



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

```
root@andrebsd:~/src # ./thread_manager
Process and Thread Manager with Deadlock Detection

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: █
```

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.

```
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: 1
Child Process: PID 1378, Parent PID 1344
Parent Process: Created Child with PID 1378
Process 1378 created and resources allocated.

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: █
```

Process Termination

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

```
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: 4
Active Processes:
- PID: 1597
```

```
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: █
```

```
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: 2
Enter PID to terminate: 1597
Terminated Process with PID 1597
```

```
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: █
```

```
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: 4
No active processes.
```

```
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: █
```

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`.

```
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: 3
Thread 1 is running.
Thread 1 has completed.
```

```
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: █
```

```
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: 5
Active Threads: 5 in pool
Created threads: 1
```

```
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: █
```

Deadlock manager

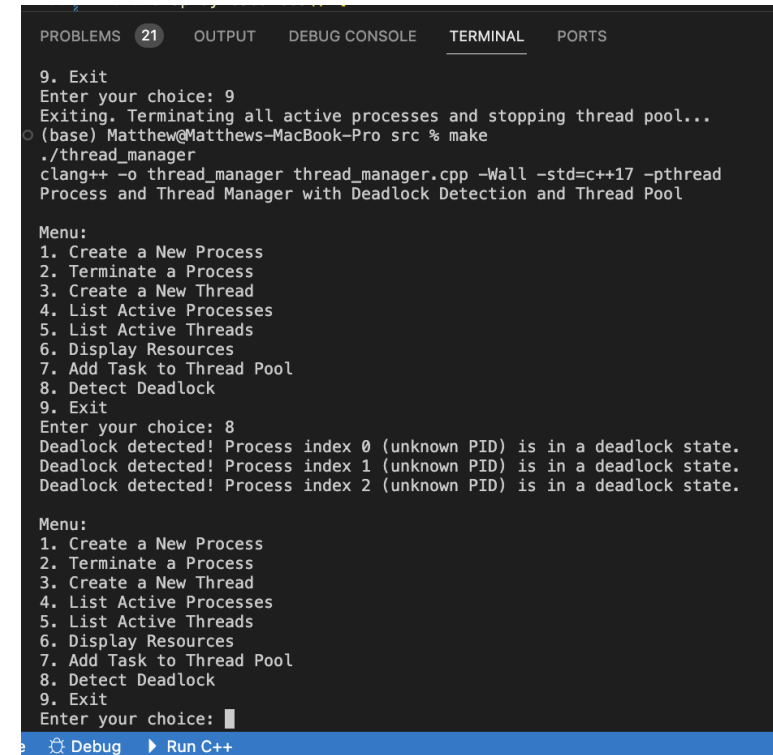
Purpose: To identify and resolve situations where processes are stuck waiting for resources held by other processes.

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.



```
PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL PORTS

9. Exit
Enter your choice: 9
Exiting. Terminating all active processes and stopping thread pool...
(base) Matthew@Matthews-MacBook-Pro src % make
./thread_manager
clang++ -o thread_manager thread_manager.cpp -Wall -std=c++17 -pthread
Process and Thread Manager with Deadlock Detection and Thread Pool

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: 8
Deadlock detected! Process index 0 (unknown PID) is in a deadlock state.
Deadlock detected! Process index 1 (unknown PID) is in a deadlock state.
Deadlock detected! Process index 2 (unknown PID) is in a deadlock state.

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: █
```

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;  
    }  
}
```

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.