

# Standard Operating Procedure

---

**Document Number:** MUTEX-SYS-001

**Standard Operating Procedure (SOP):** Mutual Exclusion  
Protocol for Two-Process Coordination

**Revision:** 1.0

**Last Updated:** [DATE]

---

## Contents

---

1. Introduction
  - 1.1 Purpose
  - 1.2 System Overview
  - 1.3 Regulatory Compliance
2. System Specifications
  - 2.1 Parameters and State Variables
  - 2.2 Mutual Exclusion Configuration
3. Operational Protocols
  - 3.1 Startup Behavior
  - 3.2 Request and Entry into Critical Section
  - 3.3 Turn-Based Conflict Resolution
4. Emergency Operations
  - 4.1 Deadlock Detection
  - 4.2 State Recovery Process
5. Maintenance Requirements
  - 5.1 Process Integrity Checks
  - 5.2 Turn Variable Diagnostics
6. Quality Assurance
  - 6.1 Liveness Verification
  - 6.2 Mutual Exclusion Invariants

- 7. Security Protocols
    - 7.1 Access Control for Shared Resources
    - 7.2 Process Isolation
  - 8. Environmental Considerations
    - 8.1 Time Determinism Under Load
    - 8.2 Fault Tolerance in Shared Memory
  - 9. Training Requirements
    - 9.1 Protocol Interpretation
    - 9.2 State Transition Analysis
  - 10. Document Control
    - 10.1 Revision History
    - 10.2 Authorization
  - 11. Process Flows and State Transitions
    - 11.1 State Transition Logic
    - 11.2 Conflict Resolution Sequence
    - 11.3 Fairness and Starvation Avoidance
- 

# 1. Introduction

---

## 1.1 Purpose

---

- Establish a reliable mutual exclusion protocol for two concurrent processes.
- Prevent simultaneous entry into the critical section to maintain data integrity.

## 1.2 System Overview

---

- The system contains two processes ( `P1` and `P2` ) managing shared access.
- Each process transitions through three logical states:
  - `n` (non-critical), `t` (trying), `c` (critical).
- Access is coordinated using a shared `turn` variable indicating priority.

### 1.3 Regulatory Compliance

---

- Aligned with formal safety and liveness verification protocols.
  - Ensures conformance to synchronization requirements in embedded and concurrent systems.
- 

## 2. System Specifications

---

### 2.1 Parameters and State Variables

---

- `state1` and `state2` : Current state of Process 1 and Process 2 respectively.
- Valid states:
  - `n1/n2` : Non-critical section
  - `t1/t2` : Trying to enter critical section
  - `c1/c2` : Inside critical section
- `turn` : Shared variable set to 1 or 2, indicating which process should yield.

### 2.2 Mutual Exclusion Configuration

---

- Both processes cannot be in `c1` and `c2` simultaneously.
  - Entry to the critical section is determined based on:
    - State of the other process.
    - Current value of the `turn` variable.
- 

## 3. Operational Protocols

---

### 3.1 Startup Behavior

---

- Initialize both processes to the non-critical section:
  - `state1 = n1` , `state2 = n2` .
- Set `turn = 1` (default starting priority to Process 1).

### 3.2 Request and Entry into Critical Section

---

- Transition from `n → t` (request phase) occurs when the other process is not in critical section.
- Transition from `t → c` (entry to critical section) is permitted if:
  - The other process is in `n` (non-critical), or
  - Both are trying and `turn` favors the current process.

### 3.3 Turn-Based Conflict Resolution

---

- After completing its critical section, a process transitions back to `n` and updates the `turn` :
  - If P1 finishes and P2 is waiting, `turn := 2` .
  - If P2 finishes and P1 is waiting, `turn := 1` .

---

## 4. Emergency Operations

---

### 4.1 Deadlock Detection

---

- Check if both processes are stuck in trying ( `t1` , `t2` ) without progressing to `c1` or `c2` .
- Trigger alert if fairness or transition liveness is violated.

### 4.2 State Recovery Process

---

- Reset `state1` , `state2` , and `turn` if deadlock or illegal state is detected.
  - Ensure mutual exclusion is re-established before resuming operations.
-

## 5. Maintenance Requirements

---

### 5.1 Process Integrity Checks

---

- Periodically verify that each process transitions through legal states only.
- Confirm state reset mechanisms are functional after each cycle.

### 5.2 Turn Variable Diagnostics

---

- Monitor consistency and fairness of `turn` updates.
- Ensure the `turn` alternates as expected to avoid process starvation.

---

## 6. Quality Assurance

---

### 6.1 Liveness Verification

---

- Confirm that if `state1 = t1`, then eventually `state1 = c1`.
- Confirm that if `state2 = t2`, then eventually `state2 = c2`.

### 6.2 Mutual Exclusion Invariants

---

- Assert that `state1 = c1` and `state2 = c2` never occur simultaneously.
  - Use model checkers to verify safety (invariants) and liveness (eventualities).
-

## 7. Security Protocols

---

### 7.1 Access Control for Shared Resources

---

- Ensure mutual exclusion is preserved during concurrent operations.
- Only one process allowed in critical section at a time.

### 7.2 Process Isolation

---

- Maintain local state per process.
- Prevent unauthorized state changes via unguarded transitions.

---

## 8. Environmental Considerations

---

### 8.1 Time Determinism Under Load

---

- Ensure state transitions occur without delay under high load.
- Avoid long waits due to missed fairness windows.

### 8.2 Fault Tolerance in Shared Memory

---

- Validate correctness of `turn` updates in memory-constrained or noisy environments.
  - Add parity or checksum checks to shared variable communication.
-

## 9. Training Requirements

---

### 9.1 Protocol Interpretation

---

- Train operators on states `n`, `t`, and `c` and their meanings.
- Understand the role of `turn` in preventing simultaneous access.

### 9.2 State Transition Analysis

---

- Practice tracing execution under normal and edge-case scenarios.
- Use visualization tools to follow the ring of state changes.

---

## 10. Document Control

---

### 10.1 Revision History

---

- Rev 1.0 – Initial SOP derived from formal mutex model (`mutex.txt`)

### 10.2 Authorization

---

- Approved by: Embedded Systems Architect
  - Reviewed by: Safety Verification Team
-

# 11. Process Flows and State Transitions

---

## 11.1 State Transition Logic

---

**For Process 1:**

- $n1 \rightarrow t1$  if state2 in (  $n2$ ,  $t2$ ,  $c2$  )
- $t1 \rightarrow c1$  if state2 is  $n2$ , or (  $t2$  and  $turn = 1$  )
- $c1 \rightarrow n1$  (after exiting critical section)

**For Process 2:**

- $n2 \rightarrow t2$  if state1 in (  $n1$ ,  $t1$ ,  $c1$  )
- $t2 \rightarrow c2$  if state1 is  $n1$ , or (  $t1$  and  $turn = 2$  )
- $c2 \rightarrow n2$  (after exiting critical section)

## 11.2 Conflict Resolution Sequence

---

- Processes resolve conflict using the `turn` variable:
  - Only one will enter the critical section when both are in `t`.

## 11.3 Fairness and Starvation Avoidance

---

- Use `turn` toggling logic to ensure neither process is starved.
- Each trying state is guaranteed eventual access (fair scheduling).