C 언어를 이용한 OSEK/VDX 기반 운영 체제의 모델링 및 검증 기법1)

정유희, 박성준, 최윤자
경북대학교 컴퓨터학부
e-mail : jungyh0218@gmail.com, sjpark0233@gmail.com, yuchoi76@knu.ac.kr

An Approach to Modeling and Verifying OSEK/VDX-based OS with C language

Yoohee Chung, Seongjun Park, Yunja Choi
Dept. of Computer Science and Engineering, Kyungpook National University

요 약

내장형 소프트웨어의 검증은 운영체제와 응용 소프트웨어에 대해 동시적으로 수행되어야 한다. 선행연구에서는 운영체제를 정형적으로 모델링하고 이를 이용한 내장형 소프트웨어 안전성 검증 기법을 제안하였다. 본 연구에서는 선행 연구에서 Promela로 작성되었던 정형 모델을 참조하여 C 언어로의 변환 규칙을 정의함으로써 구현에 독립적인 모델을 작성하였다. 작성된 검증 모델을 CBMC 모델 체커로 검증한 결과 요구사항으로부터 도출된 35개의 안전성 속성 중 가변 속성을 제외한 21개에 대하여 21개를 모두 만족함을 입증하였다.

1. 서론

내장형 소프트웨어의 검증을 위하여 다양한 모델 검증 도구와 모델링 언어가 사용되어 왔다. 내장형 소프트웨어는 일반적으로 운영체제와 응용 소프트웨어가 분리되지 않으므로 동시에 검증해야 한다. 그런데 내장형 소프트웨어를 기존의 모델링 언어를 사용해 검증하는 경우 응용 소프트웨어를 운영체제를 모델링한 언어로 모델링하거나, 운영체제 모델의 코드를 구현해야 한다는 단점이 존재한다. 이러한 점을 개선하기 위하여 본 연구에서는 응용 소프트웨어의 구현에 사용되는 C 언어로 운영체제를 모델링한 후 해당 모델의 정확성을 검증하는 기법을 제시하고자 한다. 본 연구에서 사용된 모델링 대상 프로그램은 차량 전장용 운영체제 표준인 OSEK/VDX를 준수하는 운영체제(OSEK OS)이며[1], 모델의 검증에는 assert로 표현된 제약조건과 C/C++ 대상 바운디드 모델 체킹 도구인 CBMC가 사용되었다.[2] 검증 단계에서는 OSEK/VDX 표준에서 선정한 제약조건 중 시스템 불변 속성 및 특정 속성은 모두 assert로 표현 및 CBMC로 검증 가능함을 확인하였다. 또한 C 언어의 특성으로 인해 발생하는 모델의 코드 안전성 문제를 발견할 수 있었다.

1) "본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT연구센터육성 지원사업의 연구결과로 수행되었음" (IITP-2016-H8501161012000100 2)

2. 연구 배경 및 접근 방법

기존 연구에서 내장형 소프트웨어를 검증하기 위해 Spin, NuSMV와 같은 다양한 모델 체커가 사용되었다. 그러나 이와 같은 방법을 사용할 시 검증 대상인 운영체제-응용 소프트웨어 간 표현 언어가 통일되어야 한다는 문제가 발생하게 되었다. 이를 해결하기 위하여 본 연구에서는 응용 소프트웨어의 구현에 사용되는 C 언어로 운영체제를 모델링하고 이를 검증할 수 있는 방법을 제시했다..

연구를 위한 모델링 대상으로는 차량 전장용 운영체제 표준인 OSEK/VDX 2.2.3 기반 운영체제를 선정하였다. OSEK/VDX는 분산된 내장형 컨트롤 유닛을 제어하는 단일 프로세서 운영체제로 ISO/ANSI-C 와 유사한 형태의 문법으로 시스템 서비스를 기술하고 있으므로 본 연구에 적용하기 용이하다.[1]

채택된 OSEK/VDX 기반 내장형 소프트웨어의 검증은 **그림 1**과 같은 절차를 거친다. 먼저 C언어로 OSEK OS를 모델링하기에 앞서, 적정한 수준의 추상도를 유지하고 구현과 독립적인 모델을 만들기 위해 기존에 작성된 PROMELA 언어로 작성된 모델을 참고하였다. PROMELA 모델과 OSEK/VDX 표준 문서를 참고하여 완성된 C 언어 모델에 대한 검증은 C/C++ 을 대상으로 하는 바운디드 모델 체킹 도구

인 CBMC를 사용했다.

검증 결과 발견한 제약조건 위배를 분석하여 모델의 결함으로 인한 제약조건 위배로 판단되면 이를 피드백 삼아 모델을 반복 수정한다. 최종적으로 모델의 정확성이 확보되었다고 판단될 경우 응용프로그램 코드와 함께 CBMC로 검증하여 전체 내장형 소프트웨어의 검증 결과를 도출하게 된다.

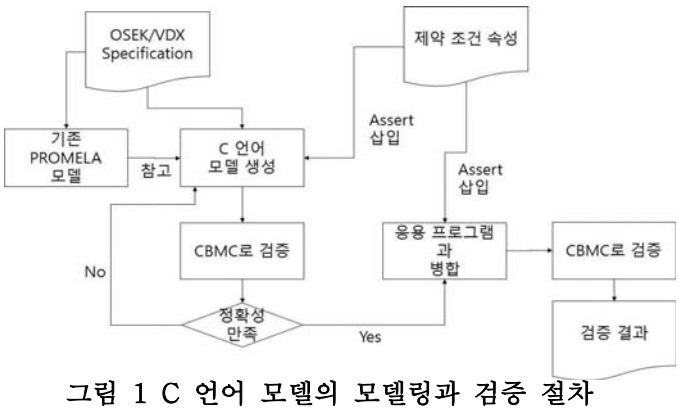

그림 1 C 언어 모델의 모델링과 검증 절차

3. 모델의 정의

OSEK OS의 모델을 기술하기에 앞서 검증 대상으로 삼고자 하는 범위에 대한 정확한 정의가 필요하다.

OSEK/VDX에서 실행 우선순위가 낮은 단계부터 시작해 높은 단계로 모델을 확장하며 구현한다. 현재는 논리적 태스크 스케줄링 단계까지 모델 범위에 포함하였다.

OSEK/VDX는 태스크보다 우선순위가 높은 사용자 행위를 정의하고 및 설정 파일(OIL 파일)을 반영할 수 있도록 훅 루틴(Hook routine)이라는 개념을 제공한다. 훅 루틴은 시스템 시작, 시스템 중단, 디버깅 및 에러 핸들링에 사용된다. 현재 모델은 시스템 시작을 위한 System startup 훅 루틴만 모델링 대상에 포함한다. 나머지 훅 루틴은 모델의 범위가 확장될 경우 추가로 구현해야 한다.

4. PROMELA - C 언어 모델 간 변환 규칙

구현에 사용되는 C 언어로 모델을 작성할 경우 가장 어려운 문제는 적절한 수준의 추상도를 유지하는 것이다. 본 연구에서는 구현과 독립적인 추상성을 갖춘 모델을 작성하기 위해 PROMELA 모델을 참조하였으며 이를 위한 C 언어로의 변환 규칙을 정의 하였다.

4.1. 문법적 변환 규칙

PROMELA의 핵심 문법은 C 언어의 문법과 유사한 시맨틱을 가지고 있다. 그림 2는 둘 간의 핵심 문법의 변환 규칙을 정리한 결과이다.

D3의 inline은 인라인 프로시저를 표현하는 키워드이다. PROMELA 모델에서는 OSEK OS의 프로시저와 훅 루틴은 모두 inline으로 표현되었다. inline은 C 언어 모델에서는 void 함수로 변환되었으며 이 때 inline 매개변수의 특성을 반영하기 위해 참조에 의해 전달되도록 한다. 단 flag와 같은 상수가 전달될 때는 값에 의한 전달을 하도록 예외 규칙을 둔다.

d_step 키워드는 C 언어가 결정적인 언어이므로 D5와 같이 그대로 표현한다.

PROMELA의 if문은 기본적으로 비결정적으로 실행된다. PROMELA OSEK OS에서도 이 특징을 반영하여 두 개 이상의 에러가 동시에 발생할 시 어떤 에러의 코드를 반환할 지를 비결정적으로 선택하도록 모델링되었다. C 언어 모델에서는 먼저 발생한 오류의 코드를 반환하도록 변환하였다. OSEK/VDX 표준은 다수의 에러 발생시 처리 방법을 명시하지 않으므로 두 방법 모두 유효하며 C 언어 모델에서는 좀 더 단순한 방식을 채택하였다.[1]

D8에 나타난 atomic 키워드는 스레드의 lock/unlock으로 표현하는 것이 올바르나[3] OSEK OS는 단일 스레드 상에서 동작하는 운영체제이므로 따로 처리하지 않는다.

$$D_1 \frac{\text{\#define S S'}}{\text{\#define S S'}} \qquad D_2 \frac{\text{mtype = \{mtype_list\}}}{\text{enum ID = \{enum_list\}}}$$

$$D_3 \frac{\text{inline ID(param1, param2)}}{\text{void ID(param1, param2)}} \qquad D_4 \frac{\text{typedef ID\{S\}}}{\text{typedef struct [id]\{S\}ID}} \qquad D_5 \frac{\text{d_step\{S\}}}{\text{S}}$$

$$D_6 \frac{\begin{array}{c}\text{if :: cond1} \rightarrow \text{S1} \\ \text{:: else} \rightarrow \text{S2} \\ \text{fi}\end{array}}{\begin{array}{c}\text{if(cond1)\{S1\}} \\ \text{else\{S2\}}\end{array}} \qquad D_7 \frac{\begin{array}{c}\text{do :: cond} \rightarrow \text{S} \\ \text{:: else} \rightarrow \text{break;} \\ \text{od}\end{array}}{\text{while(! cond)\{S\}}} \qquad D_8 \frac{\text{atomic\{S\}}}{\text{S}}$$

$$D_9 \frac{\text{skip}}{\text{[;]}} \qquad D_{10} \frac{\text{(a == b)}}{\text{if(a == b)}}$$

$$D_{11} \frac{\text{unsigned ID}}{\text{unsigned char ID}}$$

그림 2 문법 변환 규칙

4.2. 의미적 변환 규칙

PROMELA의 몇몇 개념들은 C 언어에는 존재하지 않으므로 문법의 변환 규칙만으로 정의할 수 없다. 이러한 요소들을 PROMELA-C언어 모델 간 의미적 변환 규칙으로 정의하였다

$$D_{12}\ \frac{\text{CHANID! (taskid, apiname, param0, param1)}}{\text{FUNCTIONID(taskid, param0, param1)}}$$

$$D_{13}\ \frac{\text{CHANID! (0, RT, tid, 0)}}{\text{[return;]}}$$

$$D_{14}\ \frac{\begin{array}{c}\text{CHAINID? [_, eval(api_name), _, _]}\\ \text{CHANID? (tid, _, p0, p1)}\end{array}}{\textit{call \& param copy}}$$

그림 3 서비스 호출 표현의 변환

PROMELA 모델에서 채널을 이용한 메시지 패싱으로 표현된 OS 서비스 호출은 C 모델에서 void 함수로 표현되었다. 구조적 프로그래밍 언어의 함수 호출은 메시지 패싱과 유사하기 때문이다. 이 때 현재 서비스를 호출한 태스크의 ID를 알기 위해서 taskid 라는 매개변수를 추가로 전달한다. 그러므로 taskid를 갖는 void함수는 서비스 호출이라고 정의할 수 있다.

$$D_{15}\ \frac{\text{proctype PROCID(param)\{S\}}}{\text{void PROCID(taskid, param)\{S\}}} \qquad D_{16}\ \frac{\text{run PROCID(param)}}{\text{PROCID(taskid, param);}}$$

그림 4 의미적 변환 규칙-태스크

PROMELA에서 프로세스로 표현되는 태스크는 C 언어 모델에서는 D15에서 void 함수로 함수로 표현되며, D16과 같이 태스크의 실행을 시작하기 위한 run 키워드는 해당 태스크를 표현하는 함수의 호출로 변환되었다. 태스크를 표현하는 함수는 void 타입이며 매개변수가 없다.

$$D_{18}\ \frac{\begin{array}{c}\textit{timeout}\\ \textit{if(ALL_SUSPENDED)}\\ \textit{exit}\end{array}}{}$$

그림 5 의미적 변환 규칙 - 타임아웃

timeout은 PROMELA에만 존재하는 사전 정의 변수이며 전역적, read-only 속성을 가진 Boolean 변수이다. 초기 값은 거짓이며 이 변수가 참이 될 경우 실행이 종료된다. 타임아웃 변수는 더 이상 참이 될 수 없는 조건을 기다리는 과정을 강제적으로 중단할 수 있도록 하는 변이다. PROMELA 모델에서는 모든 태스크가 종료되었으며 대기 큐가 비어있는 경우 타임아웃이 참이 되어 종료가 일어난다. C언어 모델에서는 이 경우 exit 함수를 사용해 시스템을 종료한다.

4.3. 문맥 변환

두 개 이상의 태스크가 실행되는 경우 태스크의 문맥변환이 일어난다. 하드웨어 의존적인 기능인 문맥변환을 C로 모델링하기 위해 전역 jump 연산을 지원하는 setjmp.h 라이브러리의 setjmp/longjmp 함수를 사용하였다. setjmp 함수로 현재 상태를 저장하고 longjmp를 이용해 저장된 상태로 이동하여 실행을 재개한다.

5. 모델의 검증

제시된 규칙에 따라 C 언어로 작성된 모델은 정확성을 확보하기 위한 검증 절차를 필요로 한다. 검증 결과 아래의 속성을 모두 만족하는 경우 모델이 정확하다고 할 수 있다.

- 모델의 OS 서비스의 행위, 스케줄링, 혹 루틴의 수행이 표준을 준수하는 결과를 보인다.
- 코드의 안전성이 확보된다.

속성들은 표 1과 같이 불변, 가변, 특정 속성으로 분류하였다. 각 속성들은 그 내용과 분류에 따라 검증에 사용될 assert 함수의 표현식과 삽입 위치가 결정된다.

분류	정의
불변 속성	검증 대상 시스템의 시작부터 종료까지 항상 만족해야 하는 속성. OSEK/VDX 명세로 인한 시스템 불변 속성과 자료구조/자료형으로 인한 자료구조 불변 속성으로 분류할 수 있다.
가변 속성	시스템의 시작과 종료 사이에 지속적으로 변경되는 속성. 태스크의 상태, 동적 속성(activate count, priority), 리소스 할당 여부, 이벤트 지정 여부가 이에 해당한다. 모델 내부가 아닌 독립된 테스트 모듈에 assert를 삽입해 검증해야 한다.
특정 속성	시스템의 특정 시점에서 만족해야 하는 속성을 의미한다.

표 1 검증 속성의 분류

검증 수행에 앞서 OSEK/VDX 2.2.3 표준문서에서 검증할 속성을 선정하였다. 그 결과 14개의 불변 속성(시스템 불변 속성9개, 자료구조 불변 속성 5개), 14개의 가변 속성, 7개의 특정 속성, 총 35개의 검증 속성을 식별하였다. 표 2는 그 중 일부를 요약해 정리한 내용이다.

no	분류	설명
1	불변 (시스템)	Running 상태의 태스크는 항상 한 개다.

2	불변 (시스템)	태스크의 우선순위는 0 이상의 값을 가지며 순위가 높을 수록(숫자가 클 수록) 먼저 실행된다.
3	불변 (시스템)	태스크의 activation count는 0 이상이며 설정 파일의 max activation count를 넘을 수 없다.
4	불변 (시스템)	ceiling priority protocol을 만족한다.
5	불변 (DS)	대기 큐의 인덱스 및 C의 언어적 특징으로 인한 코드 안전성 문제가 발생하지 않는다.
6	가변	OS 서비스 호출에 의한 태스크/리소스/이벤트의 상태 변화가 서비스 명세에 일치한다..
7	가변	Resource_Table의 속성이 리소스의 get/release 행위에 의해 표준에 부합하도록 할당/해제된다.
8	가변	Event_Table의 속성이 표준 명세에 맞게 set/wait/clear 된다.
9	특정	모든 태스크는 시스템 종료 전 Terminate 되어야 한다.
10	특정	리소스와 이벤트는 서비스 명세가 요구하는 특정 시점에서의 요구사항을 모두 만족해야 한다.

표 2 OSEK/VDX 표준 문서에서 선정한 검증 속성

표 2에 제시된 속성 중 불변 속성과 특정 속성은 assert 구문의 형식으로 OS 모델 내부에 삽입된다. 이렇게 assert가 삽입된 모델을 제약 조건 패턴을 만족하는 테스트 시퀀스 1개와 만족하지 않는 테스트 시퀀스 3개를 이용하여 검증하였다.

제약조건 패턴을 어긴 테스트 시퀀스로 검증하는 과정에서 C 언어의 특성인 자료형의 크기 제한으로 인하여 오버플로우가 발생하는 것을 확인했다. 따라서 모델링 규칙에 다음과 같은 내용이 추가되었다.

- 태스크/리소스/이벤트 관련 변수 범위 경계에 있는 값인지 검사 후 증감연산을 하도록 한다.

	t9.c	t3.c	t8.c	t10.c
전체	21/21	20/21	19/21	19/21
시스템 불변	9/9	8/9	9/9	9/9
자료구조 불변	5/5	5/5	5/5	5/5
특정	7/7	7/7	5/7	5/7

표 3 cbmc를 이용한 테스트 시퀀스 검증 결과

완성된 모델로 4개의 테스트 시퀀스를 검증한 결과는 표 3과 같다. t9는 제약조건 패턴을 만족하며 나머지는 만족하지 않는다. CBMC의 --all-properties 속성을 사용해 모든 assertion에 대한 만족여부를 확인한 결과 t3에서는 시스템 불변 속성 1개, t8과 t10에서는 특정 속성 2개씩이 발견됨을 확인할 수 있었다.

분석 결과 t3의 assertion은 리소스 중복 할당 시도, t8과 t10에서의 assertion은 basic task 의 event 대기 시도 및 태스크 종료 서비스 호출의 부재로 인해 발생하는 것으로 모델이 아닌 호출 시퀀스의 제약조건 위배로 인한 것으로 확인되었다. 따라서 현재 C언어 모델은 위의 네 테스트 시퀀스에 대해서 정확성을 확보했다고 할 수 있다.

6. 결론 및 향후 계획

검증 결과 C 언어로 기존 모델링 언어와 비슷한 수준의 추상도와 표현력을 갖는 모델을 구현할 수 있고 주어진 테스트 시퀀스에 대해 정확성을 보장함을 확인하였다. 또한 그러나 실험 과정에서 다음과 같은 한계를 발견하였다.

- 제약조건을 asssert 구문으로 삽입하고 이로 인해 발생한 알람을 분석하는 단계가 수동으로 이루어지고 있어 많은 시간이 소요된다.
- 문맥 변환 모델링을 위해 사용한 if(!setjmp(buf))로 인한 복잡도의 증가 및 false alarm 발생 가능성이 있다.

첫 번째와 두 번째 한계점의 개선을 위해서는 제약조건 및 alarm 분석의 자동화가 이루어져야 한다. 세 번째 한계점의 경우 문맥 변환 규칙의 수정을 통해 해결할 수 있을 것으로 예상된다.

또한 현재 검증은 모델 내에 assert 구문을 삽입하는 불변/특정 속성만을 대상으로 하여 일부 테스트 시퀀스로 검증하였으나 향후 연구에서는 가변 속성 역시 반영된 테스트 코드를 다양한 설정 파일 및 이로부터 도출될 수 있는 서비스 호출 시퀀스, 응용 소프트웨어 구현 코드와 함께 검증하여 더 정밀하게 정확성을 검증하고자 한다.

현재 구현된 모델은 논리적 스케줄링 단계 까지 반영하고 있으나 OSEK OS를 더욱 정확하게 반영하기 위해서는 미구현된 훅 루틴 및 인터럽트 계층 역시 향후 연구에서 확장하고자 한다.

7. 참고 문헌

[1] OSEK/VDX Operating System version 2.2.3, http://www.osek-vdx.org/

[2] Edmund Clarke, Daniel Kroening, Flavio Lerda, "A Tool for Checking ANSI-C Programs", 2004

[3] Ashanka Sharma, “A Refinement Calculus for Promela”, 2013