8-2 Short Paper: Deadlock Avoidance

Darren J. Oates

Southern New Hampshire University

IT-600-10151-M01 Operating Systems 2025 D-4

Michael Gallaher

October 19th, 2025

Deadlock Avoidance

During the implementation of its new and improved back-office operating system, Top Secret, Inc. (TSI) finds that the TSI OS lacks mechanisms for the ordered acquisition of synchronization object. Mechanisms would include; "**Monitors**- A design pattern that builds synchronization into an object by synchronizing all of its public methods. A thread must acquire the monitor's built-in lock to enter any synchronized method. Every object can be used as a monitor. **Mutexes** (mutual exclusion)-A synchronization primitive that can be owned by only one thread at a time, ensuring exclusive access to a shared resource. Threads must acquire ownership of the mutex before accessing the resource and must release it afterward. **Semaphores**- A signaling mechanism that controls access to a common resource by maintaining a count. A semaphore is initialized with a specific value. When a thread wants to access the resource, it decrements the count. If the count is zero, the thread blocks until a signal allows it to proceed. **Barriers**- A synchronization point where threads wait until a specified number of threads have reached it. Each thread calls a wait() or similar method on the barrier. Threads block until the required number of threads arrive. All waiting threads are then unblocked simultaneously (BillWagner, 2022)."

The result of TSI's OS lack of mechanisms is that its web servers are periodically locking up. The best synchronization primitive to solve the issue is a monitor — specifically, a mechanism that ensures ordered acquisition of locks and avoids deadlock through structured synchronization. Processes are active (ps -ef), but web server is stuck. Web server is blocked on pthread_mutex_lock().Reboot temporarily resolves the issue. Issue reappears under high load. This is a classic deadlock scenario or lock-ordering issue, where the system lacks mechanisms to manage the order in which synchronization objects (e.g., mutexes) are acquired. The use of raw pthread_mutex_lock() without higher-level coordination is leading to threads waiting on locks indefinitely.(GeeksforGeeks,2025)

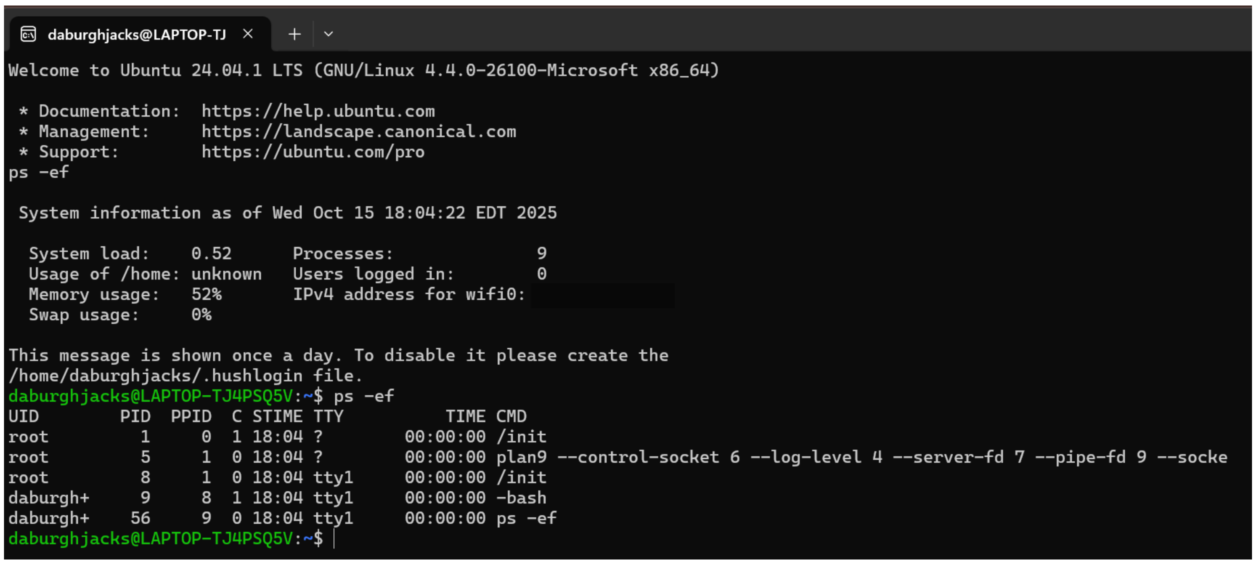
Choosing the Right Primitive

- Mutexes- Already used, but causing deadlocks due to poor ordering — not sufficient alone.
- Semaphores- Can be complex and error-prone; doesn't inherently solve lock ordering issues.
- Barriers- Irrelevant here; this isn't a coordination problem across multiple threads reaching a stage.
- **Monitors**- BEST OPTION- Provides structured locking, encapsulated mutual exclusion, and avoids manual error-prone locking.(Monitors in process synchronization,2014)

Why monitors are the best option is that they encapsulate locking logic. You don't lock/unlock mutexes manually; monitor handles it. Monitors prevent deadlocks by using a structured access model and automatic lock management. They also ensure order by implementing priority or queue-based access to critical sections. They also provide cleaner abstraction and are easier to maintain in large systems like OS components or web servers. Monitors are the best method to solve the issue. They offer structured, high-level synchronization that prevents the kind of deadlocks caused by unordered and manual mutex acquisition, which is exactly what TSI is struggling with. What TSI should do is refactor critical code (especially the web server portion) to use a monitor-based synchronization mechanism. (ChatGPT. 2025) Visitors to the website should then be able to make purchases during peak periods.

Image using the ps-ef Command on my System with Ubuntu;

Accessed October 15th, 2025, showing active processes-



```
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 4.4.0-26100-Microsoft x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
ps -ef

 System information as of Wed Oct 15 18:04:22 EDT 2025

  System load:    0.52      Processes:             9
  Usage of /home: unknown   Users logged in:       0
  Memory usage:   52%       IPv4 address for wifi0: [redacted]
  Swap usage:     0%

This message is shown once a day. To disable it please create the
/home/daburghjacks/.hushlogin file.
daburghjacks@LAPTOP-TJ4PSQ5V:~$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  1 18:04 ?        00:00:00 /init
root         5     1  0 18:04 ?        00:00:00 plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9 --socke
root         8     1  0 18:04 tty1     00:00:00 /init
daburgh+     9     8  1 18:04 tty1     00:00:00 -bash
daburgh+    56     9  0 18:04 tty1     00:00:00 ps -ef
daburghjacks@LAPTOP-TJ4PSQ5V:~$ 
```

References

BillWagner. (2022). *Overview of synchronization primitives - .NET.* .NET | Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/standard/threading/overview-of-synchronization-primitives

ChatGPT. (2025, October 16). Recommendation for synchronization primitive to resolve deadlock issue in TSI OS. OpenAI. Personal communication.

GeeksforGeeks. (2025, August 30). Monitors in process synchronization. GeeksforGeeks. https://www.geeksforgeeks.org/operating-systems/monitors-in-process-synchronization/

“Monitors in process synchronization.” (n.d.). In Wikipedia. https://en.wikipedia.org/wiki/Monitor_%28synchronization%29