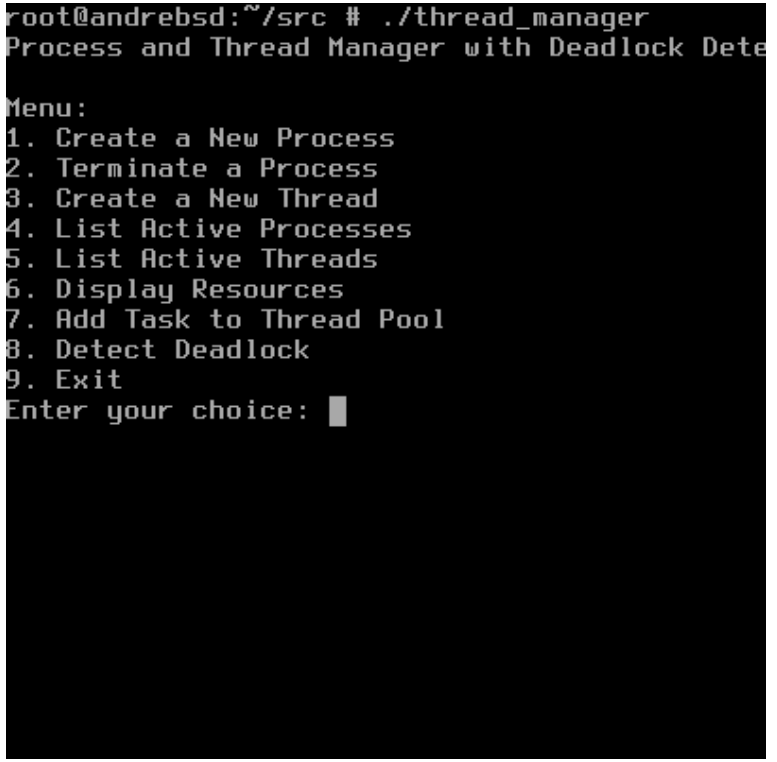
Process and Thread Manager

MATTHEW AKINMOLAYAN
ANDRE' HERRON
TOBILOBA AYODEJI
TOBILOBA IFENIKALO

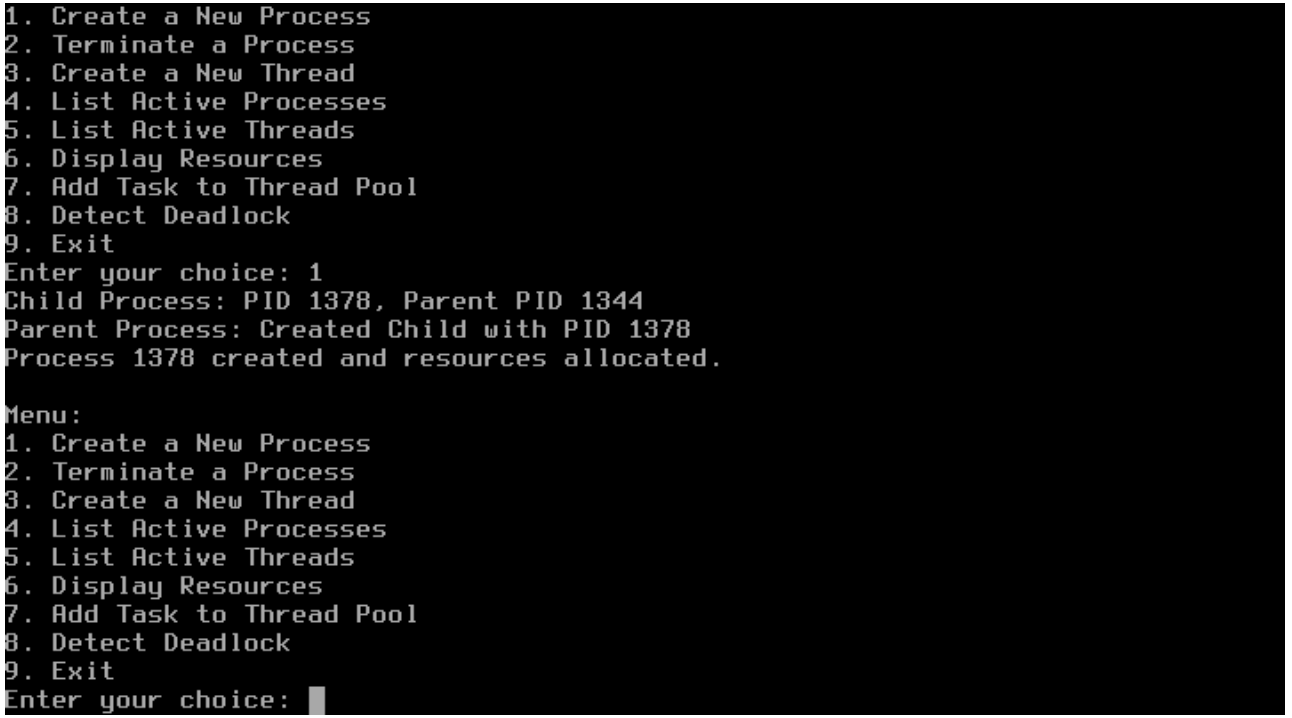
Project Overview

- This project implements a Process and Thread Manager program.
- Allows users to create and manage multiple processes and threads.
- Implements process resource allocation and a thread pool.
- Identifies potential deadlocks in the system



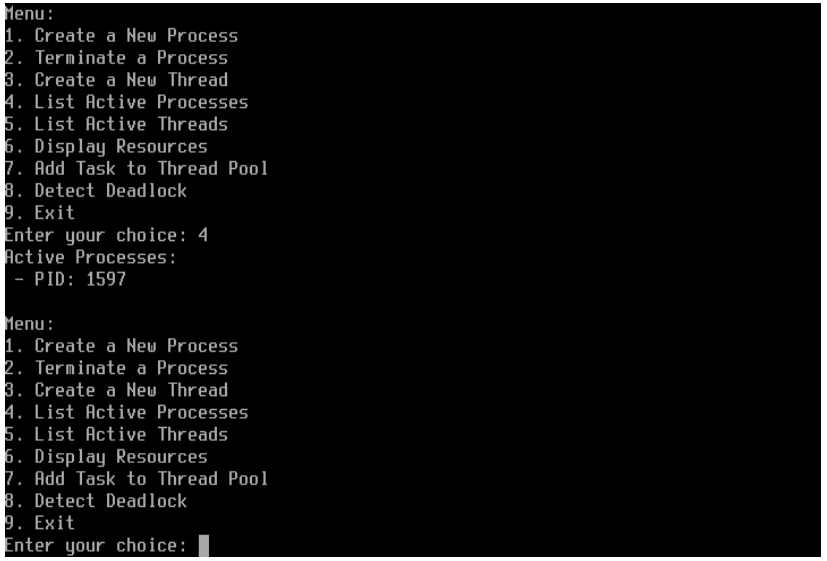
Process Creation

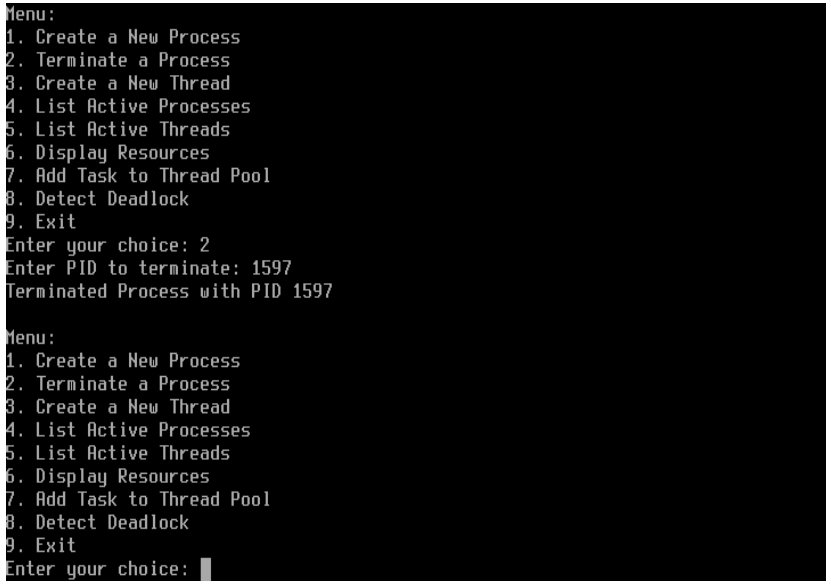
- Calls fork() to create a child process of the programs main process.
- The child's PID is added to the activeProcesses list.
- Simulates work by using sleep() in a loop.
- It requests a random number of resources between 0-5.

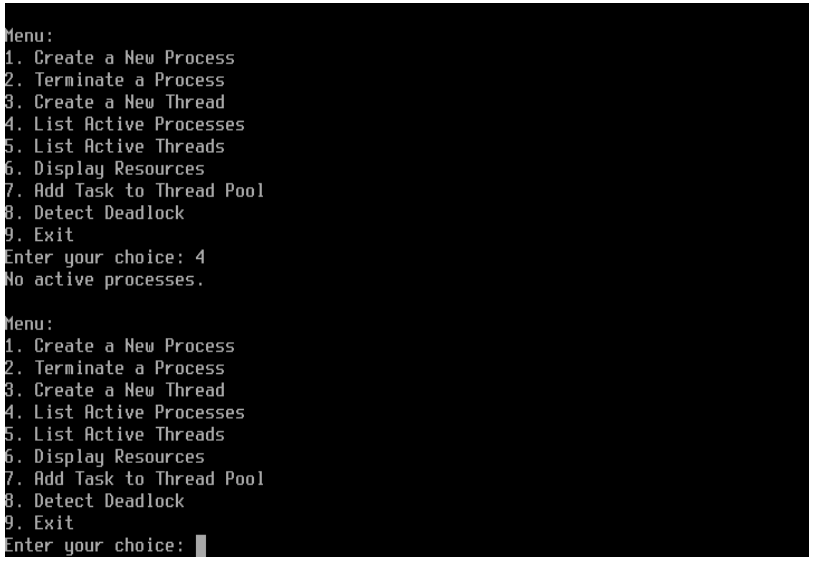


Process Termination

- Terminates a specified process.
- Deallocates the resources held.

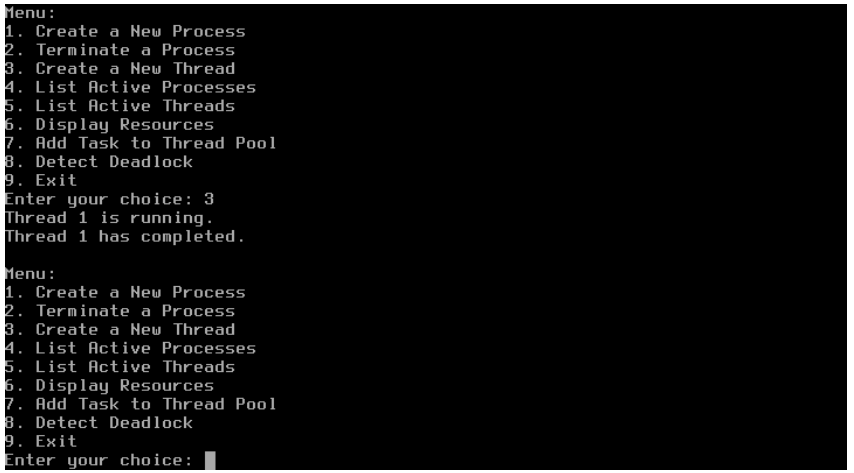


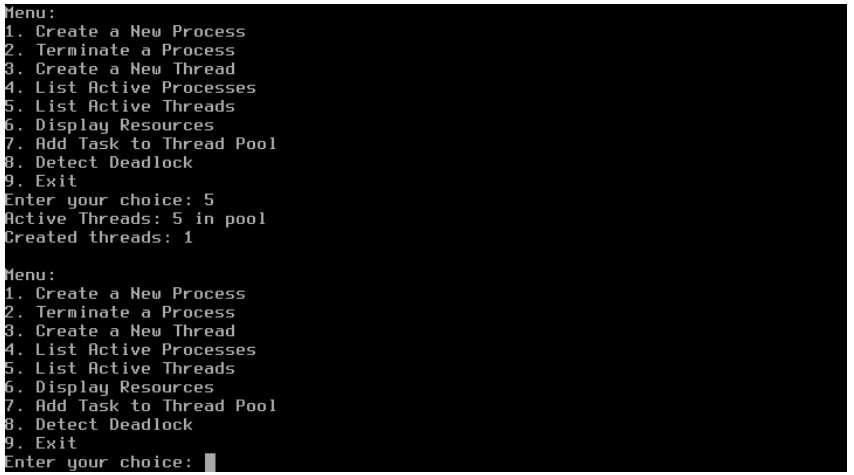




Thread Management

- 3 functions: createThread(), listThreads(), and joinThreads().
- For thread synchronization we used lock_guard to protect thread access to the activeThreads and threadPool vectors.
- CreateThread() generates a new thread ID and adds the thread to activeThreads.
- ListThreads() prints the number of threads that were created along with the threads in the thread pool.
- JoinThreads() joins all joinable() threads in activeThreads.





Deadlock manager

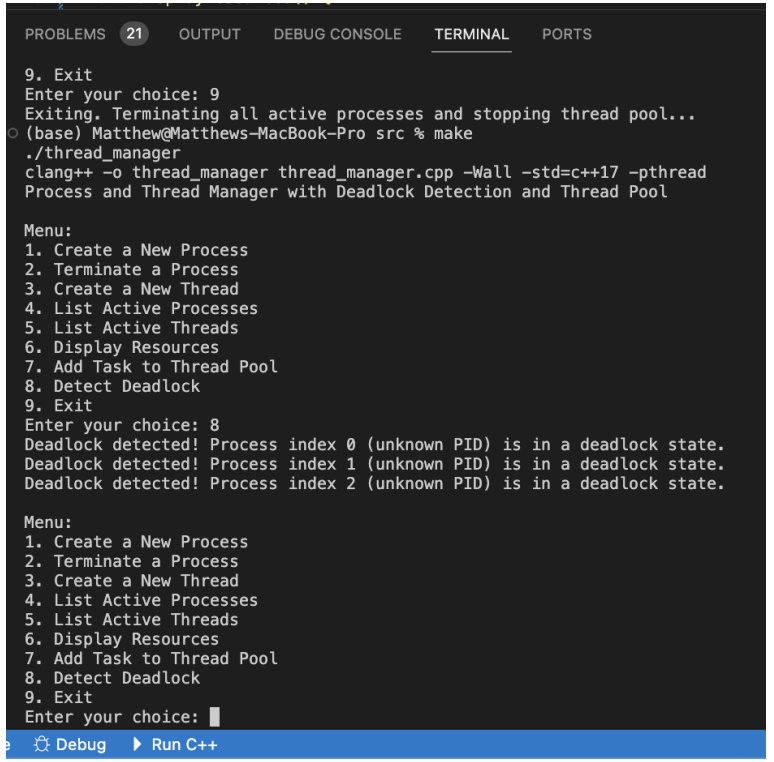
Purpose: To identify and resolve situations where processes are stuck waiting for resources held by other processes.

Methodology:

- Uses a variation of the **Banker's Algorithm**.
- Tracks resource **Allocation**, **Requests**, and **Availability** using matrices.
- Analyzes if any processes are unable to complete due to unfulfilled requests.

Output:

- Reports processes in a **deadlock state**, if any.
- Otherwise, confirms that no deadlock is detected.



Priority Inheritance

Purpose: Prevents priority inversion by temporarily elevating the priority of a lower-priority task holding a shared resource

Methodology:

- A **low-priority task** locks a resource
- A **high-priority task** is blocked, waiting for the resource
- **Priority inheritance** occurs, where the low-priority task inherits the high priority
- After releasing the resource, the low-priority task restores its original priority

```
void priorityInheritanceProtocol(int taskPriority, int &sharedPriority, mutex &m) {
    lock_guard<mutex> lock(m);
    if (taskPriority > sharedPriority) {
        sharedPriority = taskPriority;
    }
}
```

System Resource Tracking

Purpose: Facilitates real-time monitoring and management of system resources allocated to processes and threads. Ensures transparency in resource usage to prevent deadlocks and resource contention.

Methodology:

- Displays available resources, allocation, and request matrices to track usage and detect deadlocks.

Key Metrics Displayed:

Available Resources: Total resources remaining for allocation.

Allocation Matrix: Resources currently allocated to each active process.

Request Matrix: Outstanding requests for resources by active processes.

```cpp
void displayResources() {
    cout << "\nResource Allocation Tracking:\n";
    cout << "Available Resources: ";
    for (size_t i = 0; i < available.size(); ++i) {
        int temp = available[i];
        cout << temp << " ";
    }
    cout << "\n";
    if (allocation.empty()) {
        cout << "No active processes to display resource allocation.\n";
        return;
    }
    cout << "Allocation Matrix:\n";
    for (size_t i = 0; i < allocation.size(); ++i) {
        cout << "Process " << activeProcesses[i] << ": ";
        for (size_t j = 0; j < allocation[i].size(); ++j) {
            int temp = allocation[i][j];
            cout << temp << " ";
        }
        cout << "\n";
    }
    cout << "Request Matrix:\n";
    for (size_t i = 0; i < request.size(); ++i) {
        cout << "Process " << activeProcesses[i] << ": ";
        for (size_t j = 0; j < request[i].size(); ++j) {
            int temp = request[i][j];
            cout << temp << " ";
        }
        cout << "\n";
    }
}
```

```
Menu:
1. Create a New Process
2. Terminate a Process
3. Create a New Thread
4. List Active Processes
5. List Active Threads
6. Display Resources
7. Add Task to Thread Pool
8. Detect Deadlock
9. Exit
Enter your choice: 6

Resource Allocation Tracking:
Available Resources: 4 3 1
Allocation Matrix:
Process 5524: 5 4 3
Process 5542: 0 2 5
Request Matrix:
Process 5524: 0 0 0
Process 5542: 0 0 0
```

Shared Memory

Purpose: Enables quick data transfer across processes by permitting several processes to access the same memory address.

Methodology:

- **Allocate a shared memory** segment
- **Attach processes** to the shared memory
- Effective inter-process communication is made possible by read/write operations being carried out directly in the shared memory.