(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2022-0087247
(43) 공개일자 2022년06월24일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01) *G06F 11/30* (2006.01)
G06F 9/30 (2018.01)
(52) CPC특허분류
G06F 11/3648 (2013.01)
G06F 11/302 (2013.01)
(21) 출원번호 10-2020-0177755
(22) 출원일자 2020년12월17일
심사청구일자 2020년12월17일

(71) 출원인
경상국립대학교산학협력단
경상남도 진주시 진주대로 501 (가좌동)
(72) 발명자
전용기
경상남도 진주시 내동면 순환로 425-61, 111동 2004호 (남강휴먼빌아파트)
최으뜸
경상남도 진주시 진주역로96번길 15, 102동 902호 (가좌동, 포레나 신진주)
김태형
경상남도 진주시 가호로 26, 213동 804호 (가좌동, 진주가좌주공아파트)
(74) 대리인
김종선

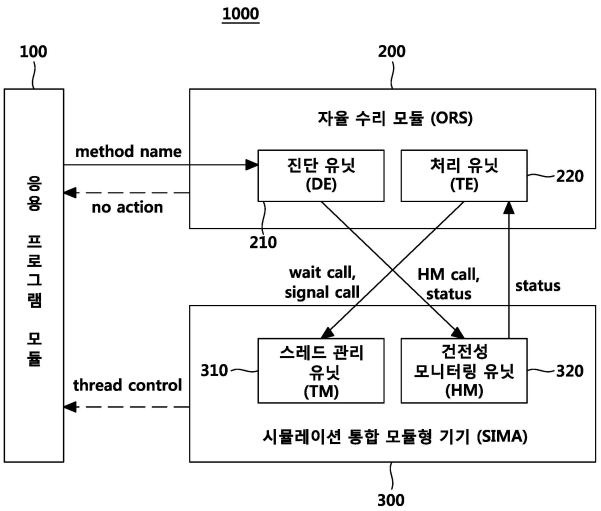
전체 청구항 수 : 총 8 항

(54) 발명의 명칭 **항공기 소프트웨어에서 순서위배의 자율적 수리를 위한 건전성 관리 시스템 및 방법**

(57) 요 약

항공기 소프트웨어의 건전성 관리 시스템 및 방법이 제공된다. 본 발명의 일 실시예에 따른 항공기 소프트웨어의 건전성 관리 시스템은, 항공기 소프트웨어의 함수 호출 순서(Type State Automaton, TSA)를 사용하여 순서 위배를 진단하고, 상기 순서 위배에 따른 스레드 제어명령을 지시하는 자율 수리 모듈(On-the-fly Repairing), 상기 순서 위배를 모니터링하고, 상기 지시된 스레드 제어명령에 따라 스레드를 관리하는 시뮬레이션 통합 모듈형 기기(Simulated Integrated Modular Avionics), 함수 이름(method name)을 상기 자율 수리 모듈로 전달하고, 상기 스레드 제어명령에 따라 제어를 수행하는 응용 프로그램 모듈을 포함하는 구성이다.

대 표 도 - 도2



(52) CPC특허분류
G06F 11/3636 (2013.01)
G06F 9/3009 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1345315287
과제번호	2018R1D1A3B07041838
부처명	교육부
과제관리(전문)기관명	한국연구재단
연구사업명	이공학학술연구기반구축(R&D)
연구과제명	항공기 소프트웨어의 원자성 위배를 실시간으로 치유하는 결정적 프레임웍 개발
기 여 율	1/1
과제수행기관명	경상대학교
연구기간	2018.06.01 ~ 2021.05.31

명 세 서

청구범위

청구항 1

항공기 소프트웨어의 함수 호출 순서(Type State Automaton, TSA)를 사용하여 순서 위배를 진단하고, 상기 순서 위배에 따른 스레드 제어명령을 지시하는 자율 수리 모듈(On-the-fly Repairing);

상기 순서 위배를 모니터링하고, 상기 지시된 스레드 제어명령에 따라 스레드를 관리하는 시뮬레이션 통합 모듈형 기기(Simulated Integrated Modular Avionics);

함수 이름(method name)을 상기 자율 수리 모듈로 전달하고, 상기 스레드 제어명령에 따라 제어를 수행하는 응용 프로그램 모듈;

을 포함하는 항공기 소프트웨어의 건전성 관리 시스템.

청구항 2

제 1항에 있어서,

상기 응용 프로그램 모듈은,

ARINC 653 규격의 항공기 소프트웨어에서 수행되는 것을 특징으로 하는 항공기 소프트웨어의 건전성 관리 시스템.

청구항 3

제 1항에 있어서,

상기 자율 수리 모듈은,

기설정된 함수 호출 순서(Type State Automaton, TSA) 정보를 사용하여 상기 순서 위배를 진단하는 진단 유닛;

상기 순서위배에 따른 스레드 제어명령의 메시지를 상기 시뮬레이션 통합 모듈형 기기로 전송하는 처리 유닛;

을 포함하는 항공기 소프트웨어의 건전성 관리 시스템.

청구항 4

제 3항에 있어서,

상기 진단 유닛은,

상기 순서위배에 따라 스레드 대기 또는 재개 시점을 진단하는 것을 특징으로 하는 항공기 소프트웨어의 건전성 관리 시스템.

청구항 5

제 1항에 있어서,

상기 시뮬레이션 통합 모듈형 기기는,

상기 순서위배를 모니터링하는 건전성 모니터링 유닛;

상기 지시된 스레드 제어명령에 따라 스레드를 관리하는 스레드 관리 유닛;

을 포함하는 항공기 소프트웨어의 건전성 관리 시스템.

청구항 6

항공기 소프트웨어의 건전성 관리 시스템이 항공기 소프트웨어의 건전성을 관리하는 방법에 있어서,

응용 프로그램 모듈이 항공기 소프트웨어의 함수 이름(method name)을 진단 유닛으로 전달하는 단계;

상기 진단 유닛이 기설정된 함수 호출 순서(Type State Automaton, TSA)를 사용하여 순서위배를 진단하는 단계;

상기 순서위배의 진단 결과를 전달받는 건전성 모니터링 유닛이 상기 순서 위배를 모니터링하는 단계;

상기 모니터링 결과에 따라 처리 유닛이 상기 모니터링된 순서 위배에 따른 메시지를 처리하여 스레드 관리 유닛에 전달하는 단계;

상기 스레드 관리 유닛이 상기 처리된 메시지에 따라 스레드를 상기 응용 프로그램 모듈에서 수행되도록 전달하는 단계;

를 포함하는, 항공기 소프트웨어의 건전성 관리 방법.

청구항 7

제 6항에 있어서,

상기 응용 프로그램 모듈은,

ARINC 653 규격의 항공기 소프트웨어에서 수행되는 것을 특징으로 하는 항공기 소프트웨어의 건전성 관리 방법.

청구항 8

제 6항에 있어서,

상기 진단하는 단계는,

상기 순서위배에 따라 스레드 대기 또는 재개 시점을 진단하는 것을 특징으로 하는 항공기 소프트웨어의 건전성 관리 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 항공기 소프트웨어의 건전성 관리 시스템에 관한 것이다. 더욱 상세하게, 항공기 소프트웨어의 순서위배를 자율적으로 수리하기 위한 건전성 관리 시스템 및 방법에 관한 것이다.

배 경 기 술

[0002] 항공전자 기술의 발달에 따라 항공기 운용에 요구되는 기능을 소프트웨어로 구현하는 비중이 급격하게 증가하고 있다. 항공기에 소프트웨어가 차지하는 비율은 1960년에 생산된 F-4의 경우 8%의 비중을 차지하며, 2007년 생산된 F-35의 경우 대략 90%까지 증가된 것을 알 수 있다. 이처럼 높은 비중을 차지하는 항공기 소프트웨어는 적용 범위, 규모 및 복잡도 등을 증가시키기 때문에 항공기 시스템에서 발생 가능한 잠재적인 오류도 크게 증가시킬 수 있다. 예를 들어, 항공기 소프트웨어의 잠재적인 오류는 불시착 사고, 추락 사고, 사망 사고 등으로 이어질 수 있다.

[0003] 상기한 사고를 방지하기 위해 소프트웨어의 오류가 발생하더라도 사고로 이어지지 않도록 즉시 수리하는 시스템이 필요하다. 즉, 잠재적인 오류는 시스템의 안전성 문제를 초래하므로 반드시 관리할 수 있어야 한다.

[0004] 최근 항공기 운행 중에 소프트웨어 오류를 진단 및 수리조치하여 프로그램/어플리케이션이 정상적으로 수행되도록 하는 항공기 소프트웨어의 건전성 관리 시스템이 개발되고 있다.

[0005] 여기서, 항공기의 건전성 관리 시스템(Health management System, HMS)은 항공기 운용 중에 발생되는 오류를 감시(monitoring), 진단(diagnosis), 조치(treatment)하여 프로그램을 정상적으로 수행되도록 하는 것으로, 프로그램을 실시간으로 감시(monitoring)하고, 실행 중인 소프트웨어에서 오류의 발생을 탐지(detection)하여 진단(diagnosis)하며, 진단된 오류를 수리하도록 적절한 복구(recovery) 기법을 적용하여 조치(treatment)하는 것을 목적으로 한다.

[0006] 구체적으로, 항공기의 건전성 관리 시스템(HMS)에서 감시(monitoring) 단계는 시스템의 수행 중에 발생하는 오류 및 관련 이벤트의 수행 정보를 수집하여 모니터링하고, 진단(diagnosis) 단계는 시스템의 감시(monitoring) 단계를 통해 수집된 정보를 이용하여 발생한 오류의 유형을 분석하거나 잠재적 오류의 발생을 예측하여 진단하며, 조치(treatment) 단계는 발생한 오류를 수리하기 위해 적절한 복구(recovery) 기법을 적용하여 소프트웨어가 정상적으로 수행되도록 하는 구성이다.

[0007] 항공기 소프트웨어에서 발생할 수 있는 오류 중, 동시성 오류는 병행 프로그램에서 적절한 동기화 기법이 없이 적어도 하나의 쓰기 사건을 포함하는 공유변수 접근이 있을 때 발생하게 된다. 동시성 오류 중, 순서위배 오류(Order Violation Errors)는 개발자가 의도한 스레드 간의 공유변수 접근사건 순서를 보장할 수 없는 오류에 해당된다. 상기한 오류는 스레드 사이의 모든 인터리빙(Interleaving)을 고려하여 오류를 디버깅할 수 없으므로, 잠재적으로 프로그램에 내재될 수 있다. 잠재된 순서위배 오류의 발생은 개발자가 의도하지 않은 비결정적인 결과를 초래하여 시스템의 실패로 이어지게 된다.

[0008] 예를 들어, 종래의 순서위배 오류에 대한 소스코드는 도 1과 같다. 도 1에 도시된 바와 같이, Thread 1의 ReadWriteProc()에서 공유변수 io_pending을 TRUE로 초기화(W1)한 후에 Thread 2의 DoneWaiting()에서 같은 공유변수를 FALSE로 변경(W2)하는 순서를 의도하였으나, PBReadAsync()가 예상보다 빠르게 진행되어 Thread 2의 DoneWaiting()에서 공유변수를 FALSE로 변경하고 Thread 1의 ReadWriteProc()에서 공유변수를 TRUE로 변경하는 순서로 실행될 수 있다. 이러한 실행은 Thread 1이 while문을 탈출하지 못하게 하여 무한 루프를 발생시키게 된다. 즉, PBReadAsync()가 개발자의 예상보다 빨리 진행되어 일어난 순서위배 오류를 발생하게 되었다.

[0009] 따라서, 종래의 항공기 소프트웨어는 개발 단계에서 소스코드 상의 순서위배 오류를 탐지하는 것에 한계를 가지게 된다.

[0010] 또한, 항공기 소프트웨어에서 순서위배는 개발 단계에서 오류를 모두 제거하기 어렵기 때문에 운용 단계에서 자율적인 수리가 필요하다. 최근 시험 단계에서 획득한 올바른 접근 순서와 수행 중 접근 순서를 비교하여 오류를 진단하는 항공기 소프트웨어의 순서위배를 자율 수리하는 건전성 관리 시스템이 개발되고 있으나, 접근 사건 수에 비례하는 시간 오버헤드가 발생하게 되기 때문에 항공기 시스템에 적용 시 실시간성을 만족시키기 어렵다.

[0011] 따라서, 항공기 소프트웨어의 건전성 관리 시스템에서 순서위배가 발생되기 전에 접근 사건을 포함하는 함수 간의 순서위배를 자율적으로 수리하는 시스템 및 방법이 필요하다.

선행기술문헌

특허문헌

[0012] (특허문헌 0001) 대한민국 등록특허 10-1440505호 (2014.09.04 등록)

(특허문헌 0002) 대한민국 등록특허 10-1690948호 (2016.12.23 등록)

(특허문헌 0003) 대한민국 등록특허 10-1694305호 (2017.01.03 등록)

비특허문헌

[0013] (비특허문헌 0001) Choi, E. T., Lee, D. S., Jun, Y. K., and Lee, S. J., "On-the-fly Atomicity Violation Repairing Technique for Airborne Health Management Systems," Journal of The Korean Society for Aeronautical and Space Sciences, Vol. 48, No. 7, 2020, pp. 547~554.

발명의 내용

해결하려는 과제

[0014] 본 발명은 상기 문제점을 해결하기 위한 것으로서, 항공기 소프트웨어의 순서위배를 자율적으로 수리하기 위한 건전성 관리 시스템 및 방법을 제공하는데 그 목적이 있다.

[0015] 본 발명의 다른 목적은 접근사건을 포함하는 함수의 올바른 순서 정보를 통해 순서위배 오류를 진단하고, 오류에 대한 해당 함수를 지연시켜 처리하기 때문에 작은 시간 오버헤드로 수리할 수 있으므로, 이에 따라 안전성을 극대화시키는 항공기 소프트웨어의 건전성 관리 시스템 및 방법을 제공하는데 그 목적이 있다.

[0016] 본 발명의 또 다른 목적은 항공기 소프트웨어에서 접근사건이 포함된 함수 간의 순서위배 오류를 진단하여 오류가 진단된 스레드는 대기시키고, 정상적으로 수행되어야 할 스레드를 진행한 후에 대기 중인 스레드를 재개시키기 때문에, 실시간으로 스레드를 관리할 수 있는 항공기 소프트웨어의 건전성 관리 시스템 및 방법을 제공하는데 그 목적이 있다.

[0017] 본 발명이 해결하고자 하는 과제들은 이상에서 언급한 과제들로 제한되지 않으며, 언급되지 않은 또 다른 과제들은 아래의 기재로부터 당업자에게 명확하게 이해될 수 있을 것이다.

과제의 해결 수단

[0018] 이와 같은 목적을 달성하기 위한 본 발명은, 항공기 소프트웨어의 함수 호출 순서(Type State Automaton, TSA)를 사용하여 순서 위배를 진단하고, 상기 순서 위배에 따른 스레드 제어명령을 지시하는 자율 수리 모듈(On-the-fly Repairing), 상기 순서 위배를 모니터링하고, 상기 지시된 스레드 제어명령에 따라 스레드를 관리하는 시뮬레이션 통합 모듈형 기기(Simulated Integrated Modular Avionics), 함수 이름(method name)을 상기 자율 수리 모듈로 전달하고, 상기 스레드 제어명령에 따라 제어를 수행하는 응용 프로그램 모듈을 포함하는 항공기 소프트웨어의 건전성 관리 시스템을 제공한다.

[0019] 상기 응용 프로그램 모듈은, ARINC 653 규격의 항공기 소프트웨어에서 수행되는 것을 특징으로 한다.

[0020] 상기 자율 수리 모듈은, 기설정된 함수 호출 순서(Type State Automaton, TSA) 정보를 사용하여 상기 순서 위배를 진단하는 진단 유닛, 상기 순서위배에 따른 스레드 제어명령의 메시지를 상기 시뮬레이션 통합 모듈형 기기로 전송하는 처리 유닛을 포함하는 구성이다.

[0021] 상기 진단 유닛은, 상기 순서위배에 따라 스레드 대기 또는 재개 시점을 진단하는 것을 특징으로 한다.

[0022] 상기 시뮬레이션 통합 모듈형 기기는, 상기 순서위배를 모니터링하는 건전성 모니터링 유닛, 상기 지시된 스레드 제어명령에 따라 스레드를 관리하는 스레드 관리 유닛을 포함한다.

[0023] 본 발명의 다른 특징에 따르면, 항공기 소프트웨어의 건전성 관리 시스템이 항공기 소프트웨어의 건전성을 관리하는 방법에 있어서, 응용 프로그램 모듈이 항공기 소프트웨어의 함수 이름(method name)을 진단 유닛으로 전달하는 단계, 상기 진단 유닛이 기설정된 함수 호출 순서(Type State Automaton, TSA)를 사용하여 순서위배를 진단하는 단계, 상기 순서위배의 진단 결과를 전달받는 건전성 모니터링 유닛이 상기 순서 위배를 모니터링하는 단계, 상기 모니터링 결과에 따라 처리 유닛이 상기 모니터링된 순서 위배에 따른 메시지를 처리하여 스레드 관리 유닛에 전달하는 단계, 상기 스레드 관리 유닛이 상기 처리된 메시지에 따라 스레드를 상기 응용 프로그램 모듈에서 수행되도록 전달하는 단계를 포함하는, 항공기 소프트웨어의 건전성 관리 방법을 제공한다.

[0024] 상기 응용 프로그램 모듈은, ARINC 653 규격의 항공기 소프트웨어에서 수행되는 것을 특징으로 한다.

[0025] 상기 진단하는 단계는, 상기 순서위배에 따라 스레드 대기 또는 재개 시점을 진단하는 것을 특징으로 한다.

발명의 효과

[0026] 본 발명에 따르면, 실시간 시스템에 적용할 수 있을 만큼의 작은 시간 오버헤드로 수리할 수 있으므로, 공유변수 접근횟수가 많은 프로그램에서 순서위배를 수리할 경우에도 항공기 소프트웨어 시스템에 대한 안전성을 극대화시킬 수 있는 효과가 있다.

[0027] 또한, 본 발명은 항공기 소프트웨어에서 접근사건이 포함된 함수 간의 순서위배 오류를 진단하여 오류가 진단된 스레드는 대기시키고, 정상적으로 수행되어야 할 스레드를 진행한 후에 대기 중인 스레드를 재개시키기 때문에,

실시간으로 스레드를 관리할 수 있다. 이에 따라, 항공기 소프트웨어 시스템에 대한 정확성을 극대화시킬 수 있으며 항공기의 실제비행에서 발생될 치명적인 에러를 미연에 방지할 수 있는 효과가 있다.

[0028] 또한, 본 발명은 순서위배를 진단하는 단계에서 공유변수 접근사건의 순서가 아닌 접근 사건을 포함하는 함수 간의 순서를 기준으로 함으로써, 공유변수 접근 사건의 횟수가 증가하더라도 함수호출 수에만 비례하는 시간 오버헤드를 가질 수 있다.

[0029] 또한, 본 발명은 사전 시험 단계에서 시공간 오버헤드가 발생되지 않으므로, 항공기 시스템의 소프트웨어 품질을 향상시킬 수 있으며 유지보수 비용을 경감할 수 있다.

도면의 간단한 설명

[0030] 도 1은 종래의 순서위배 오류에 대한 소스코드 예시도이다.

도 2는 본 발명에 일 실시에 따른 항공기 소프트웨어의 순서위배를 자율적으로 수리하기 위한 건전성 관리 시스템을 도시한 구성도이다.

도 3는 본 발명에 일 실시에 따른 항공기 소프트웨어의 건전성 관리 방법의 순서를 도시한 도면이다.

도 4a, 4b 및 도5는 본 발명에 일 실시에 따른 함수 호출 순서(TSA)를 설명하기 위한 예시도이다.

도 6는 본 발명에 일 실시에 따른 진단 유닛(DE)(210)의 알고리즘 소스코드이다.

도 7는 본 발명에 일 실시에 따른 처리 유닛(TE)(220)의 알고리즘 소스코드이다.

도 8은 본 발명에 일 실시에 따른 순서위배 오류를 수리하는 예시도이다.

도 9은 본 발명에 일 실시에 따른 함수 호출 순서(TSA)를 테이블 형식으로 구현한 표이다.

도 10은 본 발명에 일 실시에 따른 시뮬레이션 통합 모듈형 기기(SIMA)에서 출력되는 화면 예시도이다.

도 11은 본 발명에 일 실시에 따른 자율 수리 모듈(ORS)(200)의 동작을 확인할 수 있는 소스코드이다.

도 12는 도 11의 유형에 대한 출력 화면의 예시도이다.

도 13은 본 발명의 효율성을 검증하기 위한 합성 프로그램의 유형을 설명하기 위한 표이다

도 14는 도 13의 각 유형을 측정한 수행 시간을 설명하기 위한 표이다.

도 15는 도 14의 case 1에 대해 종래의 Repairing-AV과 본 발명의 자율 수리 모듈(ORS)의 시간 비교 그래프이다.

발명을 실시하기 위한 구체적인 내용

[0031] 본 발명의 목적 및 효과, 그리고 그것들을 달성하기 위한 기술적 구성들은 첨부되는 도면과 함께 상세하게 후술되어 있는 실시 예들을 참조하면 명확해질 것이다. 본 발명을 설명함에 있어서 공지 기능 또는 구성에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략할 것이다.

[0032] 그리고 후술되는 용어들은 본 발명에서의 기능을 고려하여 정의된 용어들로서 이는 사용자, 운용자의 의도 또는 관례 등에 따라 달라질 수 있다.

[0033] 그러나 본 발명은 이하에서 개시되는 실시 예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 수 있다. 단지 본 실시 예들은 본 발명의 개시가 완전하도록 하고, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 발명은 청구항의 범주에 의해 정의될 뿐이다. 그러므로 그 정의는 본 명세서 전반에 걸친 내용을 토대로 내려져야 할 것이다.

[0034] 이와 같은 본 발명은 비록 한정된 실시 예와 도면에 의해 설명되나, 본 발명은 이것에 의해 한정되지 않으며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에 의해 본 발명의 기술 사상과 아래에 기재될 청구범위의 균등 범위 내에서 다양한 수정 및 변형이 가능함은 물론이다.

[0035] 이하에서는 도면에 도시한 실시 예에 기초하면서 본 발명에 대하여 더욱 상세하게 설명하기로 한다. 그러나, 본원이 이러한 실시 예와 도면에 제한되는 것은 아니다. 본 발명을 설명하기에 앞서 본 발명과 관련된

ARINC(Aeronautical Radio Incorporated)에 대해 살펴본다.

[0036] ARINC(Aeronautical Radio Incorporated)은 미국의 비영리 단체로서, 지상 기지국과 항공기 간의 통신 서비스 및 항공전자 표준의 표준 규격을 정의하며 전세계의 항공기와 관련된 통신 서비스를 제공한다. 예를 들어, ARINC은 항공기를 사용하는 모든 업계에 항공기의 신제품과 항공기에 사용되는 부품이나 전자기기들의 새로운 규격을 재정의하고, 항공 운항의 안전성 향상 및 호환성 유지를 위해 항공전자 표준(ARINC 400 ~ 900 계열)을 제정한다. ARINC-653은 ARINC에서 정의한 표준 규격 중 하나에 해당한다.

[0037] ARINC 653은 실시간 운영체제와 상기 실시간 운영체제 상에서 동작하는 응용 프로그램 간의 인터페이스를 규정한 것으로, 항공용 어플리케이션 소프트웨어 개발을 위한 규격이다. ARINC 653은 통합 모듈형 항공전자(Integrated Modular Avionics, IMA) 시스템에 필요한 실시간 운영체제 API 표준으로 운영체제와 그 위에서 동작하는 응용 프로그램 간의 인터페이스를 규정한다. 즉, ARINC 653은 통합 모듈형 항공전자(Integrated Modular Avionics, IMA) 시스템을 목적으로 항공기 시스템의 운영체제와 그 위에서 동작하는 응용 프로그램 간의 인터페이스를 규정하는 기술표준에 해당한다.

[0038] 이러한 ARINC 653의 핵심 서비스는 시공간 분할로 하나의 시스템 내에서 응용 프로그램이 실행되는 공간과 시간을 분리시켜서 하나의 응용 프로그램의 오류가 다른 응용에 영향을 미치는 것을 방지하는 것이다.

[0039] 이하, 본 발명에 따른 항공기 소프트웨어의 건전성 관리 시스템(1000)의 바람직한 실시예를 첨부된 도면을 참조하여 설명한다.

[0040] 도 2는 본 발명에 일 실시에 따른 항공기 소프트웨어의 순서위배를 자율적으로 수리하기 위한 항공기 소프트웨어의 건전성 관리 시스템(1000)을 도시한 구성도이다.

[0041] 도 2에 도시된 바와 같이, 본 발명에 따른 항공기 소프트웨어의 건전성 관리 시스템(1000)은 응용 프로그램 모듈(100), 자율 수리 모듈(On-the-fly Repairing System, ORS)(200), 시뮬레이션 통합 모듈형 기기(Simulated Integrated Modular Avionics, SIMA)(300)를 포함할 수 있다. 그리고, 상기 자율 수리 모듈(ORS)(200)에는 진단 유닛(Diagnosis Engine, DE)(210) 및 처리 유닛(Treatment Engine, TE)(220)이 구비되며, 상기 시뮬레이션 통합 모듈형 기기(SIMA)(300)에는 스레드 관리 유닛(Thread Management, TM)(310) 및 건전성 모니터링 유닛(Health Monitor, HM)(320)가 구비된다.

[0042] 본 발명의 실시 예로서, 항공기 소프트웨어의 건전성 관리 시스템(1000)은 ARINC-653 표준에 기반한 것으로서, 자율 수리 모듈(ORS)(200) 및 시뮬레이션 통합 모듈형 기기(SIMA)(300)는 ARINC-653 표준에 기반한 운영 체제에 의해 동작하는 것일 수 있다.

[0043] 응용 프로그램 모듈(100)은 부분적으로 명령(예를 들면, 소프트웨어) 및 메모리 장치에 저장된 하나 이상의 데이터베이스(예를 들면, 하드 드라이브, 플래시 메모리 등)를 기초로 하나 이상의 동작을 수행할 수 있다.

[0044] 응용 프로그램 모듈(100)은 ARINC 653 표준 기반의 실시간 운영체제가 수행되는 메모리 상태의 읽기 또는 메모리에 값을 쓰기 등을 할 수 있는 구성이다. 응용 프로그램 모듈(100)의 메모리 내용 중 특정한 함수 이름(method name)이 사용된다. 이때, 함수 이름(method name)은 진단 유닛(DE)(210)에 전달할 수 있다.

[0045] 진단 유닛(Diagnosis Engine, DE)(210) 및 처리 유닛(Treatment Engine, TE)(220)를 포함하는 자율 수리 모듈(ORS)(200)은 순서위배 오류를 진단하기 위해 함수 호출 순서를 사용하며, 자율 수리 모듈(ORS)(200)에서 순서위배 오류가 진단되면 해당 상황에 맞는 스레드 제어를 통해 순서위배 오류를 조치처리할 수 있다.

[0046] 본 발명의 일 실시예에 따르면, 자율 수리 모듈(ORS)(200)은 발생한 오류가 다른 소프트웨어에 전파되지 않고 건전성을 관리하기 위해 ARINC 653 규격을 사용할 수 있다.

[0047] 진단 유닛(DE)(210)은 함수 이름(method name)과 함수의 수행 정보의 비교를 통해 순서위배 오류 또는 스레드를 재개하기 위한 시점을 진단한다. 여기서, 오류를 진단하기 위한 함수 수행 정보는 기설정된 함수 호출 순서(Type State Automaton, TSA)일 수 있다. 함수 호출 순서(TSA)는 특정 상태에서 각 함수가 호출되었을 때, 변하는 상태 정보들을 가지기 때문에 해당 정보들로 함수의 순서를 명시하여 오류를 진단할 수 있다.

[0048] 진단 유닛(DE)(210)은 일정 시간 만료, 예상 동작과의 차이, 특정 센서모듈의 입출력 등의 정보를 통해 오류의 종류를 진단한다. 이때, 진단 유닛(DE)(210)은 오류의 종류에 따라 진단에 필요한 정보들을 감시하여 특정 값의 차이를 분석하거나 탐지 프로토콜 등을 이용하여 오류의 발생을 진단하게 된다.

[0049] 처리 유닛(TE)(220)은 순서위배에 따른 스레드 제어명령의 메시지를 처리하도록 한다. 스레드 제어명령의 메시

지는 순서위배에 따라 스레드 대기하는 메시지 및 스레드 재개하는 메시지로 구분된다.

[0050] 처리 유닛(TE)(220)은 스레드 제어명령의 메시지에 따라 forward recovery 및 backward recovery 등의 처리 방식을 사용할 수 있다. Forward recovery는 순서위배 오류를 진단하면 오류가 예측되는 해당 접근사건을 지연시켜 오류를 수리 및 처리하는 방식이며, Backward recovery는 순서위배 오류가 발생하지 않은 특정 시점의 체크포인트를 유지하여 오류가 발생했을 시점으로 되돌리는 방식이다.

[0051] 구체적으로, 처리 유닛(TE)(220)은 진단 유닛(DE)(210)에서 순서위배 오류가 진단될 경우 상기한 오류가 발생한 스레드를 대기시키는 메시지를 처리하도록 지시 수 있으며, 다른 스레드의 함수 호출이 정상적으로 종료될 경우 대기 상태의 스레드를 재개시키는 메시지를 처리하도록 지시할 수 있다. 또한, 순서위배 오류가 진단되지 않을 경우 스레드 제어 메시지는 동작하지 않는다.

[0052] 스레드 관리 유닛(Thread Management, TM)(310) 및 건전성 모니터링 유닛(Health Monitor, HM)(320)를 포함하는 시뮬레이션 통합 모듈형 기기(SIMA)(300)는 오류 및 스레드 재개 시점을 전달받으며 해당 상태에 따른 스레드 제어 명령을 수행할 수 있다. 즉, 시뮬레이션 통합 모듈형 기기(SIMA)(300)는 진단 결과에 따라 스레드를 대기시키는 메시지 및 스레드 재개시키는 메시지를 전달받아 스레드를 관리하는 역할에 해당된다.

[0053] 건전성 모니터링 유닛(HM)(320)은 ARINC 653 HM(Health Monitor)을 사용함으로써, 오류가 다른 소프트웨어에 전파되지 않도록 하며 건전성을 관리할 수 있다. 즉, 자율 수리 모듈(ORS)(200)에서 진단된 오류가 따라 다른 소프트웨어에 전파되지 않도록 하여 건전성을 관리하기 위함이다.

[0054] 건전성 모니터링 유닛(HM)(320)은 오류의 수준에 따라 처리 방식이 설정될 수 있다. 또한, 프로세스 상에서 발생한 오류일 경우 사용자가 정의한 오류 처리기(error handler)를 호출할 수 있다.

[0055] 본 발명의 일 실시예에 따르면, 건전성 모니터링 유닛(HM)(320)은 진단 유닛(DE)(210)으로부터 순서위배 오류 및 스레드 재개 시점을 전달받으면 처리 유닛(TE)(220)을 호출한다.

[0056] 스레드 관리 유닛(TM)(310)은 처리 유닛(TE)(220)의 메시지를 전달받아 오류가 진단된 스레드를 대기상태로 변경하거나 대기상태의 스레드를 재개하도록 스레드를 관리한다.

[0057] 도 3은 본 발명에 일 실시에 따른 항공기 소프트웨어의 건전성 관리 방법의 순서를 도시한 도면이다.

[0058] 본 발명의 일 실시예로, 응용 프로그램 모듈(100)은 ARINC 653 기반 항공기 소프트웨어의 함수 이름(method name)을 진단 유닛(Diagnosis Engine, DE)(210)에 전송할 수 있다(S100). 이때, 함수에는 공유변수가 포함될 수 있다.

[0059] 진단 유닛(DE)(210)은 사용자가 기설정한 함수 호출 순서(Type State Automaton, TSA) 및 수행 정보를 비교하여 순서위배 오류를 진단할 수 있다(S200). 순서위배 오류 또는 재개 시점이 진단되면, 진단 유닛(DE)(210)이 건전성 모니터링 유닛(HM)(320)으로 요청하여 오류 및 스레드 재개 시점을 전달하게 된다. 함수 호출 순서(TSA)에 대해서는 도 4 및 도 5에서 후술한다.

[0060] 건전성 모니터링 유닛(HM)(320)은 진단 유닛(DE)(210)으로부터 전달받은 오류 및 재개 시점을 모니터링하게 된다(S300). 모니터링한 후, 오류 및 재개 시점을 관리하고 처리 유닛(TE)(220)으로 전달하게 된다.

[0061] 처리 유닛(TE)(220)은 건전성 모니터링 유닛(HM)(320)에 의해 호출되어 오류 및 재개 시점에 따른 메시지 지시를 처리하도록 한다(S400). 그리고, 상기 오류 및 재개 시점에 따른 메시지 지시를 스레드 관리 유닛(TM)(310)으로 전달하게 된다.

[0062] 오류가 진단되면 스레드 관리 유닛(TM)(310)은 오류에 따른 대기 메시지 지시로 순서위배 오류가 발생한 스레드를 대기하도록 스레드를 관리한다(S500). 이때, 스레드 관리 유닛(TM)(310)은 처리 유닛(TE)(220)으로 스레드가 대기하는 wait call를 요청하게 된다.

[0063] 반면, 다른 스레드의 함수 호출이 정상적으로 종료되면 스레드 관리 유닛(TM)(310)은 스레드 관리 유닛(TM)(310)으로부터 signal call을 요청하여 대기 상태의 스레드가 재개하도록 관리한다.

[0064] 그리고 스레드 관리 유닛(TM)(310)은 관리된 스레드에 따라 제어명령을 수행하기 위해 응용 프로그램 모듈(100)에서 스레드를 제어한다.

[0065] 도 4a, 4b 및 도5는 본 발명에 일 실시에 따른 함수 호출 순서(TSA)를 설명하기 위한 예시도이다.

[0066] 도 4a는 2개의 스레드(T1, T2) 및 4개의 함수(init, start, stop, destroy)를 가지는 소스코드가 도시된 예시

도이며, 도 4b는 각 함수가 호출되어 함수 호출 순서(TSA)에 따라 변하는 상태(status) 정보들을 도시한 예시도이다. 그리고, 도 5는 TSA는 도 4b의 함수 호출 순서(TSA)를 각 함수에 대해 정리한 표이다.

[0067] 함수 호출 순서(TSA)는 항공기 소프트웨어 시스템에서 오류를 진단하기 위해 사용되는 정보로, 사용자에 의해 미리 정의된 함수 호출 순서에 해당한다. 도 5에 도시된 바와 같이, M1, M2, M3, M4와 같이 함수 이름(method_name) 각각에 함수 호출 순서가 정의시킬 수 있다.

[0068] 본 발명의 일 실시예에 따르면, 함수 호출 순서(TSA)가 특정 상태에 따라 각 함수가 호출되었을 때, M1: I→init→U, M2: U→start→L, M3: L→stop→U, M4: U→destroy→D와 같이 메소드(method) 이름에 따른 상태 정보들을 가질 수 있다.

[0069] 이러한, 함수 호출 순서(TSA)는 자율 수리 모듈(ORS)(200)이 오류를 진단하기 위해 사용될 수 있다.

[0070] 도 6은 본 발명에 일 실시에 따른 진단 유닛(DE)(210)의 알고리즘 소스코드이다. 도 6에 도시된 바와 같이, 진단 유닛(DE)(210)은 응용 프로그램 모듈(100)에서 수행될 함수 이름(method_name)을 입력받아 getState()를 호출하며, 해당 함수는 현재 상태(state)와 함수 이름(method_name)을 매개변수로 가지게 된다.

[0071] 진단 유닛(DE)(210)은 오류가 없는 경우에는 현재 상태에서 해당 함수가 호출되었을 때, 변하는 상태를 반환하고 오류가 있는 경우에는 "bad"를 반환한다. 그리고, 진단 유닛(DE)(210)은 result_1, result_2의 if문을 통해 오류 및 스레드 재개 시점을 전달받아 진단하게 된다.

[0072] 첫 번째 if문(result_1)은 순서위배 오류를 진단하는 단계에 해당한다. 진단 유닛(DE)(210)에서 반환 값이 "bad"일 경우, HM(health Monitoring) call을 호출하여 건전성 모니터링 유닛(HM)(320)에 오류를 전달한다. 오류를 전달받은 건전성 모니터링 유닛(HM)(320)은 처리 유닛(TE)(220)을 호출하고, 스레드 관리 유닛(TM)(310)은 호출된 처리 유닛(TE)(220)에서 보낸 메시지를 전달받으면 pthread_cond_wait()을 호출하여 해당 스레드를 대기시킨다.

[0073] 두 번째 if문(result_2)은 멈춰있는 스레드의 함수가 올바른 함수의 수행 후에 재개될 시점을 진단하는 단계에 해당한다. 진단 유닛(DE)(210)에서 반환 값이 "bad"가 아닐 경우, HM call을 호출하여 건전성 모니터링 유닛(HM)(320)에 스레드 재개 시점을 보고한다. 재개시점을 보고받은 건전성 모니터링 유닛(HM)(320)은 처리 유닛(TE)(220)을 호출하고, 스레드 관리 유닛(TM)(310)은 호출된 처리 유닛(TE)(220)에서 보낸 메시지를 전달받으면 pthread_cond_signal()을 호출하여 해당 스레드를 재개한다.

[0074] 도 7은 본 발명에 일 실시에 따른 처리 유닛(TE)(220)의 알고리즘 소스코드이다.

[0075] 본 발명의 일 실시예에 따르면, 처리 유닛(TE)(220)은 건전성 모니터링 유닛(HM)(320)에 의해 호출되는 프로세스이다. 처리 유닛(TE)(220)은 오류 및 스레드 재개 시점을 전달받은 건전성 모니터링 유닛(HM)(320)에 의해 호출되며, 전달받은 오류 및 스레드 재개 시점에 따라 해당되는 메시지를 스레드 관리 유닛(TM)(310)으로 전달한다.

[0076] 도 7에 도시된 바와 같이, 처리 유닛(TE)(220)은 전달받은 오류 및 스레드 재개 시점에 대한 정보(status)를 획득하기 위해 HM call인 GET_ERROR_STATUS()를 호출한다. 해당 함수는 전달 정보를 반환한다. 처리 유닛(TE)(220)에서 반환 값이 "order_violation"인 경우에는 스레드를 대기시키기 위한 메시지를 전송하고, 반환 값이 "trigger"인 경우에는 스레드를 재개시키기 위한 메시지를 전송한다.

[0077] 도 8은 도 4a, 4b 및 도 5의 프로그램에서 발생할 수 있는 함수 호출에 따라 순서위배 오류를 수리하는 예시도이다.

[0078] 도 8에 도시된 바와 같이, execution column은 도 4a, 4b에서 destroy()가 stop()보다 먼저 호출되는 수행 순서를 나타낸다. input column은 해당 시점의 함수 호출 시 반환되는 상태(state)를 나타낸다. output of module column은 해당 시점의 오류 진단 및 스레드 상태를 나타낸다.

[0079] 도 5의 함수 호출 순서(TSA)에 따라 A시점은 init()이 수행되기 직전으로 현재 상태 "I"에서 init()을 호출하면 "U"라는 정상적인 상태가 반환되므로, init()의 호출은 오류가 없는 정상적인 순서이다.

[0080] B시점은 M1 함수에 따라 start()가 호출되기 직전으로 현재 상태 "U"에서 start()을 호출하면 "L"의 상태로 반환되므로, A시점과 동일하게 정상적인 순서로 판단된다.

[0081] C시점은 M2 함수에 따라 destroy()가 호출되기 직전으로 현재 상태 "L"에서 destroy()를 호출하면 “bad"가 반

환되므로 순서위배 오류로 판단된다. 따라서 해당 두 번째 스레드(thread 2)를 대기상태로 변경한다.

[0082] D시점은 M2 함수에 따라 stop()이 호출되기 직전으로 현재 상태 "L"에서 stop()을 호출하면 "U"의 상태로 반환된다. 즉, stop()은 A, B시점과 동일하게 정상적인 순서로 판단된다.

[0083] E시점은 stop()이 호출된 직후의 시점으로 현재 상태 "U"에서 destroy()를 호출하면 "D"라는 정상적인 상태가 반환되므로, destroy()의 재개 시점으로 판단된다. 그러므로 두 번째 스레드(thread 2)를 재개시켜 destroy()가 호출한다.

[0084] 결과적으로 네 개의 함수(M1, M2, M3, M4)가 올바른 순서로 수행되게 하여 순서위배 오류를 수리한다.

[0085] 이러한 함수 호출 순서(TSA)는 도 8과 같이 테이블 형식의 텍스트 파일로 저장될 수 있다.

[0086] 도 9은 도 4a, 4b 및 도 5의 함수 호출 순서(TSA)를 테이블 형식으로 구현한 표이다.

[0087] I, U, L, D의 State column은 현재 상태(state)를 나타내고, init, start, stop, destroy의 column은 현재 상태에서 각 함수가 호출되었을 때의 상태를 나타낸다. initial_state는 초기 상태(I)를 구별하기 위한 column에 해당한다.

[0088] 도 4b를 참고하면, 현재 상태 "I"에서 init()을 호출하면 "U"라는 정상적인 상태로 반환되므로, start(), stop(), destroy()의 호출은 "bad" 가 반환된다. 현재 상태 "U"에서 start()를 호출하면 "U"와 "D"라는 정상적인 상태로 반환되어 init(), stop()의 호출은 "bad" 가 반환되고, 현재 상태 "L"에서 stop()을 호출하면 "U"라는 정상적인 상태로 반환되어 init(), start(), destroy()의 호출은 "bad" 가 반환된다. 그리고, 현재 상태 "D"에서 init(), start(), stop(), destroy()의 호출은 "bad" 가 반환된다.

[0089] 또한, 현재 상태 "I"는 초기 상태에 해당되므로 initial_state가 YES(Y)를 나타내고, 현재 상태 "U", "L", "D"는 초기 상태가 아니므로 initial_state가 NO(N)를 나타낸다.

[0091] 다음은, 앞서 설명한 항공기 소프트웨어의 건전성 관리 시스템(1000)을 구현하기 위해 시험예를 통해 운영환경 및 코딩·통합환경에 대해 설명하고자 한다.

[0092] 시험예

[0093] 본 발명에서 명시한 시스템 및 방법에 따라 자율 수리 모듈(ORS)(200)을 구현하기 위해 코딩 및 통합 환경은 Visual Studio Code 1.37.1 프로그래밍 툴 및 gcc 5.4.0 컴파일러를 사용할 수 있다. 코드 삽입에는 Low Level Virtual Machine(LLVM) 6.0.0, 하드웨어 운영 환경은 Intel Xeon E5-2650 2.30GHz CPU 및 64GB RAM을 사용할 수 있다. 그리고, Ubuntu 16.04 LTS-64bit OS에 linux 4.14.139-rt66 kernel 및 Simulated Integrated Modular Avionics (SIMA) 1.3.1.0을 세팅하여 ARINC 653 시뮬레이션 환경을 구성할 수 있다.

[0094] 이러한 구성으로 항공기 소프트웨어의 건전성 관리 시스템(1000)의 시스템 구현은 함수 호출 순서(TSA)를 사용하는 자율 수리 모듈(ORS)(200), 시뮬레이션 통합 모듈형 기기(SIMA)(300), IPC(InterProcess Communication, 프로세스간 통신)를 포함할 수 있다.

[0095] 도 10은 본 발명에 일 실시에 따른 시뮬레이션 통합 모듈형 기기(SIMA)(300)에서 출력되는 화면 예시도이다.

[0096] 시뮬레이션 통합 모듈형 기기(SIMA)(300)는 일반 OS에서 ARINC 653 규격을 만족하는 기능을 이용하기 위해 사용하는 시뮬레이터에 해당한다. 도 10에서 도시된 바와 같이, 시뮬레이션 통합 모듈형 기기(SIMA)(300)는 여러 개의 파티션을 가지는 시스템의 화면을 출력할 수 있다.

[0097] 시뮬레이션 통합 모듈형 기기(SIMA)(300)는 파티셔닝 환경을 그래픽으로 출력해주는 Simout 도구를 사용할 수 있다. 도 10의 상단에는 두 개의 파티션으로 나눌 수 있으며, 좌측의 첫 번째 파티션에서 응용 프로그램 모듈(100) 및 처리 유닛(TE)(220)을 위한 프로세스를 확인할 수 있다.

[0098] 본 발명의 일 실시예에 따르면, 시뮬레이션 통합 모듈형 기기(SIMA)(300)는 C언어로 개발되었기 때문에 객체지향 개념을 사용하기 어려우므로, 응용 프로그램 모듈(100)을 시뮬레이션 통합 모듈형 기기(SIMA)(300)에 포팅(porting)하지 못하고 IPC(InterProcess Communication, 프로세스간 통신)로 통신할 수 있다. 여기서, IPC는 실행 프로세스 간에 통신을 가능케하는 메커니즘에 해당된다.

[0099] IPC는 응용 프로그램 모듈(100)에 대한 오류를 진단했을 때, 시뮬레이션 통합 모듈형 기기(SIMA)(300)로 오류의

진단을 전달하고 스레드 대기/재개 메시지를 전송하는 역할에 해당하며, message passing 방식을 사용할 수 있다.

[0100] 도 11은 본 발명에 일 실시에 따른 자율 수리 모듈(ORS)(200)의 동작을 확인할 수 있는 소스코드이다.

[0101] 도 11은 도 4a, 4b 및 도 5의 예시 프로그램을 사용한 것으로 shared_variable은 공유변수에 해당되며, init()부터 4개의 함수에 공유변수가 포함한다. t1_func() 및 t2_func()에 4개의 함수가 분배되어 두 개의 스레드로 수행된다.

[0102] 4가지 함수를 2개 및 4개의 스레드에 분배하여 호출될 수 있는 모든 순서 조합을 대상을 고려하면, 스레드가 2개인 경우에는 ORS를 적용할 수 없는 유형이 있다. 예를 들면, start() → init() → stop()와 같이 도 4a, 4b에서 T1에 포함된 3개의 함수 순서가 뒤바뀌는 경우에는 프로그램 코드의 접근사건 순서가 뒤바뀐 것으로 동시성 오류로 볼 수 없기 때문에 수리대상이 아니다.

[0103] 4가지 함수를 2개 및 4개의 스레드에 분배하여 호출될 수 있는 순서 조합의 모든 유형은 총 192개이고, 수리 대상이 아닌 유형 및 오류가 없는 유형을 제외한 50개의 유형을 모두 수리할 수 있다.

[0104] 도 12는 도 11의 수리할 수 있는 유형에 대한 출력 화면의 예시도이다.

[0105] 도 12는 도시된 바와 같이, ①의 init()과 ②의 start()는 정상적으로 수행되나, ③의 destroy()가 호출되어 순서위배 오류로 진단되고 해당 함수의 호출이 지연된다. 즉, ①②④⑤의 순서처럼 올바른 순서로 수행된 것을 확인할 수 있다.

[0106] 도 13은 본 발명의 효율성을 검증하기 위한 합성 프로그램의 유형을 설명하기 위한 표이다.

[0107] 합성 프로그램은 도 11의 소스코드에 대해 init(), start(), stop() 및 destroy() 이외에 fast_loop(), update_flight_mode(), input_euler_angle(), motors_output(), push() 함수를 추가하여 각 함수에 포함된 공유변수의 접근사건 개수는 최소 0개부터 최대 80개까지 포함할 수 있으며, 스레드의 fork/join을 반복할 수 있다.

[0108] 본 발명의 일 실시예에 따르면, 유형은 합성 프로그램의 스레드 수, 함수 개수, 그리고 함수의 순서 조합에 따라 나뉠 수 있다. 조합에 따른 각 유형은 스레드 수와 순서 조합에 상관없는 시간 오버헤드를 보이기 위함이다.

[0109] 스레드 수는 각 함수가 분배된 스레드의 개수이며, 함수 개수는 각 스레드에 분배된 함수의 개수에 해당한다. 그리고 순서는 함수 호출의 순서위배 오류를 유발하기 위해 임의적으로 조정한 순서이다. 이를 통해 공유변수 접근 횟수의 증가에 따른 시간 오버헤드를 측정할 수 있다.

[0110] 도 13에 도시된 바와 같이, case 1 내지 5의 유형에 따라 순서를 A부터 H까지의 알파벳으로 대체하여 나열하였다. case 1 내지 5의 유형을 대상으로 프로그램 내부 스레드 fork/join의 반복은 4,000번으로 고정하고, 공유변수 접근 횟수는 최소 5번부터 두 배씩 증가시키며 최대 640번까지 8종류의 개수를 사용하였다.

[0111] 도 14는 도 13의 각 유형을 측정한 수행 시간을 설명하기 위한 표이다.

[0112] 도 14에 도시된 바와 같이, 공유변수 접근 횟수가 5번이면 총 공유변수 접근 횟수는 20,000회, 640번이면 총 공유변수 접근 횟수가 2,560,000회 발생한다. 그리고, 총 함수 호출 횟수는 전체 32,000번으로 일정하다.

[0113] 종래의 원자성 위배 수리(Repairing-Atomicity Violation, Repairing-AV)의 진단 횟수는 총 공유변수 접근 횟수와 일치한다. 스레드가 2개인 case 1, case 2, case 3에서 Repairing-AV의 수행 시간은 공유변수 접근 횟수에 따라 약 0.3초부터 최대 약 5초까지 증가하였다. 그리고 스레드가 8개인 case 4와 case 5에서 Repairing-AV의 수행 시간은 약 0.9초부터 최대 약 5.5초까지 증가하였다. 따라서, 공유변수 접근 횟수가 증가하면 Repairing-AV의 진단 횟수도 같이 증가하기 때문에 프로그램 수행 시간도 급격히 증가함을 확인할 수 있다.

[0114] 본 발명의 일 실시예에 따른 자율 수리 모듈(ORS)(200)의 진단 횟수는 총 함수 호출 횟수와 일치한다. 스레드가 2개인 case 1, case 2, case 3인 경우 약 0.8초의 일정한 수행 시간을 확인할 수 있다. 그리고 스레드가 8개인 case 4와 case 5에서 자율 수리 모듈(ORS)(200)은 약 1.5초의 일정한 수행 시간을 보였다. ORS의 진단 횟수는 총 함수 호출 횟수와 일치하기 때문에 공유변수 접근 횟수가 증가하더라도 총 함수 호출 횟수는 32,000번으로 고정적이다. 따라서, 자율 수리 모듈(ORS)(200)이 적용된 경우 공유변수 접근 횟수에 영향을 받지 않고 일정한 오버헤드로 진단 및 조치가 가능하다.

[0115] 도 15는 도 14의 case 1에 대해 종래의 Repairing-AV과 본 발명의 자율 수리 모듈(ORS)(200)의 수행 시간 비교

그래프이다.

[0116] x축은 공유변수 접근 횟수이고 y축은 프로그램 수행 시간이다. 종래의 Repairing-AV는 총 공유변수 접근 횟수에 비례하기 때문에 수행 시간이 지수적으로 증가하나, 자율 수리 모듈(ORS)(200)은 총 함수 호출 횟수에 비례하기 때문에 수행 시간이 일정한 것을 확인할 수 있다.

[0117] 공유변수 접근 횟수가 5번에서 20번까지인 구간에서는 총 공유변수 접근 횟수의 증가량이 적기 때문에 종래의 Repairing-AV의 오버헤드가 낮으나, 자율 수리 모듈(ORS)(200)은 종래의 Repairing-AV 대비 약 두 배의 오버헤드를 가지는 것을 볼 수 있다.

[0118] 그러나, 20번일 때부터 종래의 Repairing-AV의 수행 시간이 증가하기 시작하여 약 60번일 때 ORS와 수행 시간의 교차가 일어난다. 마지막으로 약 60번에서 640번까지의 구간에서는 종래의 Repairing-AV의 수행 시간이 급격하게 증가함을 확인할 수 있다.

[0119] 실험 결과를 분석해 볼 때, 두 기법의 수행 시간이 일치하는 약 60번의 시점에 종래의 Repairing-AV의 진단 횟수는 약 240,000회, 자율 수리 모듈(ORS)(200)의 진단 횟수는 32,000회이다. 따라서 자율 수리 모듈(ORS)(200)로 진단하는 시간은 종래의 Repairing-AV보다 약 8배 높음을 알 수 있다. 따라서, 총 공유변수 접근 횟수가 총 함수 호출 횟수의 약 8배보다 낮은 경우에는 Repairing-AV를 사용하는 것이 효율적이며, 높은 경우에는 ORS를 사용하는 것이 순서위배 오류를 수리하는 것이 더 효율적인 것을 알 수 있다.

[0120] 이와 같이 본 발명에 따르면, 항공기 소프트웨어의 순서위배를 자율적으로 수리하기 위한 건전성 관리 시스템 및 방법은 접근사건이 포함된 함수 간의 순서위배 오류를 진단하고 해당 함수가 포함된 스레드를 지연시켜 올바른 순서로 수행되도록 처리할 수 있다. 이러한 구성에 따라 공유변수 접근 횟수에 영향을 받지 않고 일정한 오버헤드로 진단 및 조치가 가능하다. 또한 개발 단계에서 제거되지 않은 오류를 수행 중에 진단하고 조치할 수 있다.

[0121] 이상과 같이 본 발명의 도시된 실시 예를 참고하여 설명하고 있으나, 이는 예시적인 것들에 불과하며, 본 발명이 속하는 기술 분야의 통상의 지식을 가진 자라면 본 발명의 요지 및 범위에 벗어나지 않으면서도 다양한 변형, 변경 및 균등한 타 실시 예들이 가능하다는 것을 명백하게 알 수 있을 것이다. 따라서 본 발명의 진정한 기술적 보호 범위는 첨부된 청구범위의 기술적인 사상에 의해 정해져야 할 것이다.

부호의 설명

[0122] 100: 응용 프로그램 모듈

200: 자율 수리 모듈

210: 진단 유닛

220: 처리 유닛

300: 시뮬레이션 통합 모듈형 기기

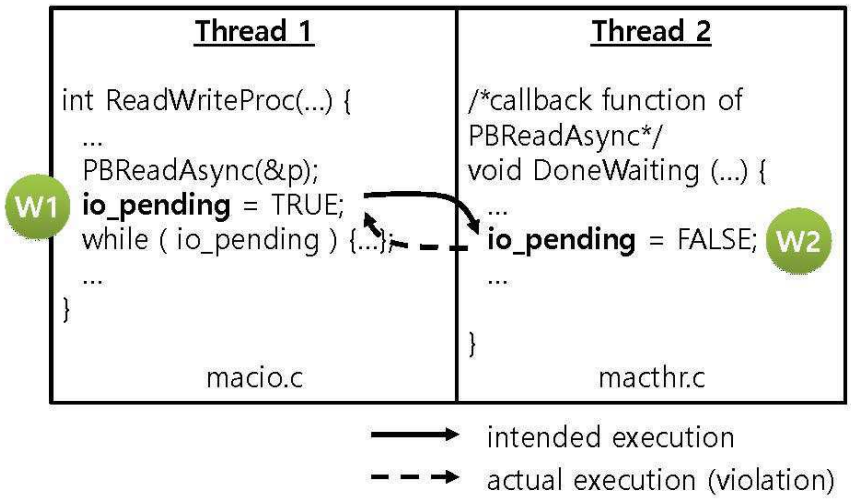
310: 스레드 관리 유닛

320: 건전성 모니터링 유닛

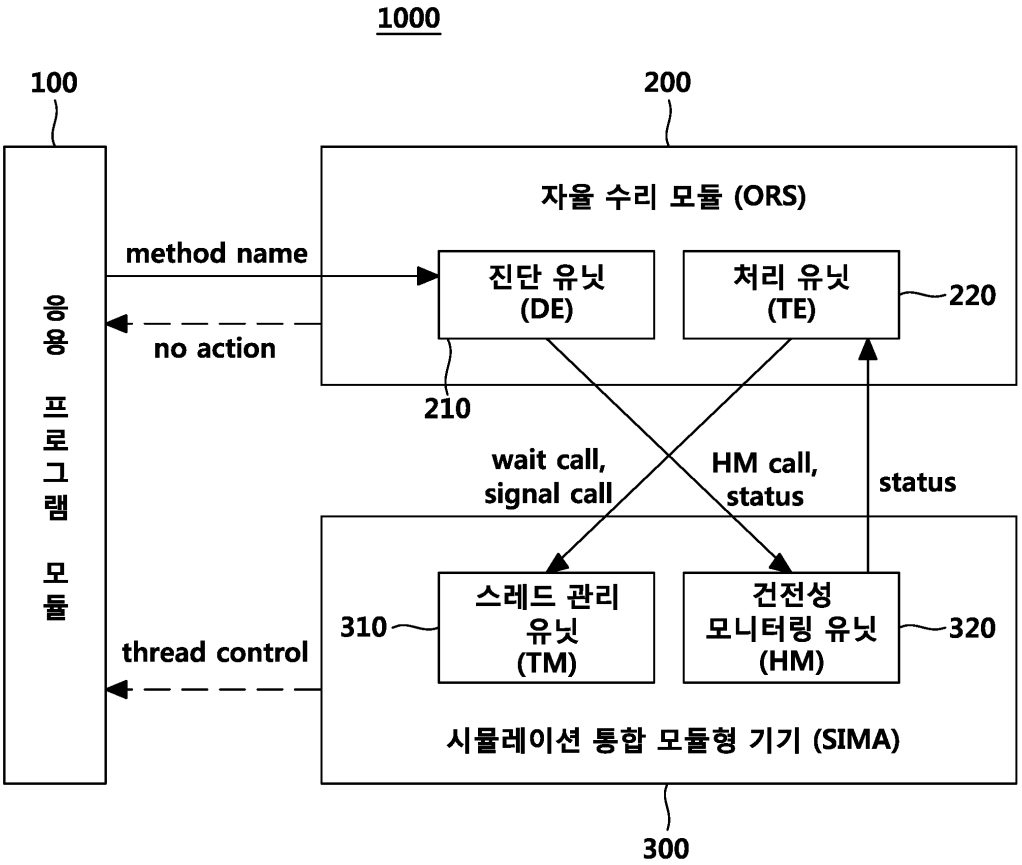
1000: 항공기 소프트웨어의 건전성 관리 시스템

도면

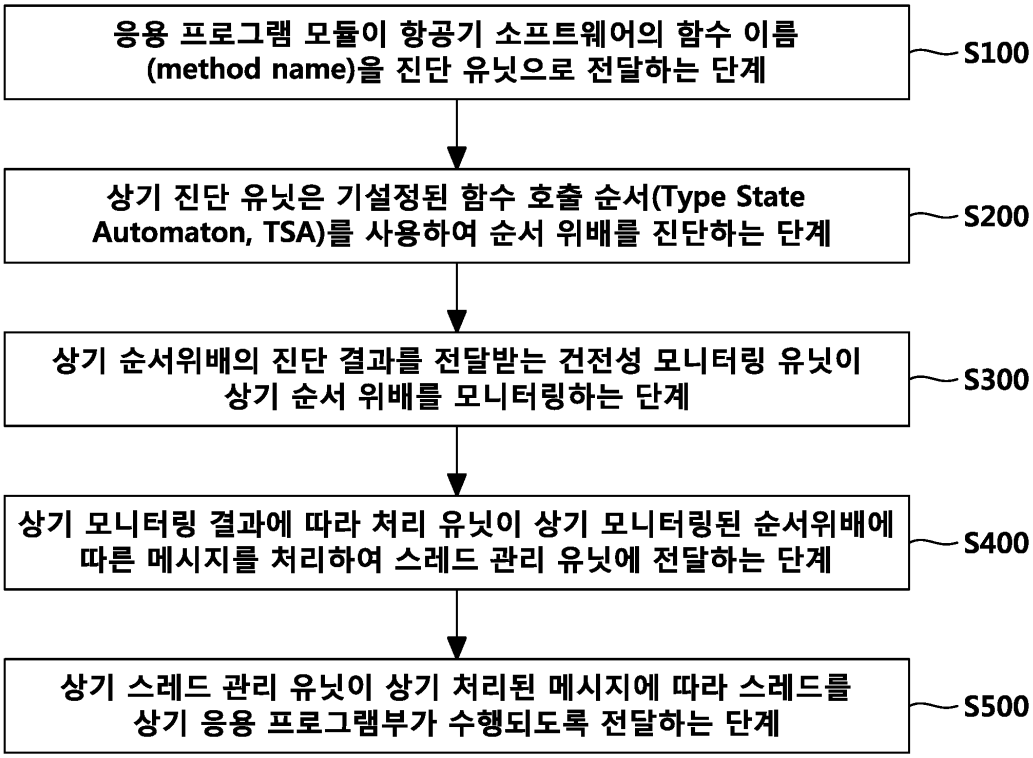
도면1


Thread 1
Thread 2
int ReadWriteProc(...) {
...
PBReadAsync(&p);
io_pending = TRUE;
while (io_pending) {...}
...
}
macio.c
/*callback function of
PBReadAsync*/
void DoneWaiting (...) {
...
io_pending = FALSE;
...
}
macthr.c
W1
W2
intended execution
actual execution (violation)

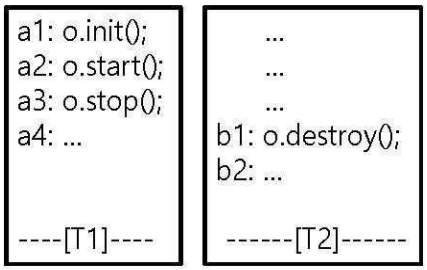
도면2


1000
100
200
응용 프로그램 모듈
자율 수리 모듈 (ORS)
method name
no action
진단 유닛 (DE)
처리 유닛 (TE)
220
210
wait call, signal call
HM call, status
status
310
스레드 관리 유닛 (TM)
건전성 모니터링 유닛 (HM)
320
thread control
시뮬레이션 통합 모듈형 기기 (SIMA)
300

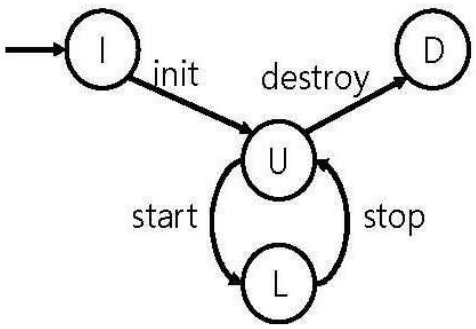
도면3



도면4a



도면4b



도면5

State	Method	State
I	M1	U
U	M2	L
L	M3	U
U	M4	D

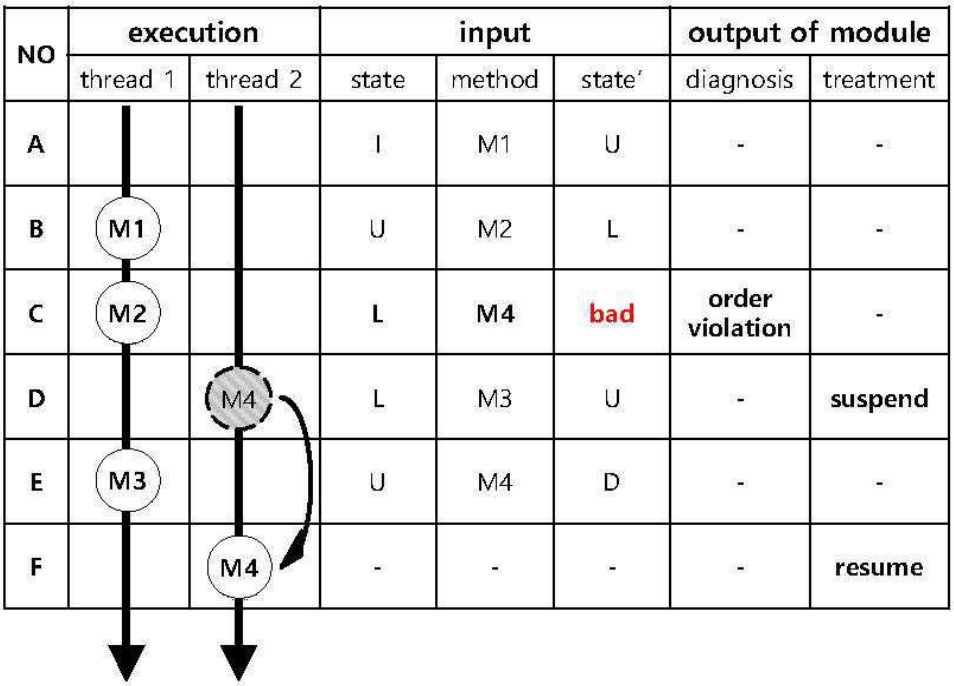
도면6

```
D-Engine algorithm:
procedure D-Engine(method_name)
  string result_1 = getState(state, method_name)
    if result_1 == “bad” then
      RAISE_APPLICATION_ERROR(order_violation)
      RECEIVE_BUFFER(suspend_buffer)
      pthread_cond_wait()
    end if
  string result_2 = getState(state, method_name)
    if result_2 != “bad” then
      RAISE_APPLICATION_ERROR(trigger)
      RECEIVE_BUFFER(resume_buffer)
      pthread_cond_signal()
    end if
end procedure
```

도면7

```
T-Engine algorithm:
procedure T-Engine()
  GET_ERROR_STATUS(status)
  string status = status.MESSAGE
  if status == “order_violation” then
    SEND_BUFFER(suspend_buffer)
  end if
  else if status == “trigger” then
    SEND_BUFFER(resume_buffer)
  end if
end procedure
```

도면8

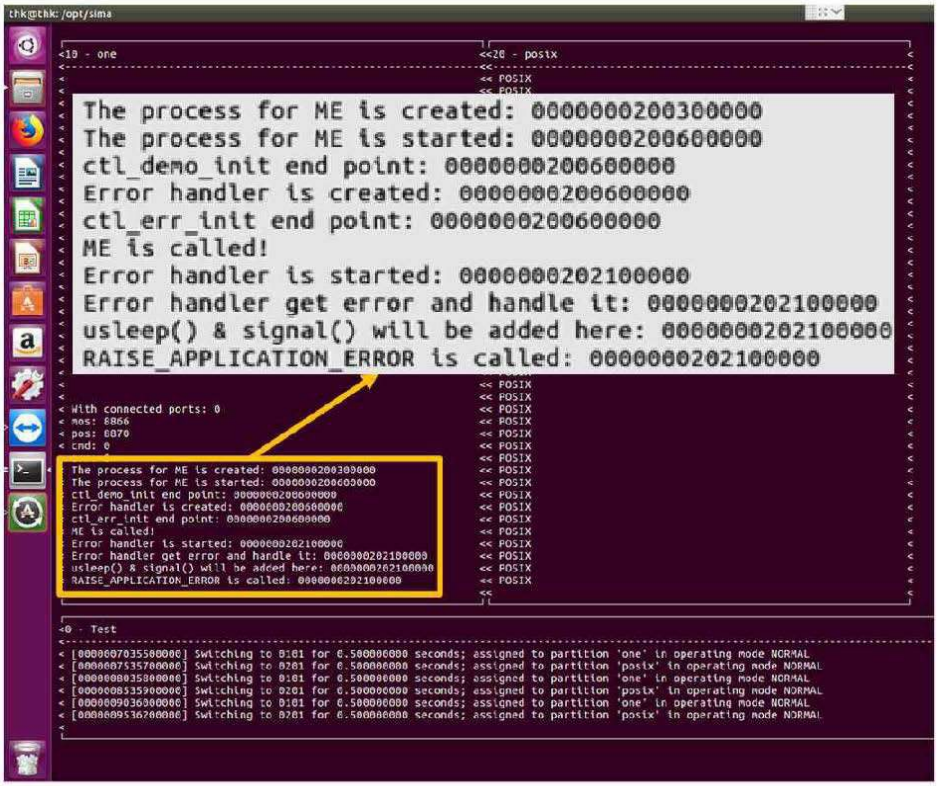


NO	execution		input			output of module	
	thread 1	thread 2	state	method	state'	diagnosis	treatment
A			I	M1	U	-	-
B	M1		U	M2	L	-	-
C	M2		L	M4	bad	order violation	-
D		M4	L	M3	U	-	suspend
E	M3		U	M4	D	-	-
F		M4	-	-	-	-	resume

도면9

state	init	start	stop	destroy	initial_state
I	U	bad	bad	bad	Y
U	bad	L	bad	D	N
L	bad	bad	U	bad	N
D	bad	bad	bad	bad	N

도면10



도면11

```
int Object::shared_variable = 0;

void Object::init()
{…
  shared_variable = 1;
}
…
void *t1_func(void *data)
{…
  pthread_mutex_lock(&sync_mutex);
  Object::init();
  pthread_mutex_unlock(&sync_mutex);
   …
void *t2_func(void *data)
{…
  pthread_mutex_lock(&sync_mutex);
  Object::destroy();
  pthread_mutex_unlock(&sync_mutex);
```

도면12

```
root@thk:/home/thk/SWHM_implementation/TSA# ./R2
thread1 id: 2633094912
(2633094912) pre_MRC(init)
(2633094912) pre_MRC: Initial state: I
(2633094912) pre_MRC: Current state: U
(2633094912) init() ①
(2633094912) post_MRC()
(2633094912) pre_MRC(start)
(2633094912) pre_MRC: Current state: L
(2633094912) start() ②
(2633094912) post_MRC()
thread2 id: 2624702208
(2624702208) pre_MRC(destroy) ③
(2624702208) pre_MRC: Order violation
(2624702208) pre_MRC: Send a message to SIMA (report)
(2624702208) (2633094912) pre_MRC(stop)
(2633094912) pre_MRC: Current state: U
(2633094912) stop() ④
(2633094912) post_MRC()
(2633094912) post_MRC: Send a message to repairing server (trigger)
(2633094912) pre_MRC: Read Data (0 for suspend): 0
(2624702208) pre_MRC: pthread_cond_wait()
post_MRC: Read Data (2 for resume): 2
(2633094912) post_MRC: pthread_cond_signal()
(2624702208) pre_MRC: Current state: D
(2624702208) destroy() ⑤
(2624702208) post_MRC()
```

도면13

Case	# of threads	# of methods	Order
case 1	2	7:1	ABCDEFHG
case 2			ABCDEHFG
case 3		4:4	ABCEDFGH
case 4	8	1:1:1:1:1:1:1:1	BACDEFGH
case 5			CABDEFGH

도면14

# of accesses	# of functions	# of total accesses	Case 1		Case 2		Case 3		Case 4		Case 5	
			Repairing -AV	ORS	Repairing -AV	ORS	Repairing -AV	ORS	Repairing -AV	ORS	Repairing -AV	ORS
5	32,000	20,000	315.29	725.31	280.97	648.20	335.69	837.32	944.39	1578.01	952.89	1524.63
10		40,000	391.45	790.26	363.24	825.43	323.37	828.96	1090.82	1544.44	1066.17	1533.89
20		80,000	423.76	770.29	331.15	772.10	442.24	847.63	1023.49	1518.74	1132.85	1507.09
40		160,000	599.11	690.73	667.94	863.94	598.58	962.47	1355.28	1718.09	1406.14	1598.91
80		320,000	871.07	734.91	870.74	791.37	925.02	849.16	1532.08	1496.20	1654.49	1512.87
160		640,000	1397.07	745.71	1457.51	805.47	1449.60	844.60	2220.98	1527.47	2274.79	1536.38
320		1,280,000	2669.01	771.29	2672.45	748.10	2696.24	857.77	3511.05	1526.69	3531.41	1540.23
640		2,560,000	4943.94	750.66	4982.22	776.81	4873.74	812.85	5527.96	1605.81	5637.81	1634.12

도면15