

1. No. if the access is read for both threads, then concurrency error will not occur.
2. b) , c) and d) are true

Correct solution

c) and d) are true

Notes

Question What does it mean when mutex is held by this thread?

Question What I do know is that `pthread_cond_wait` puts thread to sleep. My question here is, how come the mutex is not held when thread is in a blocked state/sleep?

3. a) Only b) causes starvation.
- b) Conditional variable is a queue that allows threads to be put themselves on to sleep (in blocked state) when thread it is not desired using `pthread_cond_wait` function.

Since there are no threads inside `cv1`, there is nothing to awake using `pthread_cond_signal`.

So, nothing will occur.

- c) System call is a subset of interrupt caused by user application to switch from user mode to kernel mode to perform privileged operations for the application.

Interrupt is a signal sent by hardware (e.g keyboard, mouse, hard drive) or software.

It tells the cpu to stop its activities and execute appropriate part of the operating system.

Notes

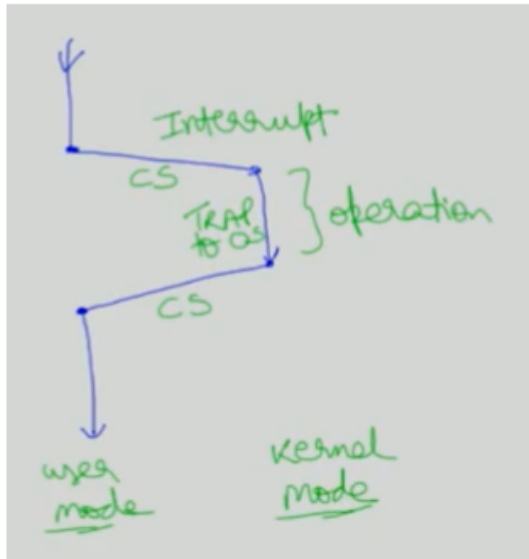
- I need to review how interrupt works. I had to look up the information.

Question How does interrupt work?

- **Interrupt**

- Is a signal
- there are two types of interrupts:
 - * Hardware interrupt
 - Is signal generated by hardware (e.g RAM is full, Hard drive is full)
 - Is sent to operating system
 - * Software interrupt
 - Is signal generated by software (e.g program crash, system call)

- Is sent to operating system
- May call trap instruction (esp. system call)



References

1. venkatesan ramachandran, What is an Interrupt?, link
- d) No. This statement is false.

User level threads are generated in user-mode without kernel being aware about it.

Notes

Question What is the difference between user-level thread and kernel-level thread?

Question Why is thread that is generated at user level using procedure call faster than kernel level thread?

Question What is procedure call? How does it work?

- **Procedure call**

- works in user-mode only
- doesn't require context switching
- doesn't need help from OS/Kernel
- no context-switching → faster

References

1. Tech Dose, System call vs Procedure call, link

e) System calls do not generate processes. `fork()` does.

With this reason the program `run_stuff` generates only 1 additional process.

Notes

Question What is a process? And how does process work?

Question How come system call doesn't generate process? And how come `fork()` generates process?

- **Process**

- Is a running program
- Has 3 states

1. **Running:**

- * means a process is running on a processor
- * means instructions are being executed

2. **Ready:**

- * means a process is ready to run
- * means OS has chosen not to run the program at the given moment

3. **Blocked:**

- * means a process has performed some kind of operation that makes it not ready to run until some event takes place