Лабораторна робота 1

«Аналіз проблем зі збоями в програмному забезпеченні»

Мета: Ознайомитися з основами тестування та світовими проблемами, спричиненими збоями в АЗ/ПЗ.

Завдання 1. Знайти дві світові проблематики, спричинені збоями в ПЗ. Описати їх у звіті та розповісти про них.

Завдання 2. Описати проблеми, яки сталися в зв'язку зі збоями.

Завдання 3. Проаналізувати, як можна було бі запобігти цім збоям.

Форма звіту

1. Тема та мета
2. Коротко теоретичні відомості
3. Звіт повинен містити опис двох проблематик з відповідями на такі питання:
    - Що сталося?
    - Як була спричинена проблема?
    - Як можна було запобігти цій проблемі?
    - Знайдені проблеми додати в таблицю https://docs.google.com/spreadsheets/d/1H3QMIm4k9uN276sylaWSh5FdJTtyyHuCcL7Yp8k5az4/edit?usp=sharing
    - Як найменше одна з найдених проблем повина бути унікальною для кожного студента

4. Короттенький висновок з аналізу ціх проблем

5. (optional) Відповіді на запитання по теорії та за результатами виконаної роботи.

ПРИКЛАД ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ З ПОКРОКОВИМ ОПИСОМ ЗАВДАНЬ

Звіт про світові проблематики, спричинені збоями в АЗ/ПЗ

Вступ

У сучасному світі, де критична інфраструктура, фінанси, охорона здоров'я та космічні дослідження все більше покладаються на комп'ютерні системи, збої в апаратному (АЗ) та програмному (ПЗ) забезпеченні можуть призводити до катастрофічних наслідків. У цьому звіті розглядаються дві такі значні світові проблематики, які ілюструють важливість надійності, тестування та відповідального підходу до розробки технологій.

Проблематика 1. Аварія ракети-носія Ariane 5 (політ 501).

1.1 Що сталося?

4 червня 1996 року нова європейська ракета-носій Ariane 5 зруйнувалася через 40 секунд після старту з космодрому Куру у Французькій Гвіані. Ракета разом з дорогим корисним вантажем (супутниками для дослідження сонячно-вітрової взаємодії) на борту була повністю знищена на висоті близько 4 км. Збитки оцінили в півмільярда доларів США. Ця аварія стала однією з найдорожчих програмних помилок в історії.

1.2 Як була спричинена проблема?

Причина аварії крилася в програмному забезпеченні системи інерційного відліку (Inertial Reference System, SRI), яке було запозичене без належної адаптації з попередньої моделі ракети, Ariane 4.

1. **Програмний модуль, відповідальний за вирівнювання інерційної платформи**, обчислював певні дані (горизонтальну швидкість) навіть після зльоту, хоча ці дані були потрібні лише доки ракета знаходилась на землі для престартової калібровки.
2. **Перетворення 64-бітного числа з плаваючою комою в 16-бітне ціле число** призвело до **переповнення** (arithmetic overflow), оскільки горизонтальна швидкість Ariane 5 (яка була потужнішою за Ariane 4) значно перевищила значення, на яке був розрахований старий код.
3. Програмний модуль аварійно завершив роботу, передавши в систему керування дані про помилку (дамп), які система інтерпретувала як коректні дані про траєкторію польоту.
4. Це призвело до того, що двигуни ракети відхилилися в неправильному напрямку, створивши критичне аеродинамічне навантаження, яке і зруйнувало ракету.

1.3 Як можна було запобігти цій проблемі?

1. **Повне перетестування запозиченого коду:** Код з Ariane 4 не можна було використовувати "як є". Його потрібно було повністю перевірити та протестувати в умовах, що імітують характеристики нової ракети Ariane 5, зокрема її більш високі швидкості та прискорення.
2. **Обробка виняткових ситуацій:** Програмне забезпечення має коректно обробляти помилки (наприклад, переповнення), а не просто аварійно завершувати роботу в критичній системі керування польотом. Можна було просто відключити функцію вирівнювання після зльоту, оскільки вона більше не була потрібною.
3. **Аналіз відмовостійкості (Fault Tolerance):** Система мала бути спроектована так, щоб відмова одного некритичного модуля не призводила до катастрофи. Дані про помилку не повинні інтерпретуватися як коректні команди.

4. **Незалежний аудит коду:** Незалежна команда експертів могла б виявити цю потенційну вразливість під час перегляду архітектури програмного забезпечення.

Проблематика 2. Смертельні дози радіації від апарату для променевої терапії Therac-25.

2.1 Що сталося?

У період між 1985 і 1987 роками в США і Канаді сталося низка інцидентів, пов'язаних з медичним апаратом для променевої терапії Therac-25. Пацієнти отримували масивні передозування рентгенівського випромінювання (у десятки тисяч разів вище за лікувальну дозу), що призводило до важких опіків, паралічу та смерті щонайменше п'яти людей. Therac-25 став символом смертельної небезпеки програмних помилок.

2.2 Як була спричинена проблема?

Проблема була викликана фатальною комбінацією програмних помилок (багів) та недоліків в апаратному забезпеченні.

1. **Помилка під назвою "Race Condition" (стан гонки):** Оператор міг швидко змінити режим роботи апарату (наприклад, з електронного на фотоновний) на командній панелі. Програмне забезпечення не встигало коректно обробити цю зміну. Якщо оператор робив це дуже швидко і вносив корекції, виникала ситуація, коли електронний промінь включався на повну потужність, але без цільового фільтра-розсіювача, який безпечно знижує інтенсивність променя. Це призводило до того, що пацієнт отримував концентровану летальну дозу.

2. **Ненадійне програмне забезпечення:** Програмний код містив критичні помилки і не мав належного захисту для подібних крайніх випадків використання.
3. **Відсутність апаратного захисту:** На відміну від попередніх моделей (Therac-6 і Therac-20), які мали незалежні апаратні блокуючі механізми (interlocks) для запобігання таким ситуаціям, в Therac-25 покладалися виключно на програмне забезпечення для безпеки. Це була фатальна архітектурна помилка.

2.3 Як можна було запобігти цій проблемі?

1. **Запобігти "Single Point of Failure":** Критичні для безпеки системи ніколи не повинні покладатися лише на один шар захисту (у цьому випадку — тільки на ПЗ). **Незалежні апаратні блокуючі механізми** (які були в попередніх моделях) обов'язково мають дублювати програмні перевірки.
2. **Ретельне тестування крайніх випадків (Edge Case Testing):** Програмне забезпечення має тестуватися не лише в ідеальних умовах, а й в умовах помилок оператора, швидких змін команд та інших нестандартних сценаріїв.
3. **Аналіз причин та наслідків відмов (FMEA - Failure Mode and Effects Analysis):** Проведення такого аналізу допомогло б виявити потенційні небезпечні режими роботи системи на етапі проектування.
4. **Прозорість та реагування на скарги:** Коли перші пацієнти почали скаржитися на "опіки" під час процедур, виробник спочатку списував це на людський фактор або чутливість пацієнтів, а не на проблеми в апараті. Швидше розслідування та усвідомлення проблеми могло б врятувати життя.

Висновок

Інциденти з Ariane 5 та Therac-25 назавжди залишаться класичними прикладами того, як програмні помилки, посилені недоліками в проектуванні систем, можуть призводити до катастрофічних наслідків. Головні уроки, які можна з них винести:

1. **Запозичений код потребує повного ретельного тестування в новому середовищі.**
2. **Критичні системи вимагають багаторівневої безпеки (апаратної та програмної) для уникнення єдиної точки відмови.**
3. **Необхідне ретельне тестування на всіх рівнях, включаючи обробку помилок і крайні випадки.**
4. **Культура безпеки та відкритості до зворотного зв'язку має бути пріоритетом, особливо в таких галузях, як космонавтика та медицина.**