

Standard Operating Procedure

Document Number: SEM-SYS-001

Standard Operating Procedure (SOP): Two-Process Semaphore-Based Mutual Exclusion Protocol

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 State Variables and Parameters
 - 2.2 Component Architecture
3. Operational Protocols
 - 3.1 Initialization Sequence
 - 3.2 Process State Transitions
 - 3.3 Semaphore Management
4. Emergency Operations
 - 4.1 Deadlock Scenarios
 - 4.2 Semaphore Release Failures
5. Maintenance Requirements
 - 5.1 Semaphore Reset Logic
 - 5.2 Process Cycle Verification
6. Quality Assurance
 - 6.1 Safety Property Validation
 - 6.2 Liveness Property Verification

7.	Security Protocols
7.1	Shared Resource Isolation
7.2	State Machine Integrity
8.	Environmental Considerations
8.1	Scheduling Fairness Assumptions
8.2	Timing Tolerance for Semaphore Access
9.	Training Requirements
9.1	State Transition Interpretation
9.2	Debugging Entry and Exit Logic
10.	Document Control
10.1	Revision History
10.2	Authorization
11.	Process Flows and State Transitions
11.1	User Process Lifecycle
11.2	Semaphore Access Logic
11.3	Fair Execution Assumptions

1. Introduction

1.1 Purpose

- Ensure mutually exclusive access to critical sections between two processes.
- Use a binary semaphore to prevent simultaneous access and ensure proper sequencing.

1.2 System Overview

- The system includes two processes (`proc1` , `proc2`) and a shared boolean semaphore.
- Each process is modeled as a finite state machine with defined behavior during resource acquisition and release.

- Transitions are conditional based on the semaphore's state.

1.3 Regulatory Compliance

- Adheres to formal verification standards for concurrent systems.
- Guarantees safety (no two processes in critical section) and liveness (each process eventually enters critical section if trying).

2. System Specifications

2.1 State Variables and Parameters

- semaphore : A shared boolean flag (TRUE means held, FALSE means free).
- state : The internal state of each process; values include:
 - idle : Not requesting access
 - entering : Attempting to acquire the semaphore
 - critical : In the critical section
 - exiting : Releasing the semaphore

2.2 Component Architecture

- Each user module (proc1, proc2) operates as an instance of a parameterized process.
 - The main controller initializes the semaphore and manages the two processes.
 - A fairness constraint ensures ongoing scheduling for each process.
-

3. Operational Protocols

3.1 Initialization Sequence

- On system start:
 - `semaphore` is set to `FALSE` (not held).
 - Each process begins in the `idle` state.

3.2 Process State Transitions

- `idle` → `entering` : The process attempts to access the critical section.
- `entering` → `critical` : Transition occurs if `semaphore` is `FALSE` .
- `critical` → `exiting` : Process completes its critical section.
- `exiting` → `idle` : The semaphore is released and the cycle repeats.

3.3 Semaphore Management

- When a process enters `entering` , it sets `semaphore := TRUE` .
 - On exiting, it resets `semaphore := FALSE` .
 - This binary semaphore restricts access to one process at a time.
-

4. Emergency Operations

4.1 Deadlock Scenarios

- If both processes remain in `entering` without a semaphore release, liveness could stall.
- Ensure fairness is enforced to prevent starvation or indefinite postponement.

4.2 Semaphore Release Failures

- If a process does not reach `exiting`, the semaphore may remain engaged.
- In this case, a system-level watchdog may reset the semaphore after timeout.

5. Maintenance Requirements

5.1 Semaphore Reset Logic

- System diagnostics should ensure `semaphore` is reset properly upon process exit.
- Manual override may be needed in case of fault recovery.

5.2 Process Cycle Verification

- Each process should cycle through `idle → entering → critical → exiting → idle`.
- Logging and audit trails must capture the full sequence to validate execution.

6. Quality Assurance

6.1 Safety Property Validation

- System must always satisfy:
`NOT (proc1.state = critical AND proc2.state = critical)`

6.2 Liveness Property Verification

- Ensure that:
`AG (proc1.state = entering → AF proc1.state = critical)`
 - Guarantees that each attempt eventually leads to access.
-

7. Security Protocols

7.1 Shared Resource Isolation

- The semaphore acts as a gatekeeper for shared memory.
- Only one process may modify shared resources during `critical` .

7.2 State Machine Integrity

- State transitions must not be bypassed or externally modified.
- Internal transitions are tightly controlled and state-driven.

8. Environmental Considerations

8.1 Scheduling Fairness Assumptions

- Fair scheduling is critical to ensuring that both processes eventually gain access.

8.2 Timing Tolerance for Semaphore Access

- Ensure the system clock granularity allows reliable access to `entering` logic without collision.

9. Training Requirements

9.1 State Transition Interpretation

- Engineers must understand the meaning of each state.
- Entry into `critical` only occurs if the semaphore is available.

9.2 Debugging Entry and Exit Logic

- Trace logs should be used to identify improper transitions.
- Validate correct release of semaphore after `exiting`.

10. Document Control

10.1 Revision History

- Rev 1.0 – Initial SOP derived from formal semaphore protocol (`semaphore.txt`)

10.2 Authorization

- Approved by: Systems Concurrency Architect
- Reviewed by: Formal Methods Review Board

11. Process Flows and State Transitions

11.1 User Process Lifecycle

- `idle → entering → critical → exiting → idle`

11.2 Semaphore Access Logic

- Access to `critical` is only permitted when `semaphore = FALSE`
- Upon entry, `semaphore := TRUE`; upon exit, `semaphore := FALSE`

11.3 Fair Execution Assumptions

- Fairness ensures that each process is eventually scheduled (`FAIRNESS running`)

- Prevents starvation and ensures forward progress