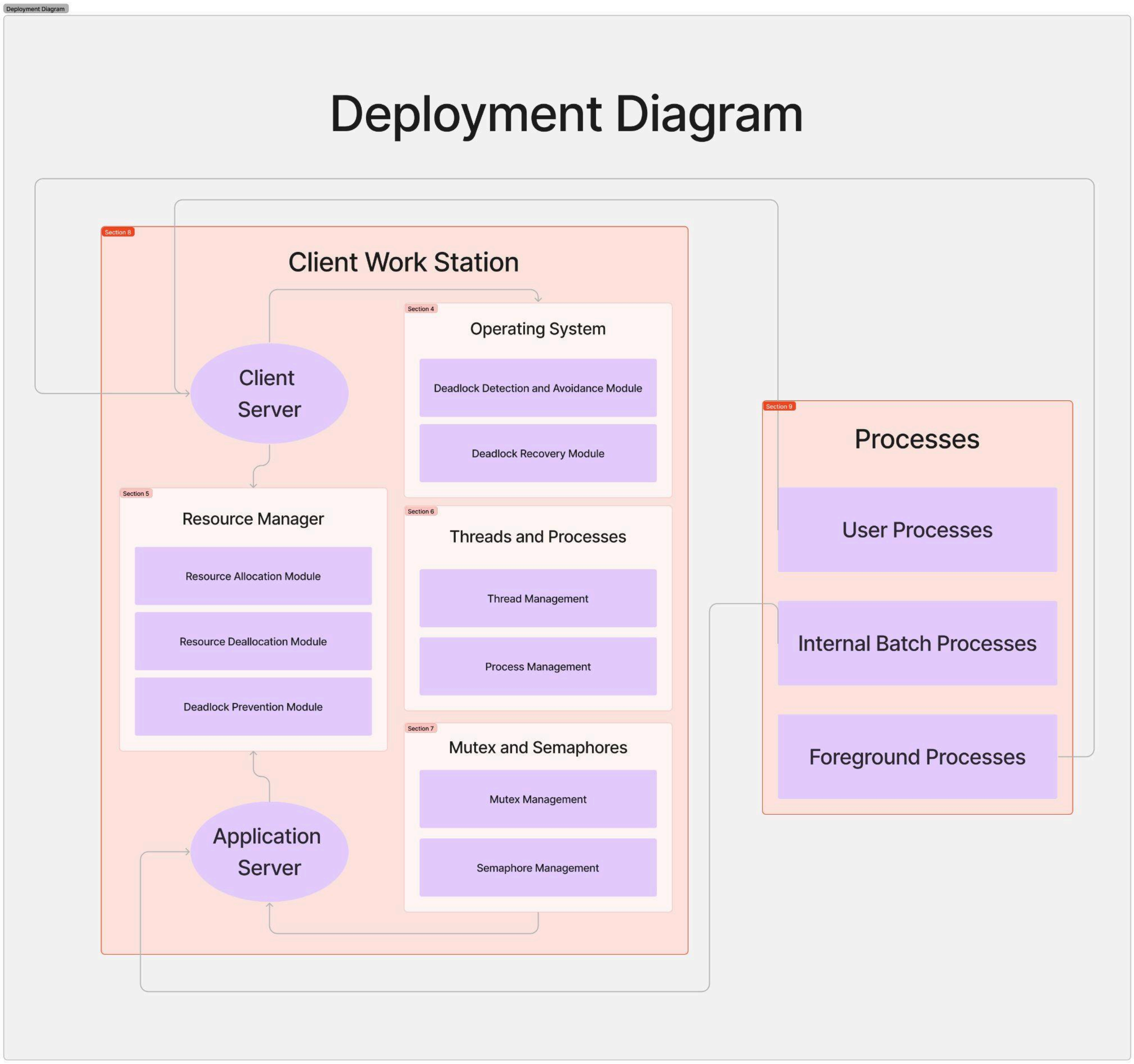
Component Diagram for Deadlock Detection and Avoidance using Banker's Algorithm


1
1. Deadlock Detection and Avoidance
Program 1
2
2. Resource Management
Parent Component 1
3
3. Main Execution
Parent Component 2
i
isSafe
component 1
r
releaseResources
component 2
r
requestResources
component 3
p
processThread
component 4
P
Prompt the user to input available instances of resources
component 5
P
Prompt the user to input maximum instances of resources for each process
component 6
I
Initialize threads for each process using processThread function
component 7
J
Join all threads to synchronize their execution
component 8

2. Deadlock Management Overall System Architecture Diagram

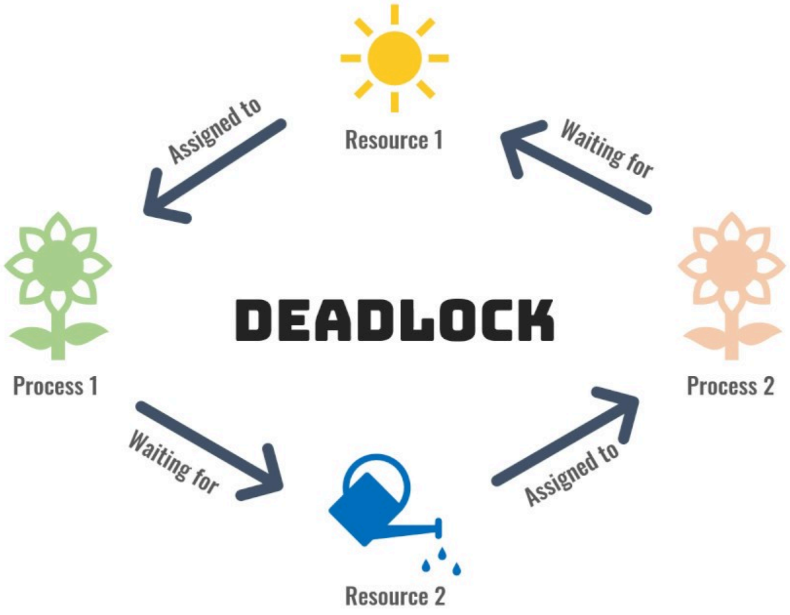


3. Deadlock Management Deployment Diagram

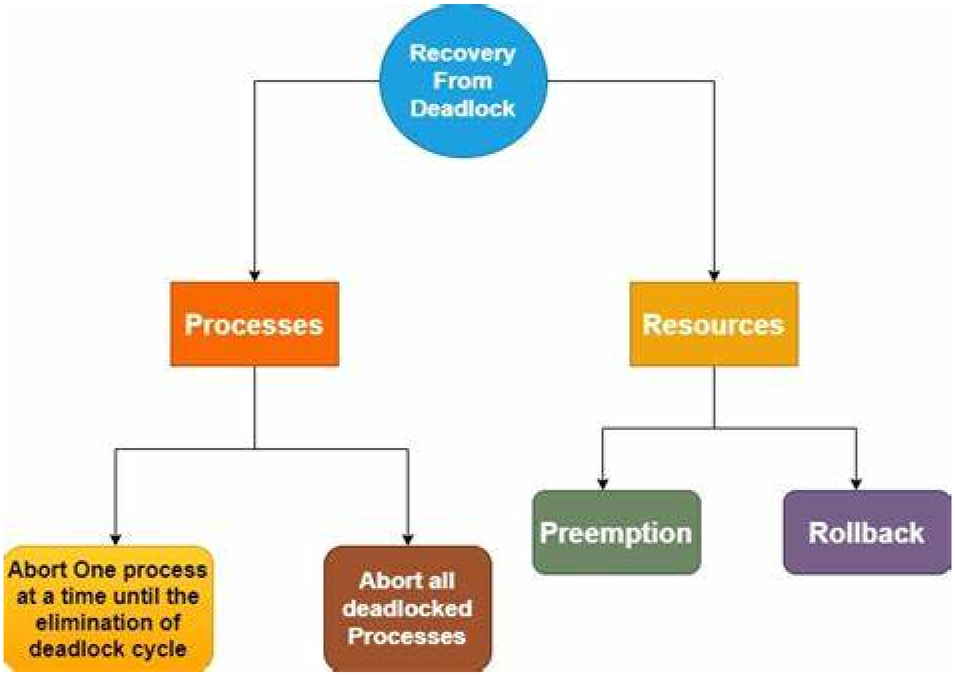


4. Data Flow Diagrams

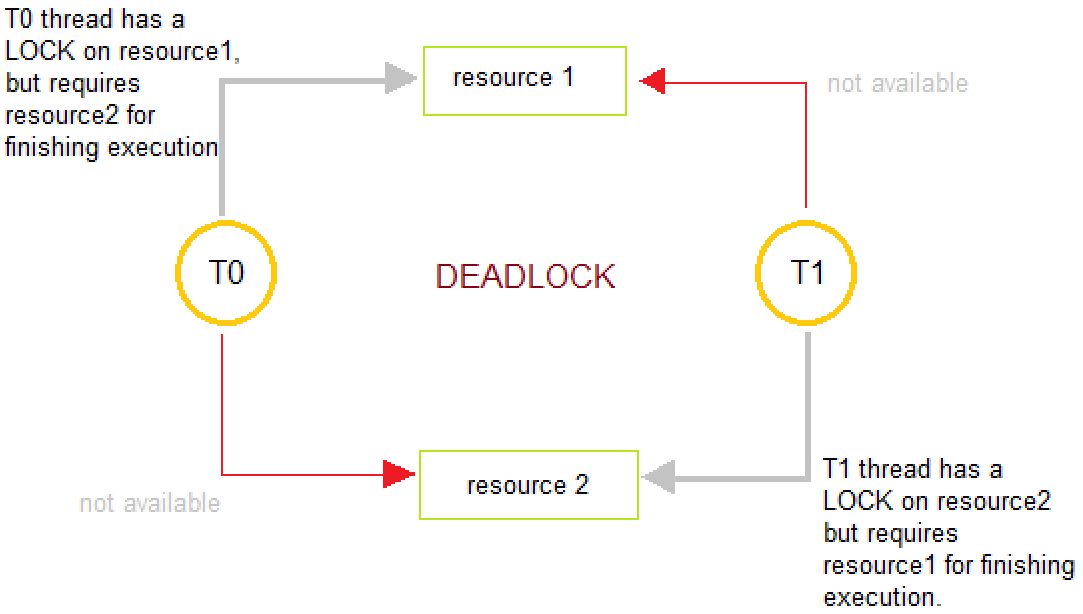
Deadlock Overview :



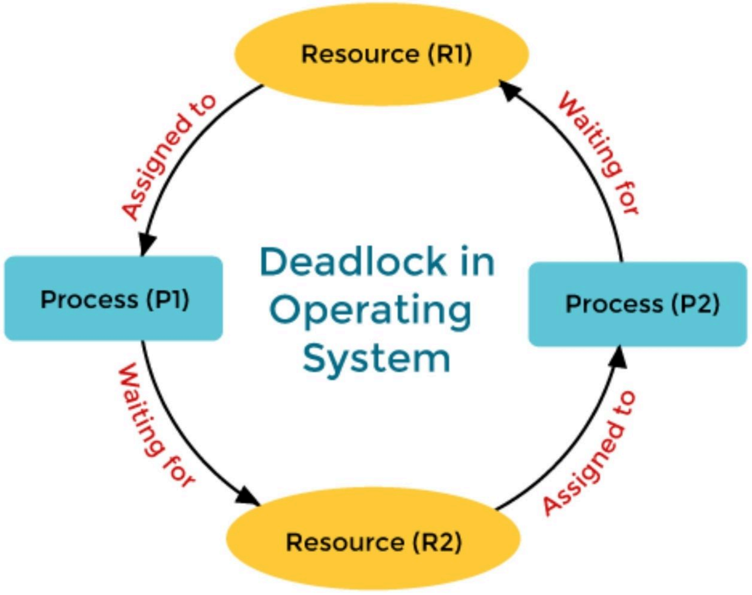
Deadlock Recovery Plans :


Recovery From Deadlock
Processes
Resources
Abort One process at a time until the elimination of deadlock cycle
Abort all deadlocked Processes
Preemption
Rollback

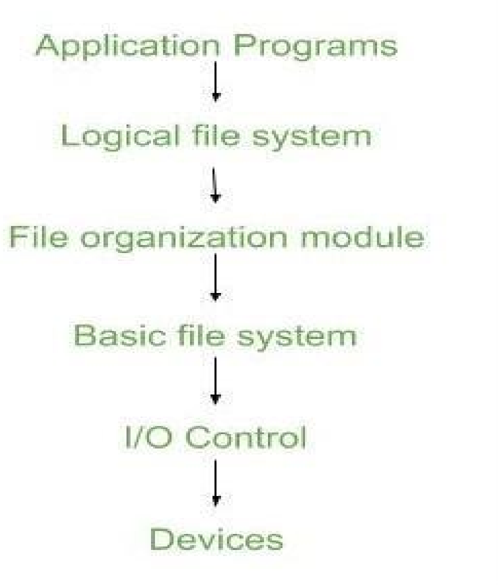
Deadlock in Multi- Threading:

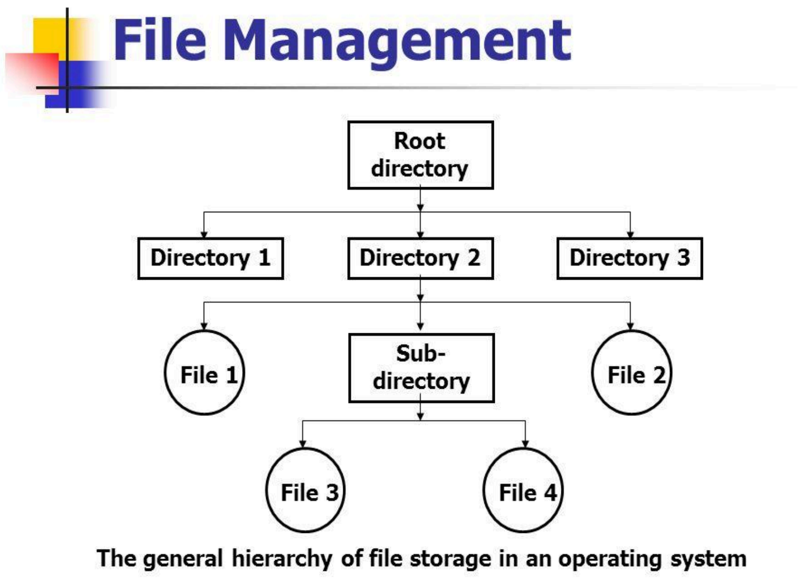

T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
resource 2
not available
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

Deadlock in Multiple Processes :



5. File System Architecture Diagram

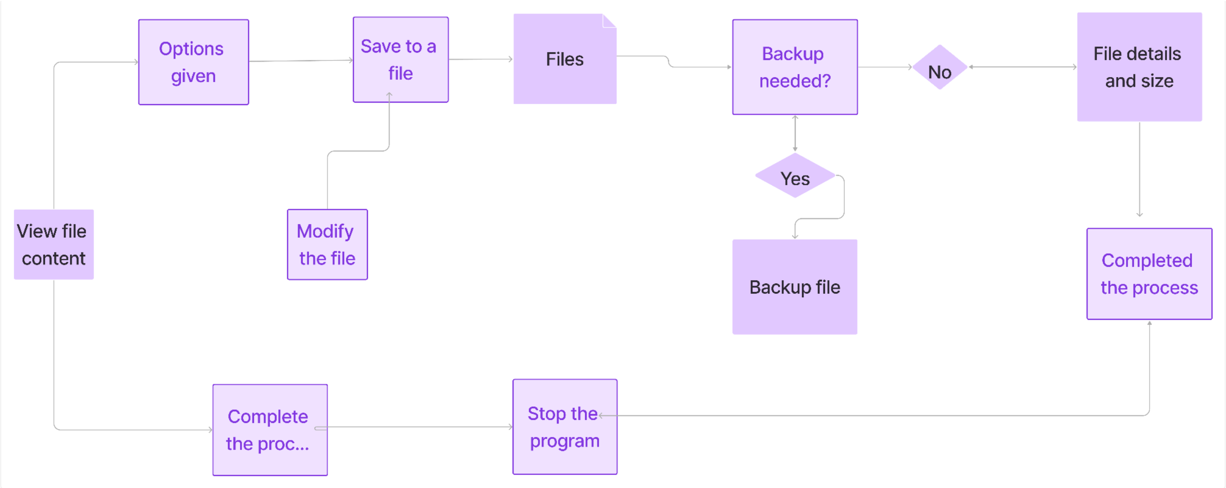




File Management

The general hierarchy of file storage in an operating system

6. File Management Deployment Diagram



ALGORITHM

1. File Management System:

1. File Creation and Management Algorithm

1.1 Initialization:

1.1.1 Initialize necessary constants and variables for file management operations.

1.1.2 Set up a mutex for thread synchronization to ensure concurrent file operations are executed safely.

1.2 File Creation:

1.2.1 Implement functionality to create a new file and initialize it with user-provided content.

1.2.2 Handle errors gracefully if file creation fails due to permission issues or other system constraints.

1.3 File Viewing:

1.3.1 Develop mechanisms to allow users to view the contents of an existing file.

1.3.2 Ensure proper error handling if the file specified by the user cannot be opened or does not exist.

1.4 File Modification:

1.4.1 Implement functionality for users to modify the content of an existing file.

1.4.2 Ensure that the original content of the file is replaced entirely with the new content provided by the user.

1.5 File Deletion:

1.5.1 Provide options for users to delete unwanted files from the system.

1.5.2 Handle file deletion operations carefully to prevent accidental data loss and confirm user intent before proceeding.

2. File Information Retrieval Algorithm

2.1 Obtain File Size:

2.1.1 Develop mechanisms to retrieve the size of a specified file in bytes.

2.1.2 Utilize appropriate system calls or library functions to accurately determine the size of the file.

2.2 Retrieve File Details:

2.2.1 Implement functionality to extract additional details about a file, such as its creation date, modification date, and other metadata.

2.2.2 Ensure that the retrieved file details are presented to the user in a clear and understandable format.

3. Error Handling and User Interaction

3.1 Implement robust error handling mechanisms to handle various exceptional scenarios, such as file not found, permission denied, or insufficient disk space.

3.2 Provide informative feedback to users regarding the outcome of file management operations, including success messages or error notifications.

3.3 Design the user interface to be intuitive and user-friendly, guiding users through the file management process seamlessly.

By following this algorithm, the file system management tool ensures efficient and reliable handling of file operations while providing a smooth user experience.

2. Deadlock hManagement using multi threading

1. Deadlock Detection and Avoidance using Banker's Algorithm

1. Initialization:
1.1 Initialize constants NUM_RESOURCES and NUM_PROCESSES.
1.2 Initialize vectors available, maximum, allocation, need, and resourceGranted for resource management.
1.3 Acquire a mutex for thread synchronization.

2. Resource Management:
2.1 Implement isSafe function to check if a process can proceed safely.
2.2 Implement releaseResources function to release resources held by a process.
2.3 Implement requestResources function to request resources for a process, ensuring deadlock prevention and recovery.
2.4 Develop processThread function to simulate resource allocation and deallocation for each process in a separate thread.

3. Main Execution:
3.1 Prompt the user to input available instances of resources.
3.2 Prompt the user to input maximum instances of resources for each process.
3.3 Initialize threads for each process using processThread function.
3.4 Join all threads to synchronize their execution.

2. Deadlock Recovery Algorithm

1. Initialization:
1.1 Define constants NUM_THREADS, NUM_RESOURCES, MAX_SLEEP, and MAX_ITERATIONS.
1.2 Implement the ResourceManager class to manage resources, including a mutex, condition variable, and vector to track resource availability.
1.3 Initialize a random number generator for deadlock recovery.

2. Resource Management:
2.1 Implement requestResource method in the ResourceManager class to request resources for a thread, ensuring mutual exclusion.
2.2 Implement releaseResource method in the ResourceManager class to release acquired resources, notifying waiting threads.
2.3 Develop recoverFromDeadlock method in the ResourceManager class to preempt resources in case of deadlock detection.

3. Thread Execution:
3.1 Implement threadFunction to simulate resource acquisition and release by each thread.
3.2 Inside threadFunction, loop until reaching the maximum iterations.
3.3 Request resources using requestResource, simulate critical section execution, and release resources.
3.4 Detect deadlock and initiate recovery mechanism using recoverFromDeadlock if necessary.

4. Main Execution:
4.1 Initialize the random number generator.
4.2 Create instances of the ResourceManager class.
4.3 Create threads, each executing the threadFunction with a unique thread ID and reference to the ResourceManager.
4.4 Join all threads to synchronize their execution and ensure proper termination.

3. Deadlock Prevention Using Semaphores, Mutex, Condition variable, Multiple threads

1. Initialization:
1.1 Define constants NUM_THREADS, NUM_RESOURCES, and MAX_SLEEP.
1.2 Implement the Semaphore class with methods wait and notify to control resource access.
1.3 Implement the ResourceManager class to manage resource acquisition and release.

2. Resource Acquisition and Release:
2.1 Implement acquireResource method in the ResourceManager class to acquire resources, ensuring mutual exclusion.
2.2 Implement releaseResource method in the ResourceManager class to release acquired resources.

3. Thread Execution:
3.1 Implement threadFunction to simulate resource acquisition and release by each thread.
3.2 Inside threadFunction, loop indefinitely.
3.3 Wait for a resource to become available using the Semaphore object's wait method.
3.4 Acquire a resource from the ResourceManager.
3.5 Perform some work, simulating critical section execution.
3.6 Release the acquired resource using the releaseResource method of the ResourceManager.
3.7 Notify the Semaphore object that a resource is available using the notify method.

4. Main Execution:
4.1 Seed the random number generator.
4.2 Create instances of the ResourceManager and Semaphore classes.
4.3 Create threads, each executing the threadFunction with a unique thread ID.
4.4 Detach threads to allow them to run independently.
4.5 Sleep the main thread to allow child threads to execute.
4.6 Terminate the program after the specified duration.

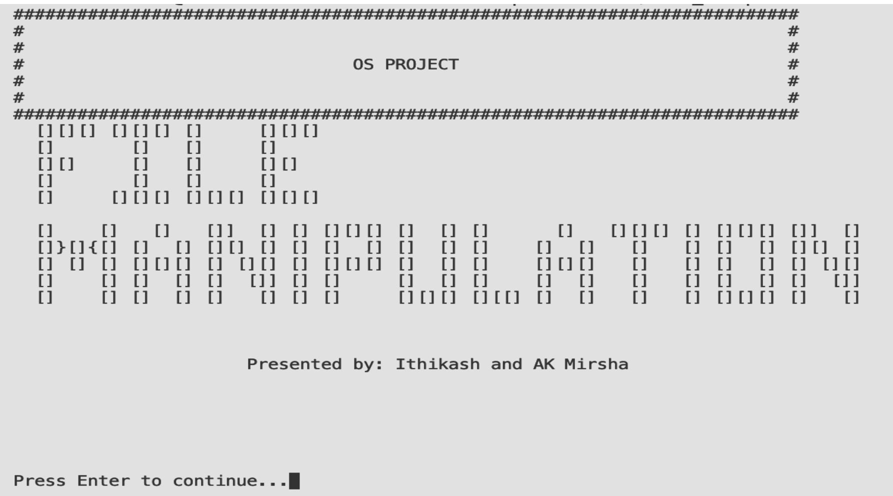
SOURCE CODE

1.File Management System

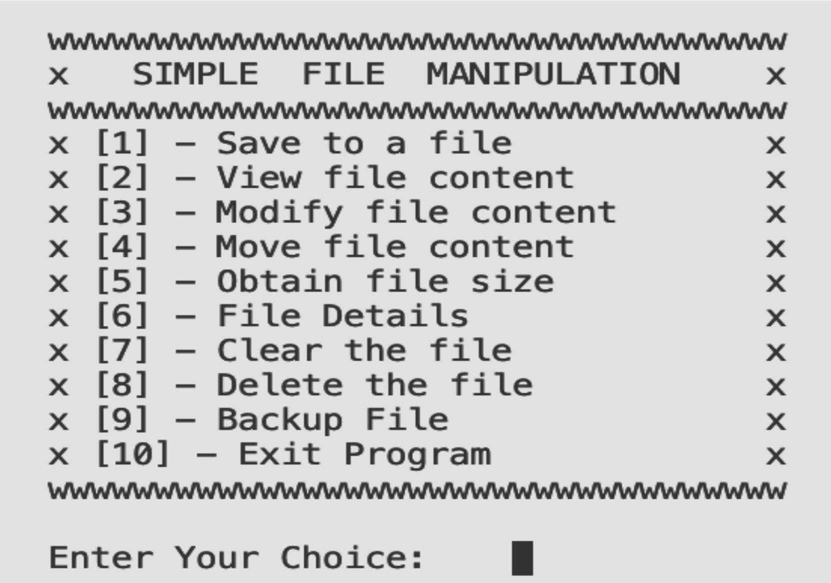
```
#include <iostream>
#include <fstream>
#include <ctime>
#include <sys/types.h>
#include <sys/stat.h>
#include <cerrno>
#include <cstdio>
#include <unistd.h> // Include for system("clear")
#include <string>
#include <vector>
#include <mutex> // Include for mutex
using namespace std;
mutex fileMutex;

int main() {
   // Display the title of the program
   cout << "            OS PROJECT \n\n";
  cout << "  [][][] [][][] []     [][][]\n";
  cout << "  []       []   []     []\n";
  cout << "  [][]     []   []     [][]\n";
  cout << "  []       []   []     [] \n";
  cout << "  []     [][][] [][][] [][][]\n\n";

 cout << "  []    []   []   []]  [] [] [][][] []  [] []      []   [][][] [] [][][] []]  []\n";
 cout << "  []}[]{[] []  [] [][] [] [] []  [] []  [] []    []  []   []   [] []  [] [][] []\n";
 cout << "  [] [] [] [][][] [] [][] [] [][][] []  [] []    [][][]   []   [] []  [] [] [][]\n";
 cout << "  []    [] []  [] []  []] [] []     []  [] []    []  []   []   [] []  [] []  []]\n";
 cout << "  []    [] []  [] []   [] [] []     [][][] [][[] []  []   []   [] [][][] []   []\n\n";

  cout  <<  "\n\n\t\t            Presented  by:  Ithikash  and  AK  Mirsha
\n\n\n\n\n\n\n";

   cout << "Press Enter to continue...";
   cin.get();

   system("clear");


   // Display the MENU
   string option;
menu:
   cout << endl;
   cout << "          wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww" << endl;
   cout << "          x   SIMPLE  FILE  MANIPULATION    x" << endl;
   cout << "          wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww" << endl;
   cout << "          x [1] - Save to a file            x" << endl;
   cout << "          x [2] - View file content         x" << endl;
   cout << "          x [3] - Modify file content       x" << endl;
   cout << "          x [4] - Move file content         x" << endl;
   cout << "          x [5] - Obtain file size          x" << endl;
   cout << "          x [6] - File Details              x" << endl;
   cout << "          x [7] - Clear the file            x" << endl;
   cout << "          x [8] - Delete the file           x" << endl;
   cout << "          x [9] - Backup File               x" << endl;
   cout << "          x [10] - Exit Program             x" << endl;
   cout << "          wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww" << endl;
   cout << "\n          Enter Your Choice:    ";
   getline(cin, option);
   if (option == "1")
   {
       //Saving a new file
       system("clear");
       string textToSave;
       cout << "        wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww" << endl;
       cout << "          ENTER THE STRING YOU WANT TO SAVE   x\n";
                                                             cout            <<
"wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwww\n\n" << endl;
       cout << " HERE: ";
       getline(cin, textToSave);
       std::string file_name;
       std::string ext = ".txt";

       cout<< " Enter File Name you wish to create : ";
       cin >> file_name;
       ofstream saveFile(file_name+ext);
       saveFile << textToSave;
       cout << "" << endl << endl << endl << endl << endl << endl;
       saveFile.close();
       system("pause");
       system("clear");
           goto menu;
}

if (option == "2")
{
       // View file content
       system("clear");
       std::string file_name;
       std::string ext = ".txt";

       cout << "Enter File Name you wish to view contents for: ";
       getline(cin, file_name);
       std::string combine = file_name + ext;

       ifstream loadFile(combine);
       if (!loadFile.is_open()) {
           cout << "Error: Unable to open file." << endl;
       }
       else {
           cout << "File Content:" << endl;
           cout << "--------------" << endl;
           char ch;
           while (loadFile.get(ch)) {
               cout << ch;
           }
           cout << endl;
           loadFile.close();
       }
       //system("pause");
       //system("clear");
       goto menu;
   }

if (option == "3")
{
   //Modify file content
   system("clear");
   std::string file_name;
   std::string ext = ".txt";

   cout << "Enter File Name you wish to modify: ";
   getline(cin, file_name);

    ofstream modifyFile(file_name + ext, ios::trunc); // Open file with trunc mode to clear existing content
   if (!modifyFile.is_open()) {
       cout << "Error: Unable to open file." << endl;
   }
   else {
        cout << "Enter the new content you want to write to the file (Press Enter twice to finish):" << endl;
       string line;
       while (getline(cin, line)) {
           if (line.empty()) {
               break;
           }
           modifyFile << line << endl;
       }
       modifyFile.close();
       cout << "\n\n\n\n";
       cout << "File successfully modified." << endl;
       cout << "\n\n\n\n";
   }
   //system("pause");
   //system("clear");
   goto menu; // Return to the main menu
}
if (option == "4")
{
   // Move file content
   system("clear");
   std::string source_file, destination_file;
   std::string ext = ".txt";

   cout << "Enter source File Name you wish to move content from: ";
   getline(cin, source_file);
   cout << "Enter destination File Name you wish to move content to: ";
   getline(cin, destination_file);

   ifstream source(source_file + ext);
     ofstream destination(destination_file + ext, ios::trunc); // Open
destination file with trunc mode to clear existing content

   if (!source.is_open() || !destination.is_open()) {
       cout << "Error: Unable to open file." << endl;
   }
   else {
           destination << source.rdbuf(); // Copy content from source to
destination
       source.close();
       destination.close();
       cout << "\n\n\n\n";
       cout << "File content successfully moved." << endl;
       cout << "\n\n\n\n";
   }

   //system("pause");
   //system("clear");
   goto menu;
}
if (option == "5")
{
   // Obtain file size
   system("clear");
   std::string file_name;
   std::string ext = ".txt";
   cout<< " Enter File Name you wish to obtain size for : ";
   getline(cin, file_name);
   ifstream myfile(file_name+ext, ios::binary | ios::ate);
   if (!myfile.is_open()) {
       cout << "Error: Unable to open file." << endl;
   }
   else {
       streampos fileSize = myfile.tellg();
       cout << "File Size: " << fileSize << " bytes" << endl;
       myfile.close();
   }
   //system("pause");
   //system("clear");
   goto menu;
}

 if (option == "6")
 {
       //  File Details
       system("clear");
       std::string file_name;
       std::string ext = ".txt";

       cout << "Enter File Name you wish to get the details for: ";
       getline(cin, file_name);

       struct stat fileInfo;
       if (stat((file_name + ext).c_str(), &fileInfo) != 0) {
           perror("Error");
       }
       else {
           cout << "File Size: " << fileInfo.st_size << " bytes" << endl;
            cout << "Drive letter saved: " << (char)(fileInfo.st_dev + 'A')
<< endl;
           cout << "Created: " << ctime(&fileInfo.st_ctime);
           cout << "Modified: " << ctime(&fileInfo.st_mtime);
       }
       // system("pause");
       // system("clear");
       goto menu;
   }
```

```
if (option == "7")
{
       // Clear the file
       system("clear");
       std::string file_name;
       std::string ext = ".txt";

       cout << "Enter File Name you wish to clear: ";
       getline(cin, file_name);

       ofstream ofs(file_name + ext, ios::trunc);
       cout << "\n\n\n\n";
       cout << "        wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww" << endl;
       cout << "        x      FILE SUCCESSFULLY CLEARED       x\n";
         cout << "          wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww\n\n" <<
endl;
       system("pause");
       system("clear");
       goto menu;
   }

   if (option == "8")
{
       //  Delete the file
       system("clear");
       std::string file_name;
       std::string ext = ".txt";

       cout << "Enter File Name you wish to delete: ";
       getline(cin, file_name);

       if (remove((file_name + ext).c_str()) != 0) {
           perror("Error");
       }
       else {
           cout << "\n\n\n\n";
           cout << "        wwwwwwwwwwwwwwwwww   " << endl;
           cout << "        x      FILE SUCCESSFULLY DELETED       x\n";
            cout << "        wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww\n\n"
<< endl;
       }
       system("pause");
       system("clear");
       goto menu;
       }
else if (option == "9")
{
   // Backup File
   string originalFileName;
   cout << "Enter the name of the file you wish to backup: ";
   getline(cin, originalFileName);
   originalFileName += ".txt";

   ifstream originalFile(originalFileName);
   if (!originalFile) {
       cout << "Error: Original file not found." << endl;
       goto menu;
   }
   originalFile.close();

   // Backup file name
   string backupFileName;
   cout << "Enter the name of the backup file: ";
   getline(cin, backupFileName);
   backupFileName += ".txt";

   // Create backup
   ifstream original(originalFileName, ios::binary);
   ofstream backup(backupFileName, ios::binary);
   backup << original.rdbuf();

   cout << "Backup created successfully." << endl;
   goto menu;
}

if (option == "10")
{
       system("color 0");
       system("cls");
           cout<<"\n\n";
           cout<<"[][][] [][][] [][][] [][][] [][][]    []    []    []\n";
           cout<<"[]  [] []  [] []  [] []     []  []  []  []  [][][][]\n";
           cout<<"[][][] [][][] []  [] [] []  [][][] [][][][] [] [] []\n";
           cout<<"[]     [] []  []  [] []  [] [] []  []    [] []    []\n";
           cout<<"[]     []  [] [][][] [][][] []  [] []    [] []    []\n";
           cout<<"[][][] [][][]     [][][] []     [][][] [][][] [][][] []   [] [][][]\n";
           cout<<"  []   []         []  [] []     []  [] []       []   []]  [] []\n";
           cout<<"  []   [][][]     []     []     []  [] [][][]   []   [][] [] [] []\n";
           cout<<"  []       []     []  [] []     []  []     []   []   [] [][] []  []\n";
           cout<<"[][][] [][][]     [][][] [][][] [][][] [][][] [][][] []  [[] [][][]\n";
           cout<<"[][][]  []     [] [][][]   [] [] []\n";
           cout<<"[]   []  []  []   []       [] [] []\n";
           cout<<"[]  []     []     [][][]   [] [] []\n";
           cout<<"[]   []    []     []\n";
           cout<<"[][][]     []     [][][]   [] [] []\n\n";

       system("pause");
       return 0;
    }

// Invalid option, return to menu
system("clear");
cout << "\n\n\n\n\n\n";
cout << "\t\t\t[][][] [][][] [][][] [][][] [][][]\n";
cout << "\t\t\t[]     []  [] []  [] []  [] []  []\n";
cout << "\t\t\t[][][] [][][] [][][] []  [] [][][]\n";
cout << "\t\t\t[]     [] []  [] []  []  [] [] []\n";
cout << "\t\t\t[][][] []  [] []  [] [][][] []  []\n\n";
cout << "\t\t\tPlease input a valid number.\n";
cout << "\t\t\tPress any key to go back to the Menu.\n\n\n\n\n\n\n";

system("pause");
system("clear");
goto menu;

return 0;
```

INPUT & OUTPUT:

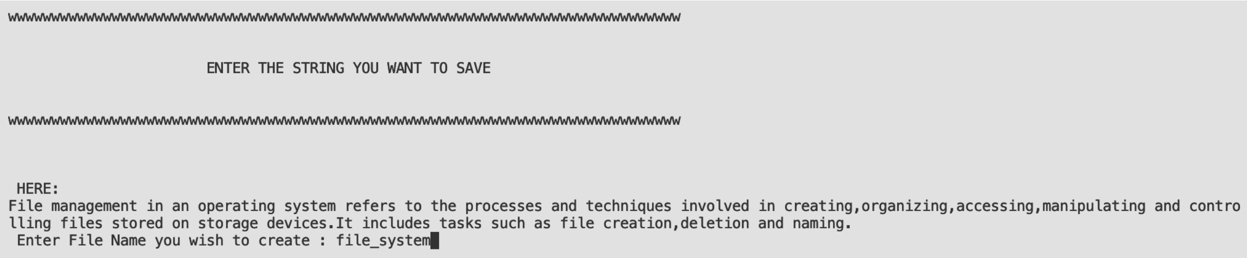


```
##########################################################################
#                                                                        #
#                                                                        #
#                               OS PROJECT                               #
#                                                                        #
#                                                                        #
##########################################################################
  [][][] [][][] []     [][][]
  []       []   []     []
  [][]     []   []     [][]
  []       []   []     []
  []     [][][] [][][] [][][]

  []    []   []   []]  [] [] [][][] []  [] []      []   [][][] [] [][][] []]  []
  []}[]{[] []  [] [][] [] [] []  [] []  [] []    []  []   []   [] []  [] [][] []
  [] [] [] [][][] [] [][] [] [][][] []  [] []    [][][]   []   [] []  [] [] [][]
  []    [] []  [] []  []] [] []     []  [] []    []  []   []   [] []  [] []  []]
  []    [] []  [] []   [] [] []     [][][] [][[] []  []   []   [] [][][] []   []


                      Presented by: Ithikash and AK Mirsha




Press Enter to continue...
```



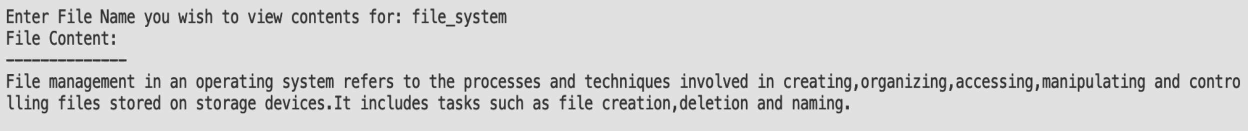
```
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
x   SIMPLE  FILE  MANIPULATION    x
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
x [1] - Save to a file            x
x [2] - View file content         x
x [3] - Modify file content       x
x [4] - Move file content         x
x [5] - Obtain file size          x
x [6] - File Details              x
x [7] - Clear the file            x
x [8] - Delete the file           x
x [9] - Backup File               x
x [10] - Exit Program             x
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww

Enter Your Choice:
```

1.SAVE TO A FILE



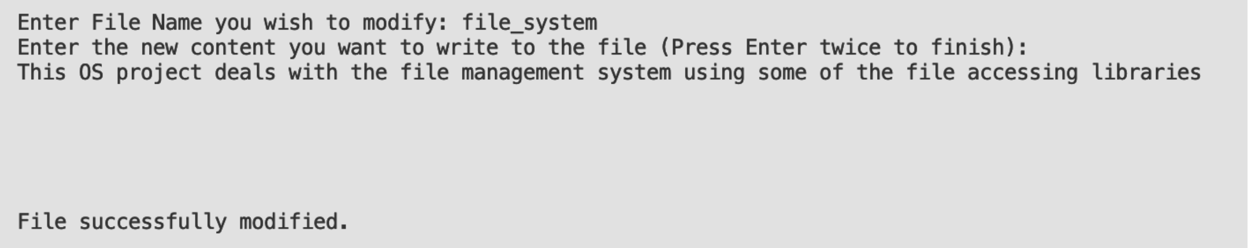
```
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww


                       ENTER THE STRING YOU WANT TO SAVE


wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww



 HERE:
File management in an operating system refers to the processes and techniques involved in creating,organizing,accessing,manipulating and contro
lling files stored on storage devices.It includes tasks such as file creation,deletion and naming.
 Enter File Name you wish to create : file_system
```

2.VIEW FILE CONTENT



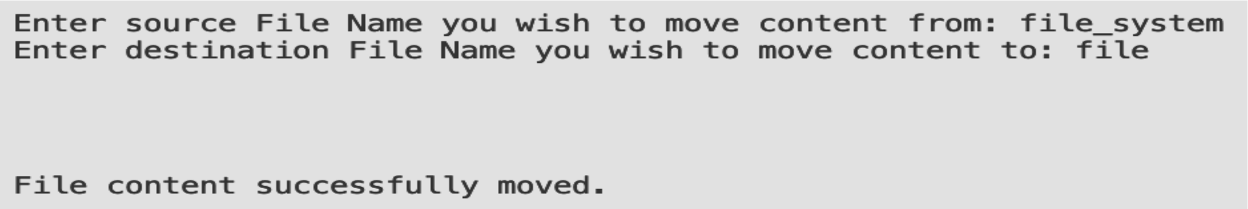
```
Enter File Name you wish to view contents for: file_system
File Content:
--------------
File management in an operating system refers to the processes and techniques involved in creating,organizing,accessing,manipulating and contro
lling files stored on storage devices.It includes tasks such as file creation,deletion and naming.
```

3.MODIFY FILE CONTENT

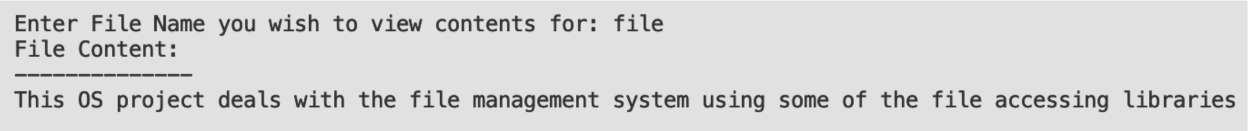


```
Enter File Name you wish to modify: file_system
Enter the new content you want to write to the file (Press Enter twice to finish):
This OS project deals with the file management system using some of the file accessing libraries





File successfully modified.
```

4.MOVE FILE CONTENT

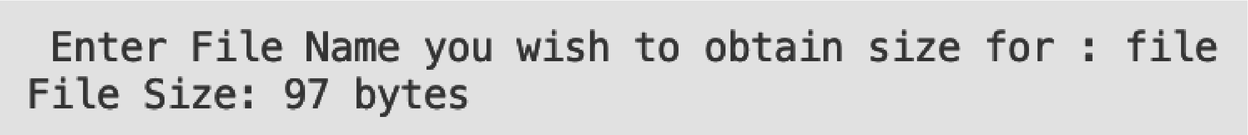


```
Enter source File Name you wish to move content from: file_system
Enter destination File Name you wish to move content to: file




File content successfully moved.
```



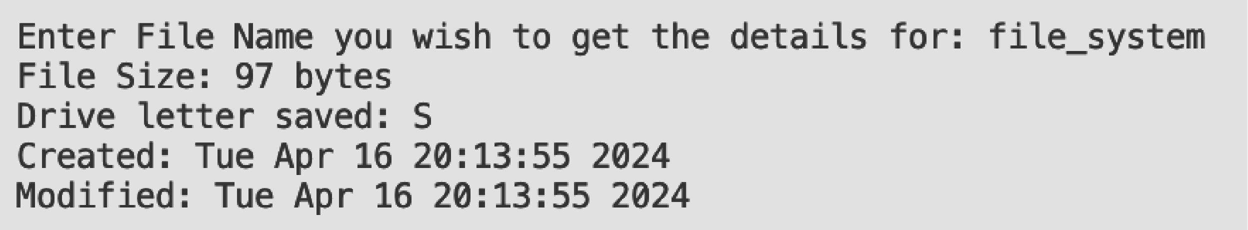
```
Enter File Name you wish to view contents for: file
File Content:
---------------
This OS project deals with the file management system using some of the file accessing libraries
```

5.OBTAIN FILE SIZE



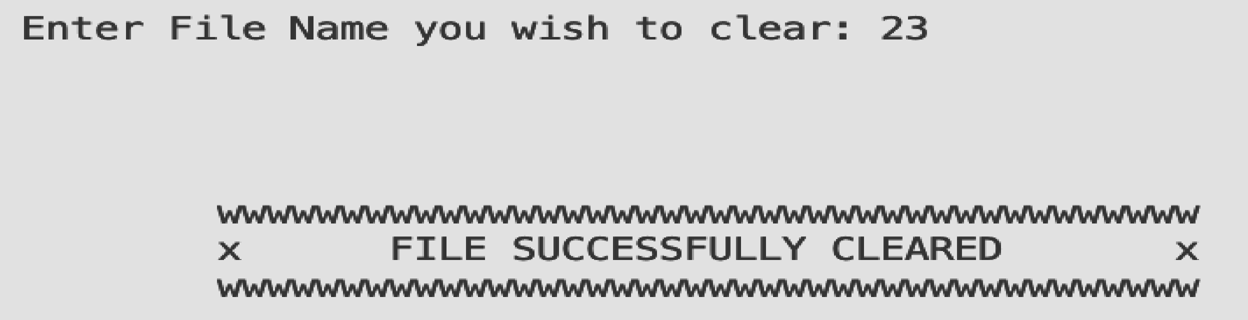
```
 Enter File Name you wish to obtain size for : file
File Size: 97 bytes
```

6.FILE DETAILS



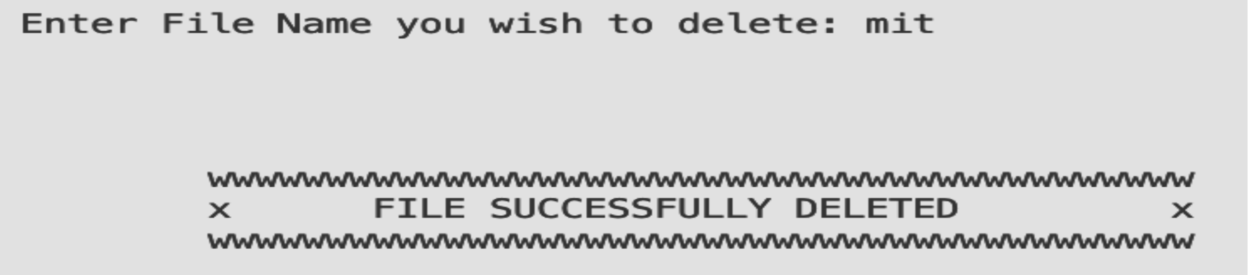
```
Enter File Name you wish to get the details for: file_system
File Size: 97 bytes
Drive letter saved: S
Created: Tue Apr 16 20:13:55 2024
Modified: Tue Apr 16 20:13:55 2024
```

7.CLEAR THE FILE



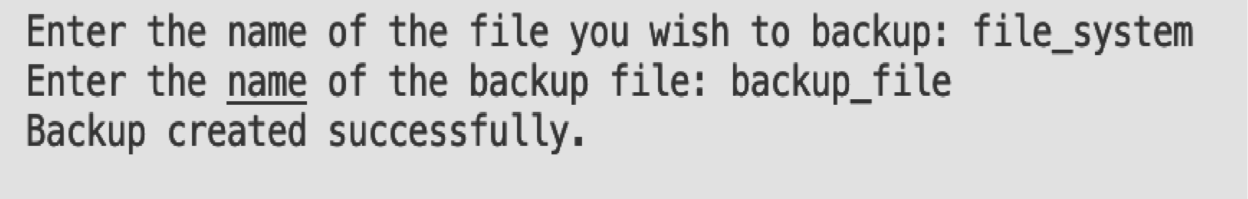
```
Enter File Name you wish to clear: 23



        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        x      FILE SUCCESSFULLY CLEARED       x
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

8.DELETING THE FILE



```
Enter File Name you wish to delete: mit



        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        x      FILE SUCCESSFULLY DELETED       x
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

9.BACKUP FILE



```
Enter the name of the file you wish to backup: file_system
Enter the name of the backup file: backup_file
Backup created successfully.
```

10.EXIT THE PROGRAM

```
[][][] [][][] [][][] [][][] [][][]    []    []    []
[]  [] []  [] []  [] []     []  []  []  []  [][][][]
[][][] [][][] []  [] [] []  [][][] [][][][] [] [] []
[]     [] []  []  [] []  [] [] []  []    [] []    []
[]     []  [] [][][] [][][] []  [] []    [] []    []

[][][] [][][]     [][][] []     [][][] [][][] [][][] []   [] [][][]
  []   []         []  [] []     []  [] []       []   []]  [] []
  []   [][][]     []     []     []  [] [][][]   []   [][] [] [] []
  []       []     []  [] []     []  []     []   []   [] [][] []  []
[][][] [][][]     [][][] [][][] [][][] [][][] [][][] []  [[] [][][]

[][][]  []     [] [][][]   [] [] []
[]   []  []  []   []       [] [] []
[]  []     []     [][][]   [] [] []
[]   []    []     []
[][][]     []     [][][]   [] [] []
```

2. Deadlock Recovery and Management System

1. Deadlock Detection and Avoidance using Banker’s Algorithm

```
#include <iostream>
#include <vector>
#include <mutex>
#include <thread>
#include <chrono>
#include <algorithm>

using namespace std;

const int NUM_RESOURCES = 3;
const int NUM_PROCESSES = 5;

vector<int> available(NUM_RESOURCES);
vector<vector<int>> maximum(NUM_PROCESSES, vector<int>(NUM_RESOURCES));
vector<vector<int>> allocation(NUM_PROCESSES, vector<int>(NUM_RESOURCES));
```

```
vector<vector<int>> need(NUM_PROCESSES, vector<int>(NUM_RESOURCES));
vector<bool> resourceGranted(NUM_PROCESSES, false); // Flag to track if
resources are already granted

mutex mtx;

bool isSafe(int processId) {
    vector<int> work = available;
    vector<bool> finish(NUM_PROCESSES, false);

    for (int i = 0; i < NUM_RESOURCES; ++i) {
        work[i] -= allocation[processId][i];
        need[processId][i] = 0;
    }
    finish[processId] = true;

    bool canProceed = true;
    while (canProceed) {
        canProceed = false;
        for (int i = 0; i < NUM_PROCESSES; ++i) {
            if (!finish[i]) {
                bool canAllocate = true;
                for (int j = 0; j < NUM_RESOURCES; ++j) {
                    if (need[i][j] > work[j]) {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate) {
                    canProceed = true;
                    finish[i] = true;
                    for (int j = 0; j < NUM_RESOURCES; ++j) {
                        work[j] += allocation[i][j];
                        need[i][j] = 0;
                    }
                }
            }
        }
    }
    return all_of(finish.begin(), finish.end(), [](bool b) { return b; });
}

void releaseResources(int processId) {
    lock_guard<mutex> lock(mtx);
    for (int i = 0; i < NUM_RESOURCES; ++i) {
        available[i] += allocation[processId][i];
        allocation[processId][i] = 0;
        need[processId][i] = 0;
    }
    resourceGranted[processId] = false; // Reset the flag when releasing
resources
}

void requestResources(int processId, const vector<int>& request) {
    lock_guard<mutex> lock(mtx);
    if (resourceGranted[processId]) {
        cout << "\033[1;32mProcess " << processId << " has already been
granted resources\033[0m\n";
        return;
    }
    for (int i = 0; i < NUM_RESOURCES; ++i) {
        if (request[i] > need[processId][i] || request[i] > available[i]) {
            cout << "\033[1;33mProcess " << processId << " is
waiting...\033[0m\n";
            return;
        }
    }

    for (int i = 0; i < NUM_RESOURCES; ++i) {
        available[i] -= request[i];
        allocation[processId][i] += request[i];
        need[processId][i] -= request[i];
    }

    if (isSafe(processId)) {
        cout << "\033[1;32mProcess " << processId << " has been granted
resources\033[0m\n";
        resourceGranted[processId] = true; // Set the flag when resources
are granted
    }
    else {
        cout << "\033[1;31mProcess " << processId << " must wait, deadlock
might occur\033[0m\n";
        // Rollback resources
        for (int i = 0; i < NUM_RESOURCES; ++i) {
            available[i] += request[i];
            allocation[processId][i] -= request[i];
            need[processId][i] += request[i];
        }
    }
}

void processThread(int processId) {
    vector<int> request(NUM_RESOURCES);
    while (true) {
        for (int i = 0; i < NUM_RESOURCES; ++i) {
            request[i] = rand() % (maximum[processId][i] -
allocation[processId][i] + 1);
        }

        requestResources(processId, request);

        // Skip further requests if resources have been granted
        if (resourceGranted[processId]) {
            return;
        }

        this_thread::sleep_for(chrono::milliseconds(2000));

        releaseResources(processId);

        this_thread::sleep_for(chrono::milliseconds(1000));
    }
}

int main() {
    // Initialize available resources
    for (int i = 0; i < NUM_RESOURCES; ++i) {
        cout << "\033[1;34mEnter available instances of resource " << i <<
": \033[0m";
        cin >> available[i];
    }

    // Initialize maximum resources for each process
    for (int i = 0; i < NUM_PROCESSES; ++i) {
        cout << "\033[1;34mEnter maximum instances of resource for process
" << i << ":\033[0m\n";
        for (int j = 0; j < NUM_RESOURCES; ++j) {
            cout << "\033[1;34mResource " << j << ": \033[0m";
            cin >> maximum[i][j];
            need[i][j] = maximum[i][j];
        }
    }

    vector<thread> threads;
    for (int i = 0; i < NUM_PROCESSES; ++i) {
        threads.emplace_back(processThread, i);
    }

    for (auto& t : threads) {
        t.join();
    }

    return 0;
}
```

2. Customized Strategic Deadlock Recovery Algorithm

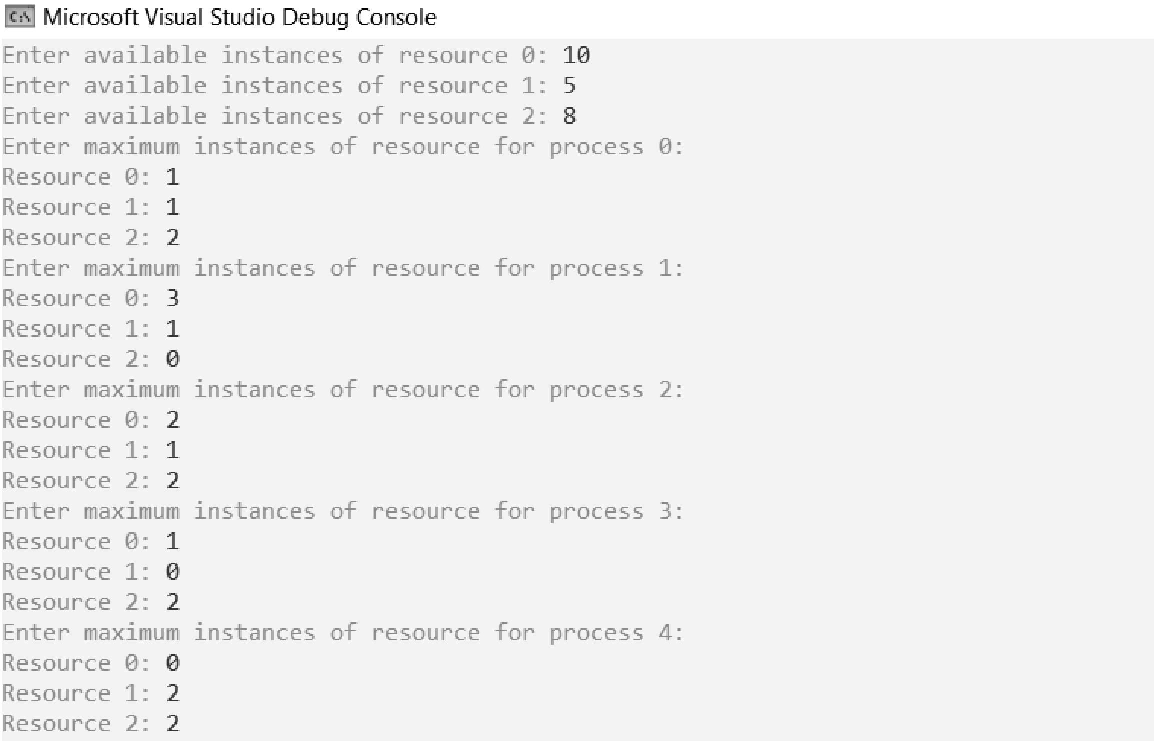
```
#include <iostream>
#include <thread>
#include <mutex>
#include <vector>
#include <chrono>
#include <atomic>
#include <condition_variable>
#include <random>

using namespace std;

// Constants
constexpr int NUM_THREADS = 5; // Updated to 5 threads
constexpr int NUM_RESOURCES = 3;
constexpr int MAX_SLEEP = 500; // milliseconds
constexpr int MAX_ITERATIONS = 10; // Terminate after 10 iterations

// Resource Manager class
class ResourceManager {
private:
    mutex mtx;
    condition_variable cv;
    vector<bool> resources;

public:
    ResourceManager() : resources(NUM_RESOURCES, false) {}

    bool requestResource(int threadId) {
        unique_lock<mutex> lock(mtx);
        for (int i = 0; i < NUM_RESOURCES; ++i) {
            if (!resources[i]) {
                resources[i] = true;
                cout << "\033[1;32mThread " << threadId << "\033[0m
acquired resource " << i << endl;
                return true;
            }
        }
        return false; // No available resources
    }

    void releaseResource(int resourceIdx) {
        lock_guard<mutex> lock(mtx);
        resources[resourceIdx] = false;
        cv.notify_all();
    }

    bool isDeadlocked() const {
        // Simplified deadlock detection logic
        return false;
    }

    void recoverFromDeadlock() {
        // Preempt resources or roll back thread states based on predefined
criteria
        // In this example, we will simply preempt resources from threads
in a random manner
        random_device rd;
        mt19937 gen(rd());
        uniform_int_distribution<int> distr(0, NUM_RESOURCES - 1);

        for (int i = 0; i < NUM_RESOURCES / 2; ++i) {
            int resourceIdx = distr(gen);
            resources[resourceIdx] = false;
            cout << "\n\033[1;31mPreempted resource " << resourceIdx <<
```

```
"\033[0m\n";
        }
    }
};

// Thread function
void threadFunction(int id, ResourceManager& resourceManager) {
    int iterations = 0;
    while (iterations < MAX_ITERATIONS) {
        if (resourceManager.requestResource(id)) {
            // Critical section
            this_thread::sleep_for(chrono::milliseconds(rand() %
MAX_SLEEP));
            resourceManager.releaseResource(rand() % NUM_RESOURCES);
            iterations++;
        }
        else {
            // Deadlock detected, initiate recovery mechanism
            cout << "\033[1;33mDeadlock detected by Thread " << id <<
"\033[0m\n";
            resourceManager.recoverFromDeadlock();
        }
    }
}

int main() {
    srand(time(nullptr));
    ResourceManager resourceManager;

    cout << "\033[1;36m========== Resource Management System
==========\033[0m\n\n";

    vector<thread> threads;
    for (int i = 0; i < NUM_THREADS; ++i) {
        threads.emplace_back(thread(threadFunction, i,
ref(resourceManager)));
    }

    for (auto& t : threads) {
        t.join();
    }

    return 0;
}
```

3. Deadlock Prevention Using Semaphores, Mutex, Condition variable, Multiple threads

```
#include <iostream>
#include <thread>
#include <mutex>
#include <vector>
#include <chrono>
#include <condition_variable>
#include <random>
#include <algorithm>

using namespace std;

// Constants
constexpr int NUM_THREADS = 5;
constexpr int NUM_RESOURCES = 3;
constexpr int MAX_SLEEP = 500; // milliseconds

// Semaphore class
class Semaphore {
private:
    mutex mtx;
    condition_variable cv;
    int count;

public:
    Semaphore(int initialCount = 0) : count(initialCount) {}

    void notify() {
        lock_guard<mutex> lock(mtx);
        count++;
        cv.notify_one();
    }

    void wait() {
        unique_lock<mutex> lock(mtx);
        while (count <= 0) {
            cv.wait(lock);
        }
        count--;
    }
};

// Resource Manager class
class ResourceManager {
private:
    vector<bool> resources;
    mutex mtx;

public:
    ResourceManager() : resources(NUM_RESOURCES, false) {}

    int acquireResource() {
        lock_guard<mutex> lock(mtx);
        for (int i = 0; i < NUM_RESOURCES; ++i) {
            if (!resources[i]) {
                resources[i] = true;
                return i;
            }
        }
        return -1; // No available resources
    }

    void releaseResource(int resourceIdx) {
        lock_guard<mutex> lock(mtx);
        resources[resourceIdx] = false;
    }
};

// Global variables
ResourceManager resourceManager;
Semaphore semaphore(NUM_RESOURCES);
mutex cout_mutex; // Declaration of cout_mutex

// Thread function
void threadFunction(int id) {
    while (true) {
        semaphore.wait(); // Wait until a resource is available
        int resourceIdx = resourceManager.acquireResource();
        if (resourceIdx != -1) {
            // Resource acquired, perform some work
            {
                lock_guard<mutex> lock(cout_mutex); // Lock cout to ensure
thread-safe output
                cout << "\033[1;32m[Thread " << id << "]\033[0m Acquired
resource " << resourceIdx << endl;
            }
            this_thread::sleep_for(chrono::milliseconds(rand() %
MAX_SLEEP));
            // Release the resource
            resourceManager.releaseResource(resourceIdx);
            semaphore.notify(); // Notify that a resource is available
        }
    }
}

int main() {
    srand(static_cast<unsigned int>(time(nullptr)));

    cout << "\n\033[1;36m========== Resource Management System
==========\033[0m" << endl<< endl;

    vector<thread> threads;
    for (int i = 0; i < NUM_THREADS; ++i) {
        threads.emplace_back(thread(threadFunction, i));
    }

    cout << "\033[1;36mThreads have been initialized.\033[0m" << endl<<
endl;

    for (auto& t : threads) {
        t.detach(); // Detach threads to allow them to run independently
    }

    cout << "\033[1;36mMain thread sleeps to allow child threads to
execute.\033[0m" << endl;
    cout <<
"\n\033[1;36m=================================================\033[0m" <<
endl;

    // Main thread sleeps to allow child threads to execute
    this_thread::sleep_for(chrono::seconds(10));

    return 0;
}
```

INPUT

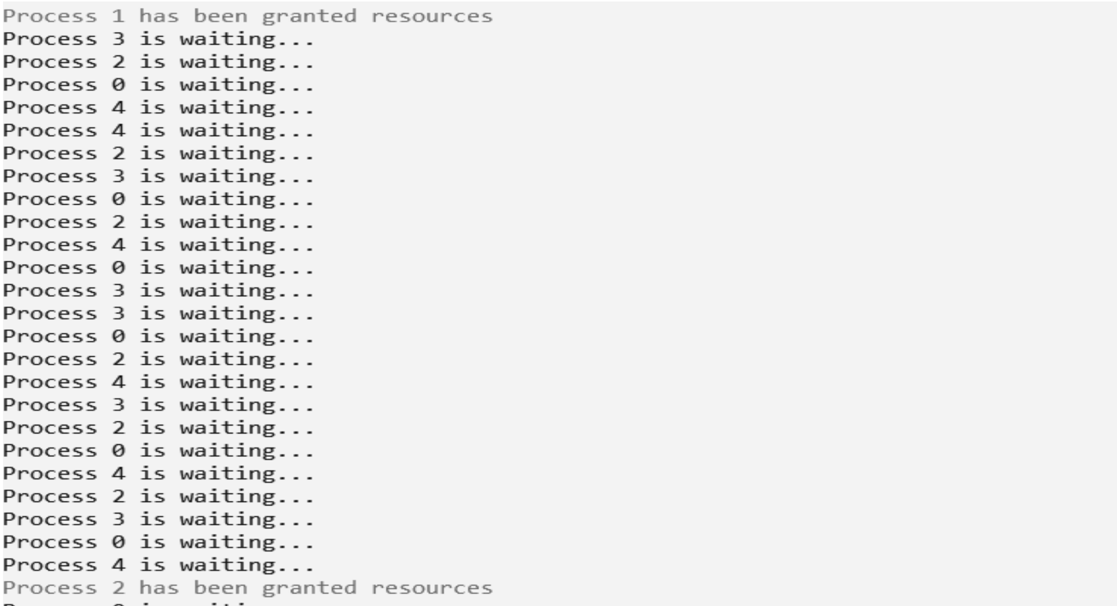


Microsoft Visual Studio Debug Console

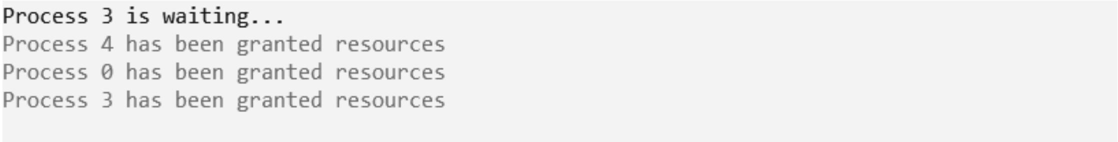
```
Enter available instances of resource 0: 10
Enter available instances of resource 1: 5
Enter available instances of resource 2: 8
Enter maximum instances of resource for process 0:
Resource 0: 1
Resource 1: 1
Resource 2: 2
Enter maximum instances of resource for process 1:
Resource 0: 3
Resource 1: 1
Resource 2: 0
Enter maximum instances of resource for process 2:
Resource 0: 2
Resource 1: 1
Resource 2: 2
Enter maximum instances of resource for process 3:
Resource 0: 1
Resource 1: 0
Resource 2: 2
Enter maximum instances of resource for process 4:
Resource 0: 0
Resource 1: 2
Resource 2: 2
```

OUTPUT

Program 1 :

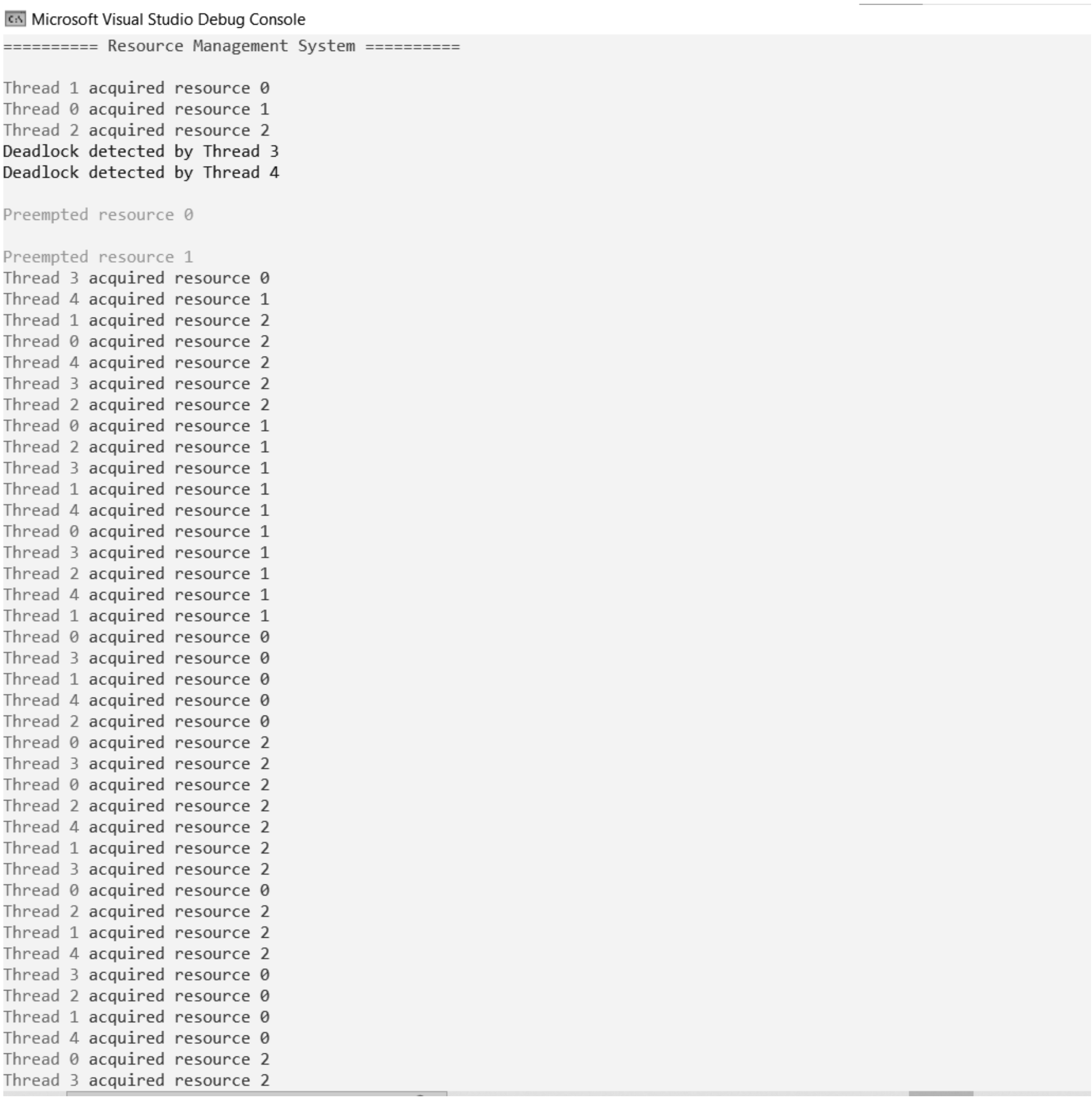


```
Process 1 has been granted resources
Process 3 is waiting...
Process 2 is waiting...
Process 0 is waiting...
Process 4 is waiting...
Process 4 is waiting...
Process 2 is waiting...
Process 3 is waiting...
Process 0 is waiting...
Process 2 is waiting...
Process 4 is waiting...
Process 0 is waiting...
Process 3 is waiting...
Process 3 is waiting...
Process 0 is waiting...
Process 2 is waiting...
Process 4 is waiting...
Process 3 is waiting...
Process 2 is waiting...
Process 0 is waiting...
Process 4 is waiting...
Process 2 is waiting...
Process 3 is waiting...
Process 0 is waiting...
Process 4 is waiting...
Process 2 has been granted resources
```

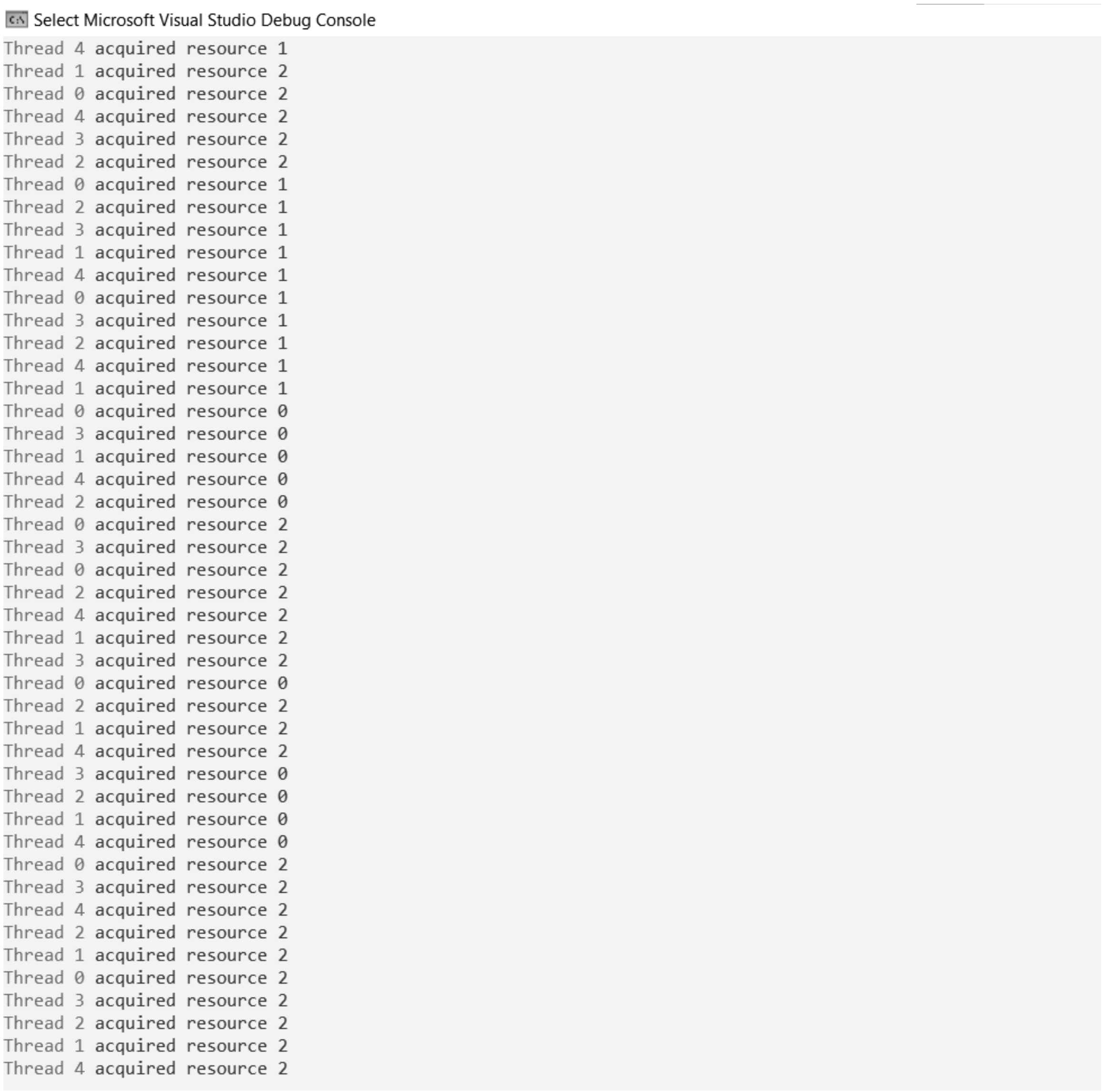


```
Process 3 is waiting...
Process 4 has been granted resources
Process 0 has been granted resources
Process 3 has been granted resources
```

Program 2 (Test Case - 1) :

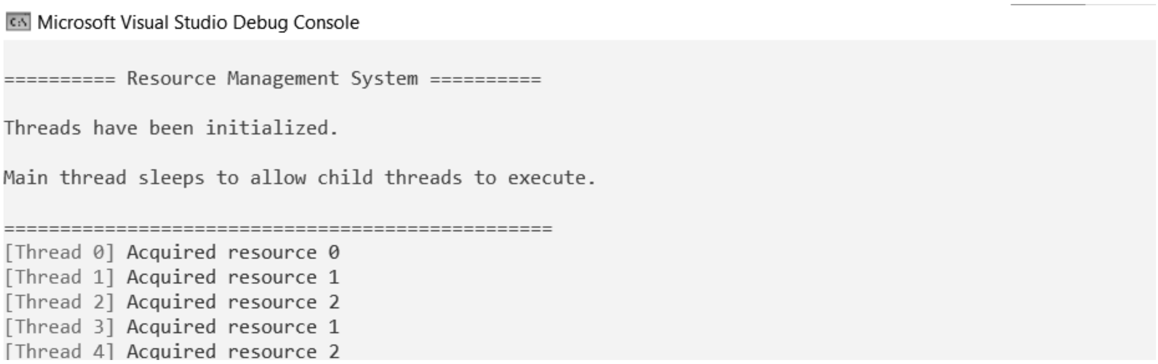


```
========== Resource Management System ==========

Thread 1 acquired resource 0
Thread 0 acquired resource 1
Thread 2 acquired resource 2
Deadlock detected by Thread 3
Deadlock detected by Thread 4

Preempted resource 0

Preempted resource 1
Thread 3 acquired resource 0
Thread 4 acquired resource 1
Thread 1 acquired resource 2
Thread 0 acquired resource 2
Thread 4 acquired resource 2
Thread 3 acquired resource 2
Thread 2 acquired resource 2
Thread 0 acquired resource 1
Thread 2 acquired resource 1
Thread 3 acquired resource 1
Thread 1 acquired resource 1
Thread 4 acquired resource 1
Thread 0 acquired resource 1
Thread 3 acquired resource 1
Thread 2 acquired resource 1
Thread 4 acquired resource 1
Thread 1 acquired resource 1
Thread 0 acquired resource 0
Thread 3 acquired resource 0
Thread 1 acquired resource 0
Thread 4 acquired resource 0
Thread 2 acquired resource 0
Thread 0 acquired resource 2
Thread 3 acquired resource 2
Thread 0 acquired resource 2
Thread 2 acquired resource 2
Thread 4 acquired resource 2
Thread 1 acquired resource 2
Thread 3 acquired resource 2
Thread 0 acquired resource 0
Thread 2 acquired resource 2
Thread 1 acquired resource 2
Thread 4 acquired resource 2
Thread 3 acquired resource 0
Thread 2 acquired resource 0
Thread 1 acquired resource 0
Thread 4 acquired resource 0
Thread 0 acquired resource 2
Thread 3 acquired resource 2
```

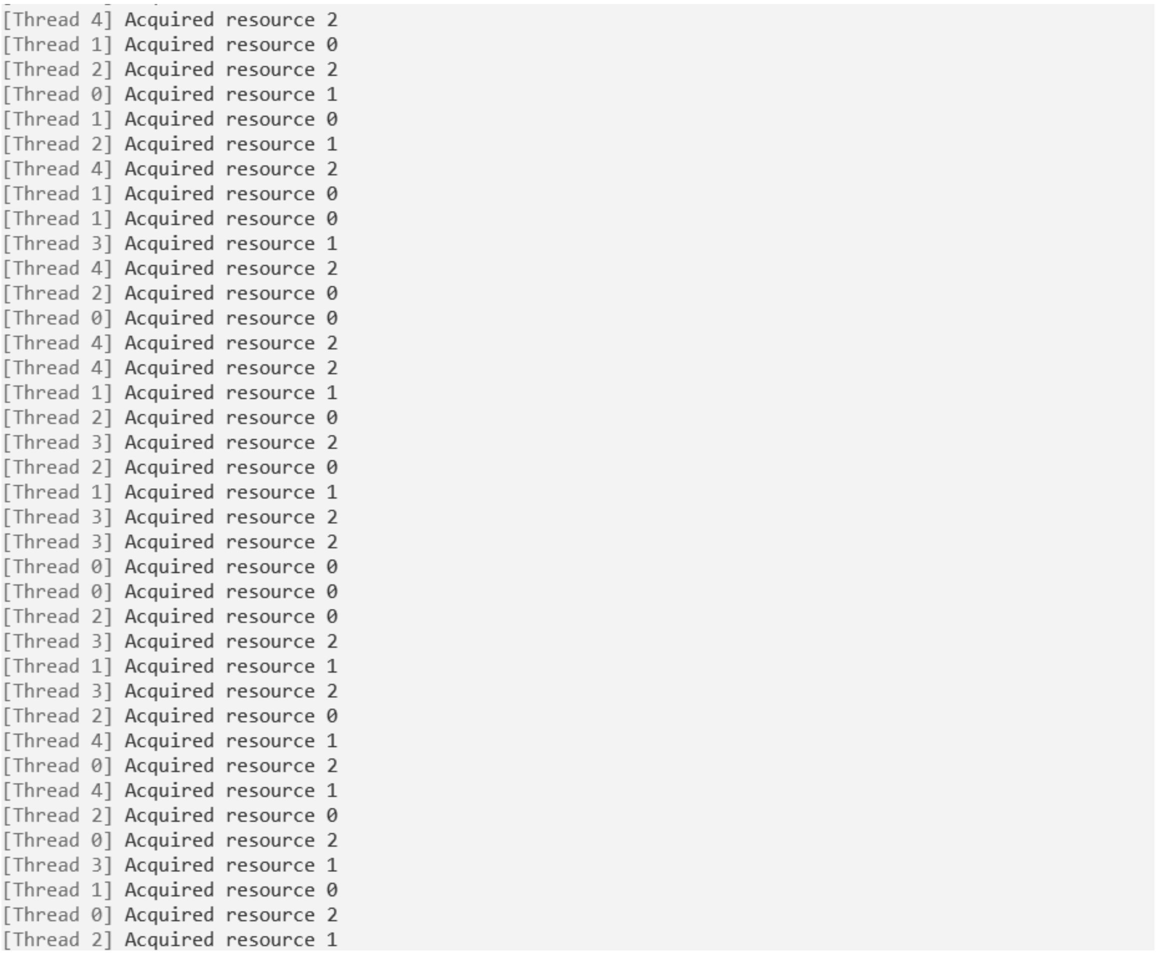


```
Thread 4 acquired resource 1
Thread 1 acquired resource 2
Thread 0 acquired resource 2
Thread 4 acquired resource 2
Thread 3 acquired resource 2
Thread 2 acquired resource 2
Thread 0 acquired resource 1
Thread 2 acquired resource 1
Thread 3 acquired resource 1
Thread 1 acquired resource 1
Thread 4 acquired resource 1
Thread 0 acquired resource 1
Thread 3 acquired resource 1
Thread 2 acquired resource 1
Thread 4 acquired resource 1
Thread 1 acquired resource 1
Thread 0 acquired resource 0
Thread 3 acquired resource 0
Thread 1 acquired resource 0
Thread 4 acquired resource 0
Thread 2 acquired resource 0
Thread 0 acquired resource 2
Thread 3 acquired resource 2
Thread 0 acquired resource 2
Thread 2 acquired resource 2
Thread 4 acquired resource 2
Thread 1 acquired resource 2
Thread 3 acquired resource 2
Thread 0 acquired resource 0
Thread 2 acquired resource 2
Thread 1 acquired resource 2
Thread 4 acquired resource 2
Thread 3 acquired resource 0
Thread 2 acquired resource 0
Thread 1 acquired resource 0
Thread 4 acquired resource 0
Thread 0 acquired resource 2
Thread 3 acquired resource 2
Thread 4 acquired resource 2
Thread 2 acquired resource 2
Thread 1 acquired resource 2
Thread 0 acquired resource 2
Thread 3 acquired resource 2
Thread 2 acquired resource 2
Thread 1 acquired resource 2
Thread 4 acquired resource 2
```

Program 3 (Test Case - 2) :



```
========== Resource Management System ==========

Threads have been initialized.

Main thread sleeps to allow child threads to execute.

=================================================
[Thread 0] Acquired resource 0
[Thread 1] Acquired resource 1
[Thread 2] Acquired resource 2
[Thread 3] Acquired resource 1
[Thread 4] Acquired resource 2
```



```
[Thread 4] Acquired resource 2
[Thread 1] Acquired resource 0
[Thread 2] Acquired resource 2
[Thread 0] Acquired resource 1
[Thread 1] Acquired resource 0
[Thread 2] Acquired resource 1
[Thread 4] Acquired resource 2
[Thread 1] Acquired resource 0
[Thread 1] Acquired resource 0
[Thread 3] Acquired resource 1
[Thread 4] Acquired resource 2
[Thread 2] Acquired resource 0
[Thread 0] Acquired resource 0
[Thread 4] Acquired resource 2
[Thread 4] Acquired resource 2
[Thread 1] Acquired resource 1
[Thread 2] Acquired resource 0
[Thread 3] Acquired resource 2
[Thread 2] Acquired resource 0
[Thread 1] Acquired resource 1
[Thread 3] Acquired resource 2
[Thread 3] Acquired resource 2
[Thread 0] Acquired resource 0
[Thread 0] Acquired resource 0
[Thread 2] Acquired resource 0
[Thread 3] Acquired resource 2
[Thread 1] Acquired resource 1
[Thread 3] Acquired resource 2
[Thread 2] Acquired resource 0
[Thread 4] Acquired resource 1
[Thread 0] Acquired resource 2
[Thread 4] Acquired resource 1
[Thread 2] Acquired resource 0
[Thread 0] Acquired resource 2
[Thread 3] Acquired resource 1
[Thread 1] Acquired resource 0
[Thread 0] Acquired resource 2
[Thread 2] Acquired resource 1
```