TIE-02506 Concurrency

Therac-25 project

Student name: Trinh Gia Huy (290290), Huynh Khuong Van ()

1. Write a summary about **the concurrency problems** in the system
2. If you'd be allowed to change any part of the system (and money is no issue), what fixes would you implement?
3. Based on the system description and our course topics - could you fix the software? How confident you'd be on your fix? (Would you let the machine running your fix treat you?)

1.

Therac-25 was a radiation therapy machine which offered 2 modes: direct electron-beam therapy and Megavolt X-ray (photon) therapy. Should the machine produce a photon beam with the beam flattener not in position, the patient is subjected to a higher, potentially lethal dose. This is a common hazard of dual-mode machines. Therac-25 relied on software controls to switch between modes rather than physical hardware. Furthermore, it lacked the hardware interlocks that could have mitigated the errors that resulted in the Tyler accident, the interlocks which its predecessors, the Therac-20, had.

The Tyler accidents were related to problems in the data-entry routines that allowed the code to proceed before the full prescription had been entered and acted upon. Specifically, during the setting up of the Therac25' magnets, any edits made could only be recognized if a specific flag, which indicated the bending magnets were being set, was raised. However, the subroutine for introducing a time delay, Ptime, called during the setting up of magnets, had already cleared the flag, and any changes made weren't registered.

The second problem in the Tyler accidents concerned the Data Entry Complete variable, which indicated whether the data entry was complete. However, the variable worked differently from its intended behavior, which resulted in any changes made not being registered.

The Yakima accident was attributed to a type of software "bug", race condition, which allowed devices to be activated in error setting. The Yakima Software bug is concerned with the variable Class3, which was used to check for inconsistency and if the machine could proceed. However, the Class3 variable was one byte, containing a maximum value of 255 decimal. Therefore, on the 256^{th} time the variable passing through the check, the variable overflowed and had a zero value, in which case the erroneous setting was allowed to run.

Race conditions played a significant role in the accidents. There was no real synchronization aside from data that are stored in shared variables, even though the software allows concurrent access to such variables. Furthermore, the “test” and “set” for such variables were not indivisible operations.

2.

Beside the changes implemented by AECL mentioned in the study, I would change the software by dividing operations as to prevent unnecessary concurrent processes. In case it is necessary to run processes concurrently and use the same shared variable, I would introduce a safeguard in addition to lock guard.

As another layer of safeguard, I would reintroduce hardware interlocks, which lessen reliance on software designs.

3.

Based on the system description and course topics, I believe I can fix the software and am confident enough to let the machine running my fix treat me.