University of Europe for Applied Sciences

Report of Assignment:
“Process Synchronization Mechanisms with Random Job Generation”

Marco Sebastian Vizñay Cabrera

Operating Systems

Software Engineering B.Sc. “B”

Winter Semester 2024-25

Introduction

This report documents the development, execution, and analysis of a system that simulates process synchronization in a shared environment. The main objective was to implement and compare different mechanisms of mutual exclusion (Mutexes, Semaphores, and Peterson's Solution) to solve problems such as race and deadlocks conditions, while using a pre-emptive system to plan the execution of works.

Programming in Java, I'm going to simulate a workspace where physical jobs are happening, different office workers (users) are going to be randomly assigned either a printing or scanning job, which will have random number of pages and arrival times.

Job Generator

The job generator aims to create 10 jobs for each of the 5 users (P1 to P5), assigning them randomly generated attributes such as:

- Job Type: Print or scan.

- Length: Short (1-5 pages), medium (6-15 pages) or large (16-50 pages)

- User: The user who initiated the job.

- Arrival time: Random interval time between 1 to 5

Each page of a job requires 1 second to process, simulating real-world delays.

A random Boolean number generator is used to assign the type of job to each user.

For the length of the job, the medium length is the more common one between the lengths, this was made with the intention to replicate real world examples and to ensure a vary of lengths without complete randomness. First find a random number between 0 and 2, giving us a 33% of chances to get a short job when we get 0 and a medium job if we get 1 or 2. For the long jobs we find a random number between 0-4 ensuring a 20% of a long job if a 0 is encountered.

The arrival time is randomly generated between 1-5 and added to an initial *previousTime=0* this ensures that the arrival times are increasing simulating a workflow.

Without synchronization

Jobs access resources (printer or scanner) without restrictions, leading to race conditions and mixed results.
Observed Issues:

- Two or more jobs attempt to use the same resource simultaneously.
- Progress messages from jobs are mixed, making it difficult to identify which job used which resource.

With synchronization

Integration: In all versions, jobs are interrupted after completing every 2 pages to give priority to the oldest job.
Results:

- Improved fairness among jobs.
- Allowed older jobs to be processed more quickly.

Mutexes
A ReentrantLock was used to ensure that only one job accesses the printer or scanner at a time.
Results:

- Jobs accessed resources exclusively.
- Accuracy improved significantly.

Semaphores
Semaphores (Semaphore) were used to handle exclusive access to resources.
Results:

- Worked similarly to mutexes, ensuring mutual exclusion.
- Allowed efficient handling of multiple jobs in the queue.

Peterson's Solution

Peterson's solution was applied to ensure mutual exclusion between two processes (jobs).

Results:

- Limited to two simultaneous processes.
- Demonstrated basic mutual exclusion, but not scalable for multiple users.

Results Analysis

For this we list the average execution time for each job

Mechanism	Average time	Comments
Without synchronization	52668.8	Race conditions caused interference.
Mutexes	669316.8	Correctly excludes but adds slight overhead.
Semaphores	623653	Like mutexes but allows more controlled queues.
Peterson’s Solution	617223.6	Correctly excludes but limited to two processes.

Conclusion

This project demonstrated:

- The importance of mutual exclusion mechanisms: Mutexes and semaphores resolved race conditions and ensured exclusive access.
- Limitations of Peterson's Solution: Useful for teaching basic mutual exclusion but impractical for multiple processes.
- Benefits of the pre-emptive system: Improved fairness and reduced wait times for older jobs.