UNIVERSIDADE D
COIMBRA

Relatório do Assignment 1

Qualidade e Confiabilidade de Software (QCS)

Bruno Silva 2021232021

Diogo Honório 2021232043

Introdução

O presente relatório foi realizado no âmbito da cadeira de Qualidade e Confiabilidade de Software com o objetivo de apresentar os mecanismos de *fault tolerence* aplicados no decorrer do trabalho.

Estes mecanismos foram implementados em dois dos requisitos funcionais disponibilizados pelo professor, sendo estes o requisito funcional três, que simula o aviso da pressão dos pneus, e o requisito funcional quatro, que consiste num sistema de sensor de marcha atrás.

1. Lista de Mecanismos de Fault Tolerance Considerados

FR3

TMR e DMR	(Hardware redundancy)
Verificação de erros num período de tempo	(Time redundancy)
Data consistency	(Consistency check)
Time consistency	(Consistency check)
Backward recovery	(Error recovery)
N-version programming	(Software diversity)

FR4

N-version programming	(Software diversity)
Data consistency	(Consistency check)
Forward recovery	(Error recovery)
Time consistency	(Consistency check)

2. Mecanismos de Fault Tolerance Implementados

Os seguintes mecanismos descritos foram implementados em um dos requisitos funcionais, ou em ambos:

2.1. Forward Recovery (FR4)

Este mecanismo de forward recovery consiste na transformação do estado de erro do sistema em um novo estado em que o mesmo consiga prosseguir na sua execução. Este tipo de *fault tolerance* foi aplicado em certos casos, como validações de dados provenientes de inputs, conseguindo, assim, ignorar este tipo de erros quando possível evitando que o sistema fique preso sem comprometer a integridade do mesmo.

2.2. Backward Recovery (FR3)

Esta técnica permite detetar um erro e retornar a um estado anterior, ignorando as operações que causaram esse mesmo erro. Para isso, utiliza-se checkpoints, onde o estado do sistema é armazenado ao longo do tempo.

No nosso código, implementámos esta funcionalidade ao primeiramente guardar os dados de entrada num checkpoint. Se ocorrer algum erro durante o processo, o sistema irá tentar realizar os cálculos novamente com os dados guardados do checkpoint. Cada erro poderá fazer uso desta mecânica até três vezes, no caso se o erro persistir, o sistema irá abortar a operação, interpretando o mesmo como um erro crítico.

Este mecanismo foi implementado com o objetivo de permitir ao sistema recuperar de falhas temporárias, voltando ao último estado válido guardado (checkpoint). Assim, evita-se a perda de dados e garante-se a continuidade de funcionamento do sistema sem comprometer a sua integridade.

2.3. Data Consistency (FR3 & FR4)

Ambos os requisitos funcionais contêm este mecanismo implementado de modo a garantir que todos os dados sejam válidos e coerentes antes de proceder à sua execução. Este tipo de implementação é possível encontrar em casos como na verificação do número de sensores mínimo no FR4.

Casos como o anterior também são encontrados no FR3, como a verificação de que o número de pneus é exatamente quatro, que os valores das pressões e da pressão desejada devem ser sempre positivos, e que valores de pressão igual a 0 PSI são ignorados.

Adicionalmente, em ambos os requisitos, os dados de entrada são validados, de modo a garantir que os dados possuem sempre o tipo de dados correto.

Este mecanismo foi implementado para garantir que todos os dados são validados antes de se proceder aos cálculos. Esta etapa é fundamental, pois podem ocorrer erros que não têm origem direta nos sistemas (FR3 & FR4), mas sim nos inputs fornecidos. Ao validar previamente a informação, evita-se que dados corrompidos ou inconsistentes influenciem os resultados, impedindo assim a propagação de erros nos sistemas.

Esta abordagem contribui significativamente para que o funcionamento dos sistemas não seja condicionado por erros de outros sistemas de onde provém os inputs afetados, deste modo protegesse a si mesmo não condicionando o seu funcionamento e mantendo a precisão do funcionamento geral. Isto é essencial especialmente em contextos como este, onde a segurança e a confiança nas decisões do sistema são cruciais.

2.4. Triple Modular Redundancy (FR3)

Este mecanismo de Fault Tolerance foi implementado no FR3 e recorre ao uso de três módulos idênticos para garantir um sistema com uma maior fidelidade. No contexto deste trabalho, são feitos três processos com cópias do código responsáveis por calcular a pressão dos pneus, sendo posteriormente aplicado uma função de "voting" para escolher o valor mais frequente dos três calculados.

De forma mais prática, utilizam-se três instâncias da função "compute_averages", executadas em paralelo, cada uma é responsável por calcular a pressão de forma independente. Após este cálculo, os resultados são enviados para a função "voting", que compara os três valores e escolhe o que ocorre pelo menos duas vezes. Caso todos os valores sejam diferentes, é apresentada uma mensagem de erro.

Este método foi escolhido porque, num sistema como este, em que erros podem ter consequências graves, é fundamental garantir a máxima precisão nos resultados obtidos. Ao implementar o TMR (Triple Modular Redundancy), temos como objetivo aumentar significativamente a fiabilidade do sistema, uma vez que o código é executado três vezes e, para que o resultado seja considerado válido, pelo menos duas das execuções têm de coincidir.

Desta forma, o condutor pode tomar decisões mais seguras sobre, por exemplo, quando deve aumentar ou não a pressão dos pneus. Além disso, no caso de ocorrer um erro momentâneo no sistema, desde que dois dos três resultados sejam iguais, não será necessário alertar o condutor. Assim, evita-se gerar alarmes desnecessários, tornando o sistema mais eficaz e transmitindo maior confiança ao utilizador, ao sinalizar apenas quando é realmente necessário.

2.5. Time Redundancy (FR3)

Este mecanismo baseia-se na realização de múltiplas verificações ao longo do tempo, para aumentar a confiabilidade do sistema, pois tem como principal função tentar contornar possíveis falhas momentâneas.

No nosso sistema, caso seja detetado um erro, a operação é repetida até três vezes. Se o erro for temporário, esta abordagem tem a finalidade de corrigi-lo antes de ser considerado crítico, garantindo assim que o sistema continue a funcionar mesmo perante falhas pontuais. Por outras palavras ao repetir a mesma operação ao longo do tempo, é possível confirmar se um erro é persistente ou apenas momentâneo.

Desta forma, este mecanismo é considerado importante porque torna o sistema mais robusto e seletivo na sinalização de falhas, apresentando um erro apenas quando este é consistente, o que evita alarmes falsos e aumenta a confiança do utilizador no funcionamento do sistema.

2.6. N-Version Programming (FR4)

Este mecanismo consiste em múltiplas versões independentes de design e código ao mesmo conjunto de requisitos, fornecendo uma diversidade de abordagens e metodologias. Neste caso, foram usadas as seguintes versões: a nossa solução ao problema e as versões fornecidas sem *fault tolerance* do grupo 3 e 13. As adaptações realizadas nos códigos fornecidos foram mínimas, de modo a preservar a metodologia original de cada grupo — inclusive no caso do grupo 3, cuja implementação original em C++ foi convertida para Python sem comprometer a lógica do algoritmo.

Para garantir que o resultado seja conseguido, é implementado, também, um *software voter*. O *software voter* é uma função simples que executa as várias versões em paralelo, com recurso ao uso de threads, e verifica se os resultados obtidos das versões são iguais ou não. Caso sejam iguais este retorna o valor que chegou em consenso, caso contrário retorna o valor com maioria entre as versões.

Este mecanismo foi escolhido para este requisito pois as metodologias que podem ser implementadas para este problema conseguem ser mais diversificadas em relação ao requisito 3, podendo produzir abordagens distintas que contribuem para a robustez do sistema. Além disso, ao utilizar diferentes implementações, reduzimos a probabilidade de falhas correlacionadas, aumentando a confiabilidade da solução.

2.7. Time Consistency (FR3 & FR4)

A time consistency refere-se à capacidade de um sistema garantir que determinadas operações sejam concluídas dentro de um intervalo de tempo específico.

Esta mecânica é assegurada em ambos os *requirements* do nosso trabalho. Por exemplo, no FR3 é usada uma "pool" com três processos a executar em paralelo, devido ao mecanismo do TMR. Como os processos são executados ao mesmo tempo existe a possibilidade de um deles ficar preso num loop infinito, bloqueando assim os outros processos.

De modo a evitar esse cenário, foi definido um *timeout* de um segundo para cada processo. Caso algum processo ultrapasse este tempo, é interpretado como preso num loop infinito e é considerado como erro.

Este mecanismo foi implementado para evitar que os processos do TMR no FR3 e as threads do FR4 entrem em loops infinitos, visto que isto poderia bloquear sistema. Caso um processo demore mais do que o necessário é automaticamente interrompido. Esta abordagem é considerada essencial para quando se trata de processos em paralelo em que um processo mais lento pode comprometer a resposta de todo o sistema.

3. Mecanismos não implementados

3.1. DMR - Dual Modular Redundancy

Este modelo foi considerado durante o desenvolvimento, porém acabou por não ser implementado, dado que o TMR apresenta ser eficaz para o problema em questão. O DMR consiste na execução paralela de duas versões do sistema, comparando os resultados obtidos. No entanto, caso ocorra uma divergência entre os resultados, o sistema apenas consegue detetar a existência de um erro, sem conseguir identificar qual das versões está correta. Ou seja, o DMR atua apenas como mecanismo de deteção de falhas.

3.2. N-Version Programming

Embora tenha sido implementado no FR4, o mecanismo também foi considerado para o FR3. No entanto, a sua implementação não avançou, pois, o FR3 limita-se a calcular a média das pressões, verificar se é superior à pressão desejada e ordenar os valores. Dada a simplicidade e a natureza direta dessas operações, não há muitas variações possíveis na implementação do código, tornando o uso do N-Version Programming redundante. Então decidi-mos não aplicar este mecanismo no FR3 devido a não haver diversidade suficiente nas soluções para justificar a aplicação do mesmo.

4. Fault Models:

Neste trabalho foram considerados os seguintes Fault Models: Intermittent Faults, Permanent Faults, Data Consistency Faults e Timing Faults

4.1. Intermittent Faults:

As Intermittent Faults tratam-se de falhas momentâneas que de forma irregular, estas falhas foram consideradas em mecanismos como os de Time Redundancy, Backward Recovery, TMR, N-Version Programming.

Todos estes mecanismos fazem uso de mecânicas como repetições de operações ou processos de *voting* se proteger contra falhas não consistentes, que só se manifestam às vezes.

4.2. Permanent Faults:

Os Permanent Faults são considerados erros que se mantêm no sistema até que a parte causadora do mesmo seja corrigida por terceiros. Este erro foi considerado no limite de três tentativas de repetição de operações, e no TMR e N-Version Programming em que

no caso de um processo ou versão de código ser incorreta continuamos a ter um output no caso dos outros dois processos ou versões se sobreporem.

Estes mecanismos conseguem detetar um erro permanente e dependendo do caso até isolá-lo de modo que não afete o sistema.

4.3. Data Consistency Faults:

Os Data Consistency Faults são erros que ocorrem quando os parâmetros de entrada estão incompletos, corrompidos, diferentes do esperado ou incoerentes. Estes erros podem acontecer mesmo que o sistema esteja funcional, pois podem resultar de erros de outros sistemas. Estes erros foram considerados na implementação de mecanismos de Data Consistency.

Os mecanismos aplicados protegem o sistema de falhas originadas por dados externos e que erros de outros sistemas se propaguem para o sistema em causa.

4.4. Timing Faults:

Os Timing Faults são erros que ocorrem quando um sistema não cumpre um tempo estipulado para uma operação. Estes erros foram postos em consideração, recorrendo à implementação do Time Consistency, garantindo que, em pools de processos ou em threads, haja um timeout para o caso de ficarem num loop infinito, impedindo que o sistema bloqueie.

Divisão do trabalho:

Bruno Silva: REQ 4 e implementação do relatório.

Diogo Honório: REQ 3 e implementação do relatório.

Conclusão

Este trabalho demonstrou a importância da aplicação de mecanismos de tolerância a falhas na construção de sistemas mais robustos e confiáveis. Através de diferentes abordagens, como *Triple Modular Redundancy*, *Recovery* e *N-Version Programming*, foi possível mitigar diversas falhas, assegurando a continuidade e integridade das operações. A definição de *fault models* adequados foi essencial para orientar as escolhas técnicas, reforçando a resiliência do sistema frente a comportamentos inesperados.