**Phase 2 Implementation Of Your Operating System (OS)**

# Operating Systems Phase 2 - Multiprogramming OS Implementation Guide

## 1. Introduction

The Phase 2 project implements a Multiprogramming Operating System (MOS) with the following key features:

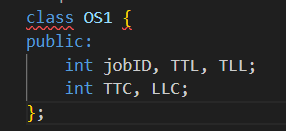
* Program error handling through interrupts (PI)
* Memory management through paging
* Time and line limit monitoring
* Job separation and error handling

### System Specifications

* Main Memory: 100 words × 4 bytes
* Block Size: 10 words × 4 bytes
* Input Device: Card reader (40 bytes per card)
* Output Device: Line printer (40 bytes per line)

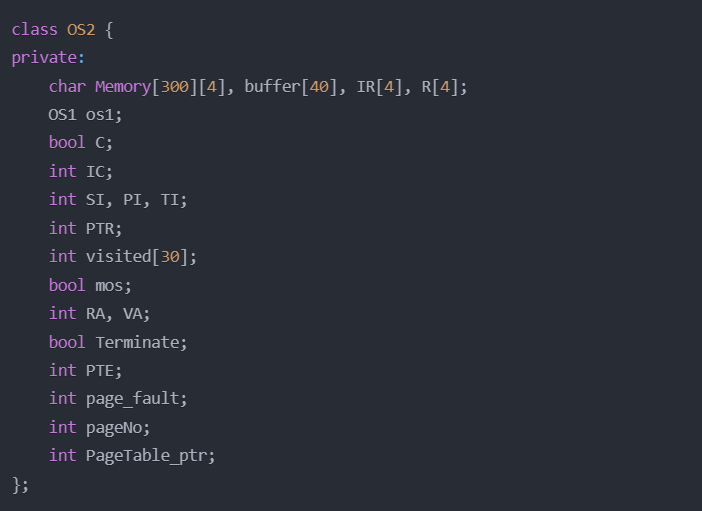
## 2. Core Data Structures

### 2.1 Process Control Block (PCB)



Key Variables:

* jobID: Unique identifier for each job
* TTL: Total Time Limit allowed for the job
* TLL: Total Line Limit for output
* TTC: Total Time Counter (tracks execution time)
* LLC: Line Limit Counter (tracks output lines)

**2.2 Operating System Core Class:-  
**

Key Components:

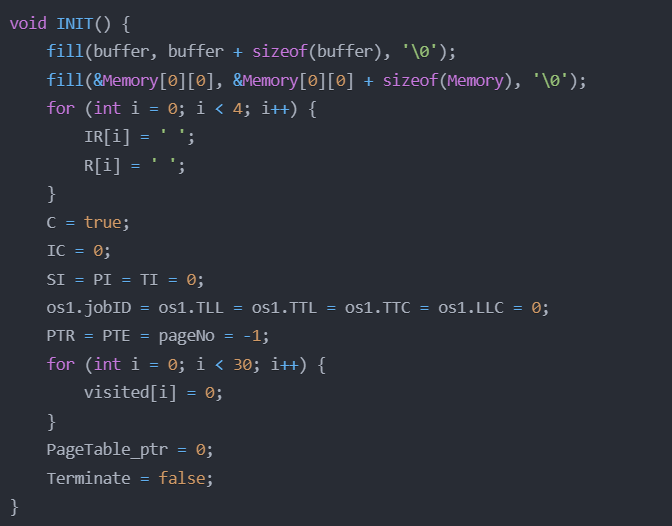
* **Memory[300][4]:** Main memory simulation (300 blocks × 4 bytes)
* **buffer[40]:** Input/output buffer
* **IR[4]:** Instruction Register
* **R[4]:** General Purpose Register
* **C:** Toggle for comparison operations
* **IC:** Instruction Counter
* **SI/PI/TI:** Interrupt flags
* **PTR:** Page Table Register
* **VA/RA:** Virtual/Real Address

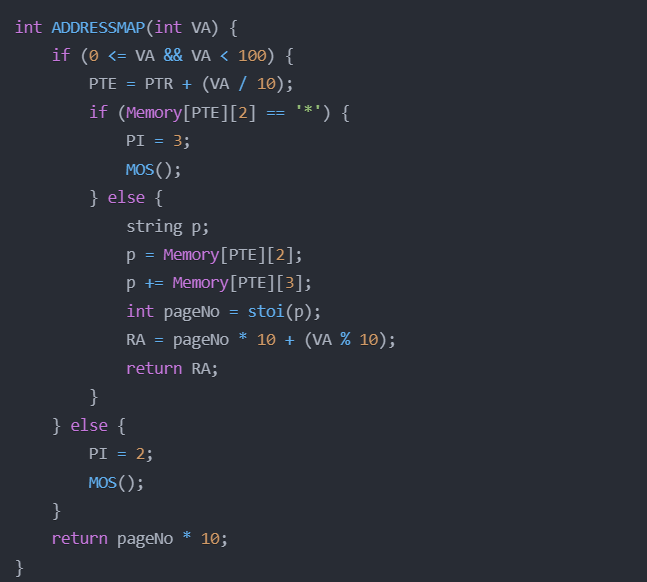
## 3. Memory Management

### 3.1 Page Table Implementation

Memory Management Features:

* Random allocation of memory blocks
* Page table stored in real memory
* Virtual to physical address mapping
* Page fault handling



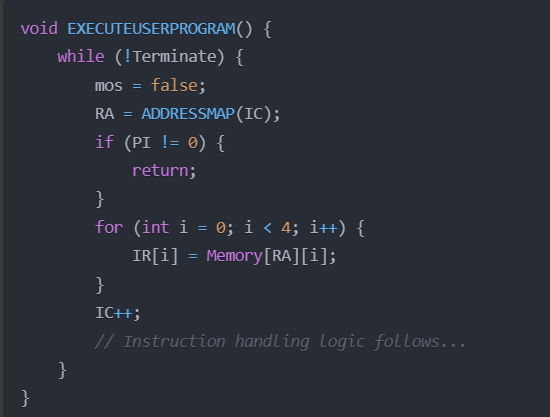
**3.2 Address Mapping:-  
**

## 4. Instruction Set Implementation

### 4.1 Supported Instructions

* GD: Get Data from input
* PD: Print Data to output
* LR: Load Register
* SR: Store Register
* CR: Compare Register
* BT: Branch on Toggle
* H: Halt

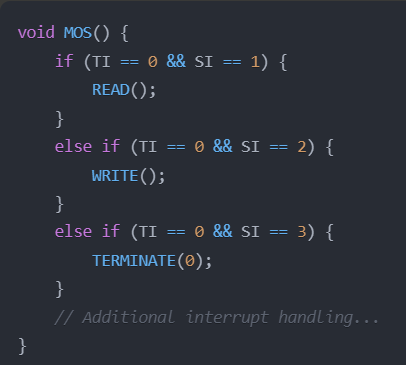
### 4.2 Instruction Execution

****

## 5. Interrupt Handling

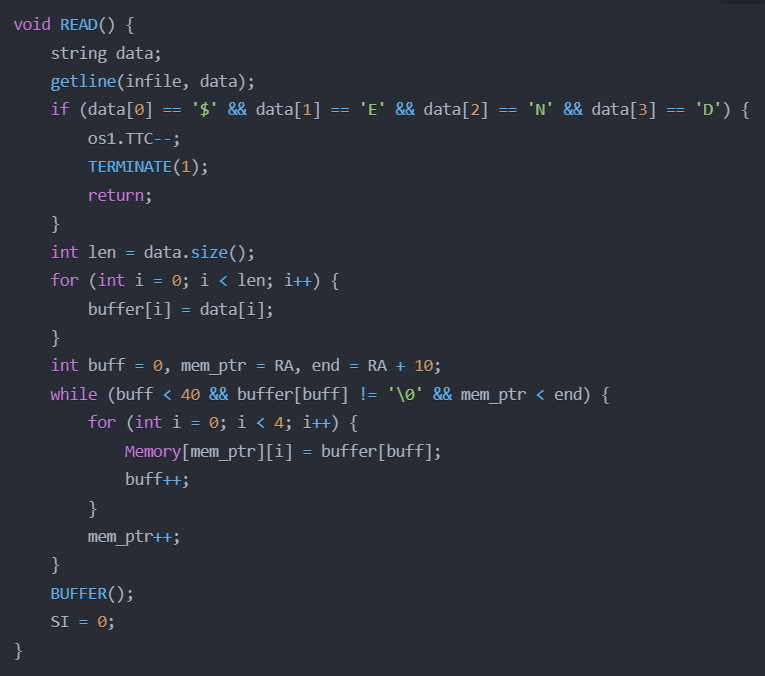
### 5.1 Interrupt Types

1. SI (System Interrupt):
   * SI=1: Get Data
   * SI=2: Print Data
   * SI=3: Halt
2. PI (Program Interrupt):
   * PI=1: Operation Error
   * PI=2: Operand Error
   * PI=3: Page Fault
3. TI (Timer Interrupt):
   * TI=2: Time Limit Exceeded

**5.2 Master Mode Operations  
**

## 6. Input/Output Operations

### 6.1 Read Operation



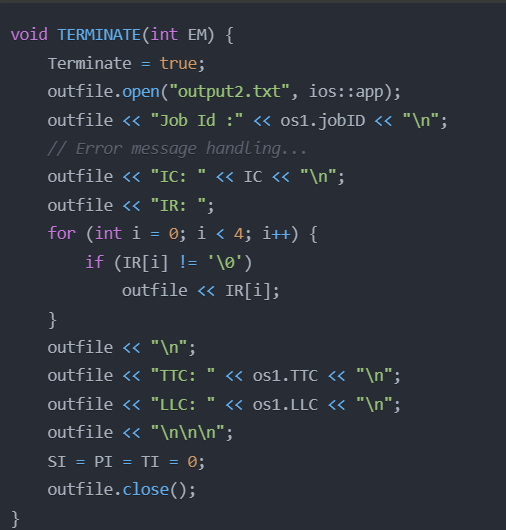
**6.2 Write Operation**

****

## 7. Error Handling

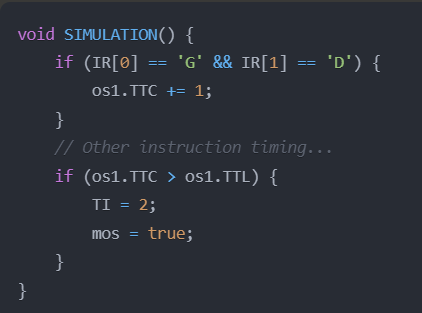
### 7.1 Error Types

1. Operation Error (PI=1)
2. Operand Error (PI=2)
3. Page Fault (PI=3)
4. Time Limit Exceeded (TI=2)
5. Line Limit Exceeded
6. Out of Data

**7.2 Termination Implementation:-  
**

## 8. Time Management

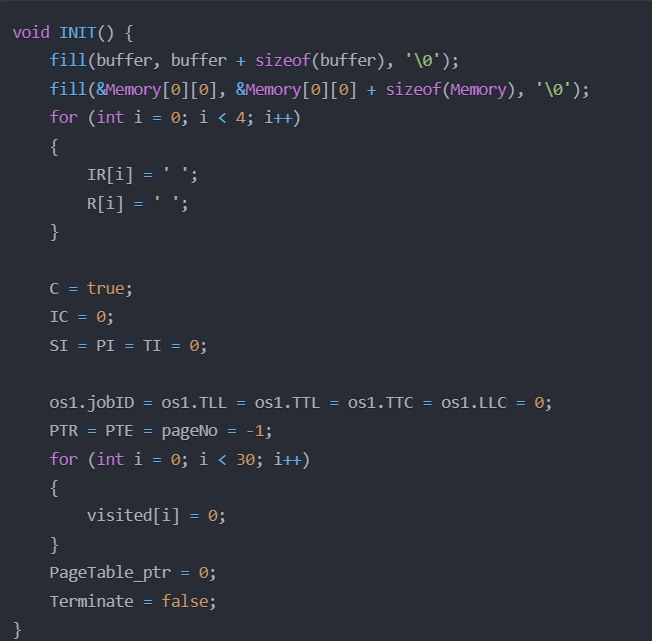
### 8.1 Time Simulation

****

# 9. Program Execution Workflow Implementation

**9.1 System Initialization Flow**

The system initialization is handled by the INIT() function which prepares the system for a new job:

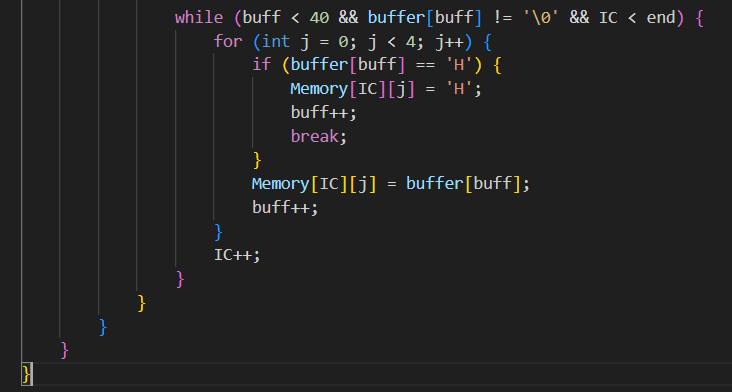
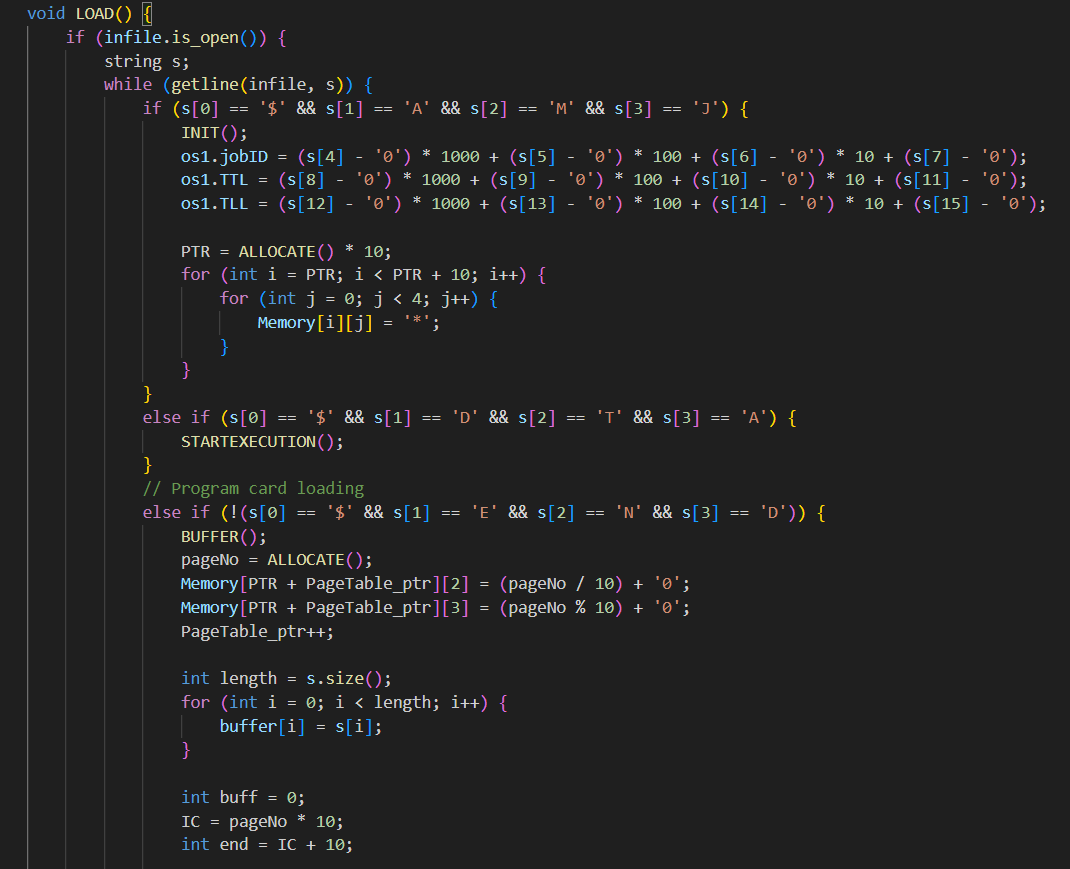
****

This initialization process:

1. Clears memory buffers
2. Resets registers
3. Initializes control flags
4. Resets page table entries
5. Prepares system for new job execution

**9.2 Job Loading Process**

The LOAD() function handles the job loading process:

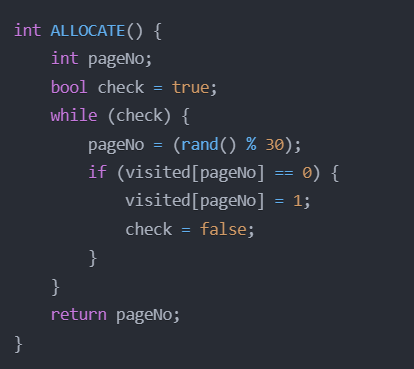


The loading process:

1. Reads control cards ($AMJ, $DTA, $END)
2. Initializes PCB with job parameters
3. Allocates page table
4. Loads program cards into memory pages
5. Manages memory allocation through paging

**9.3 Memory Allocation Implementation:-**

The ALLOCATE() function handles dynamic memory allocation:

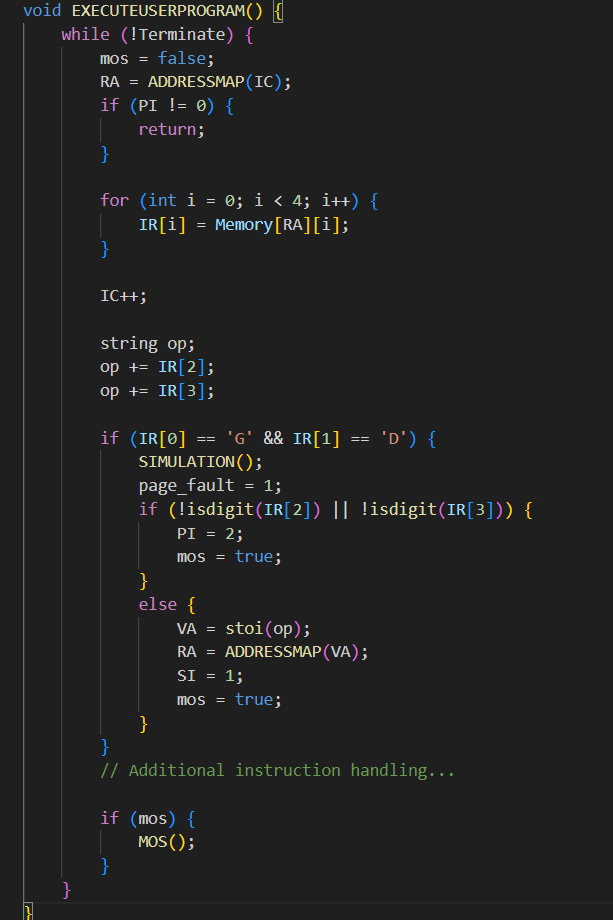


This allocation process:

1. Generates random page numbers
2. Checks for page availability
3. Marks allocated pages
4. Returns available page number

**9.4 Execution Flow Implementation:-**

The EXECUTEUSERPROGRAM() function manages the main execution cycle:

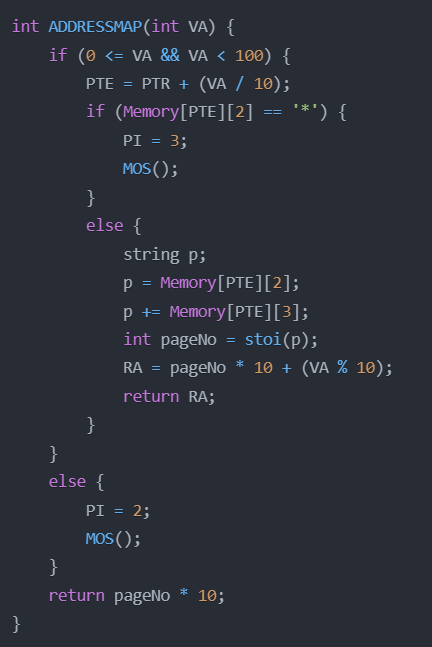


The execution cycle:

1. Fetches instructions using virtual addressing
2. Decodes instruction operands
3. Executes instructions
4. Handles interrupts and errors
5. Maintains execution time monitoring

**9.5 Address Mapping Implementation**

The ADDRESSMAP() function handles virtual to physical address translation:

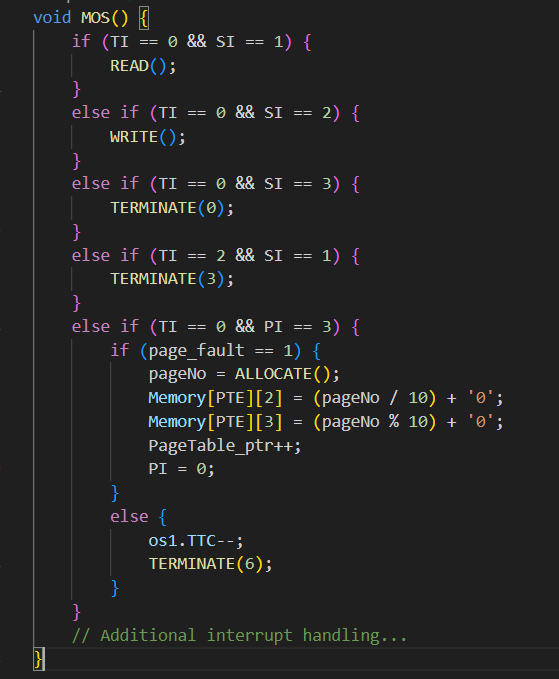


The address mapping process:

1. Validates virtual address range
2. Calculates page table entry
3. Handles page faults
4. Converts virtual to real address
5. Manages memory access errors

**9.6 Interrupt Management Flow**

The MOS() function handles system interrupts:

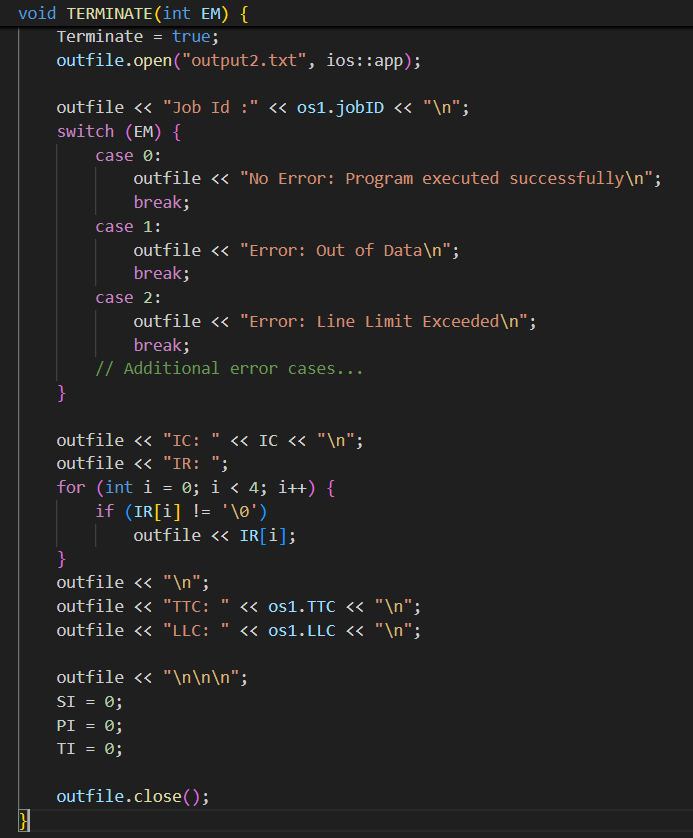


The interrupt management process:

1. Prioritizes different interrupts
2. Handles service interrupts
3. Manages program interrupts
4. Processes timer interrupts
5. Coordinates error handling

**9.7 Program Termination Implementation**

The TERMINATE() function handles program completion and errors:



The termination process:

1. Sets termination flag
2. Records execution statistics
3. Handles error reporting
4. Writes final state to output
5. Cleans up system resources

This implementation provides a complete simulation of a multiprogramming operating system with:

* Dynamic memory allocation
* Virtual memory management
* Interrupt handling
* Error management
* Resource monitoring
* Program execution control

The system successfully manages multiple jobs, handles errors gracefully, and provides detailed execution information for debugging and monitoring purposes.

**9. Input/Output Scenarios and Error Handling**

**10.1 Sample Input Cases**

### Case 1: Normal Execution

### 

**Explanation:**

* JobID: 0002
* Time Limit: 0003 (3 time units)
* Line Limit: 0001 (1 line of output)
* Program: Reads data into block 10 and prints it
* Data: "Hello World"

### Case 2: Time Limit Exceeded

### 

**Explanation:**

* Time Limit: 0002 (insufficient for all operations)
* Program attempts more operations than time allows

### Case 3: Invalid Operation

### 

**Explanation:**

* Contains invalid operation code 'XD'
* Should trigger Operation Code Error

**10.2 Error Conditions and Output**

### 1. Operation Code Error

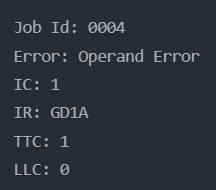
**Output:**

### 

**Trigger:** Using undefined operation codes

### 2. Operand Error

**Output:**

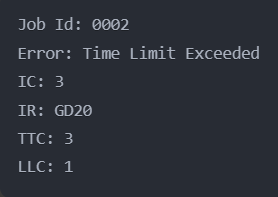


**Trigger:**

* Invalid memory address (non-numeric)
* Address outside range (0-99)

### 3. Time Limit Exceeded

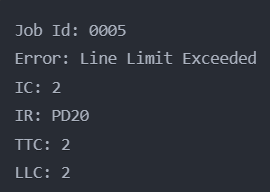
**Output:**



**Trigger:** Total Time Counter (TTC) exceeds Total Time Limit (TTL)

### 4. Line Limit Exceeded

**Output:**



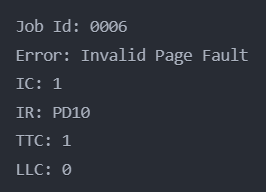
**Trigger:** Line Limit Counter (LLC) exceeds Total Line Limit (TLL)

### 5. Page Fault

**Valid Page Fault Output:**



**Invalid Page Fault Output:**



**10.3 System Response to Errors**

1. **Immediate Termination Cases:**
   * Operation Code Error
   * Operand Error
   * Invalid Page Fault
   * Out of Data Error
2. **Graceful Termination Cases:**
   * Time Limit Exceeded (completes current instruction)
   * Line Limit Exceeded (skips further output)
   * Valid Page Fault (allocates new page)

# 11. Conclusion

The Phase 2 implementation of the Multiprogramming Operating System successfully demonstrates:

1. **Memory Management**
   * Implementation of paging mechanism
   * Dynamic page allocation
   * Address mapping between virtual and real addresses
   * Page fault handling
2. **Process Management**
   * Process Control Block (PCB) implementation
   * Job scheduling and execution
   * Time and resource management
3. **Error Handling**
   * Comprehensive interrupt system (SI, PI, TI)
   * Graceful error recovery where possible
   * Detailed error reporting and logging
4. **Resource Management**
   * Time limit enforcement
   * Line limit monitoring
   * Memory allocation tracking
5. **Key Improvements over Phase 1**
   * Addition of virtual memory support
   * Implementation of paging
   * Enhanced error detection and handling
   * More sophisticated memory management
6. **System Reliability**
   * Robust error checking
   * Protected memory access
   * Resource limitation enforcement
   * Detailed system state logging

This implementation provides a solid foundation for understanding:

* Operating system memory management concepts
* Process scheduling and control
* System resource allocation
* Error handling and system reliability
* Virtual memory and paging mechanisms

The system successfully handles various error conditions while maintaining system stability and providing detailed feedback for debugging and monitoring purposes. The implementation serves as an excellent educational tool for understanding fundamental operating system concepts and their practical implementation.