OSEK/VDX OS 를 위한 시스템 생성기 설계

임성락*, 송기석**, 유영창***
*호서대학교, **충북대학교, ***(주)에프에이리눅스
e-mail : srrim@hoseo.edu

A Design of System Generator for OSEK/VDX OS

Rim Seong Rak*, Song Ki Seok**, Yu Young Chang***
*Dept. of Computer Engineering, Hoseo University
**Dept. of Computer Science, chungbuk University
*** Falinux Co.,Ltd.

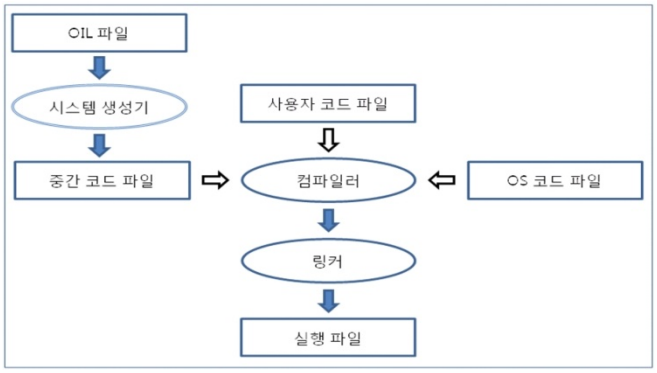
OSEK/VDX OS 는 자동차 전자 제어 장치(ECU)를 위하여 OSEK/VDX 에서 제안한 사양을 준수하는 실시간 운영체제이다. 시스템 생성기는 OIL 언어로 작성된 파일을 C 언어의 중간 코드 파일로 변환하기 위한 도구로써 OSEK/VDX OS 개발에 필요한 요소이다. 본 연구에서는 OSEK/VDX OS 개발에 필요한 시스템 생성기를 보다 쉽게 구현할 수 있는 기법을 제시하였다.

1. 서론

최근 자동차는 성능향상에 필요한 다양한 전자 제어 장치(ECU: Electronic Control Unit)와 이를 제어하기 위한 소프트웨어가 필요하다[1]. 이에 따라 안정적이고 효율적으로 전자 제어 장치를 제어하고 소프트웨어 개발의 효율을 향상시키기 위한 자동차용 소프트웨어의 표준화 작업 필요성이 증가하고 있다. 실제로 자동차 업체들은 표준화 단체 등을 통하여 이러한 작업을 진행하고 있다.

OSEK/VDX OS 는 자동차 전기 제어 장치를 위하여 OSEK/VDX 에서 제안한 사양을 준수하는 실시간 운영체제이다[2]. OSEK/VDX[3]는 OSEK/VDX OS, OSEK/VDX COM, OSEK/VDX NM 에 관한 표준을 정의한다.

OSEK/VDX 응용 프로그램이 개발되는 과정은 (그림 1)과 같다.

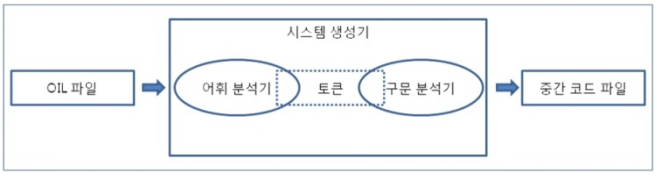


(그림 1) OSEK/VDX 응용 프로그램 개발 과정

OIL(OSEK Implementation Language) 은 OSEK/VDX OS 및 응용 프로그램의 속성을 설정하기 위한 OSEK 구현 언어이다[3][4]. 시스템 생성기(System Generator)는 OIL 언어로 작성된 파일을 C 언어의 중간 코드 파일로 변환한다. 중간 코드 파일, 사용자 코드 파일 그리고 OS 코드 파일을 각각 컴파일링하고 링킹하여 최종적으로 하나의 실행 파일을 생성한다.

2. 설계

본 연구에서는 단순성, 효율성, 이식성을 높이기 위하여 시스템 생성기를 (그림 2)와 같이 어휘 분석기와 구문 분석기로 구분하여 설계한다.



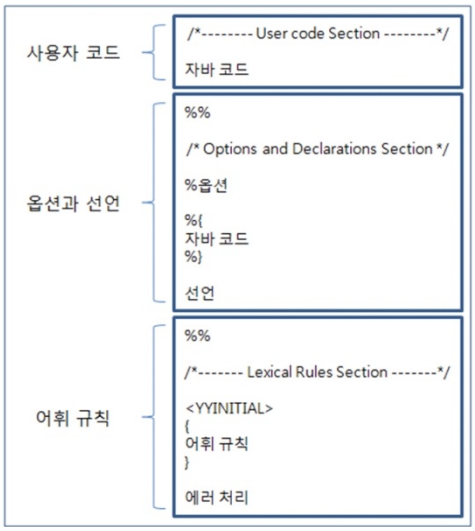
(그림 2) 시스템 생성기 구조

2.1. 어휘 분석기

어휘 분석기는 입력된 코드를 검사하여 키워드, 숫자, 문자로 각각 토큰 분류하고 이를 처리한다. 본 연구에서는 JFlex(ver 1.4.3)[5] 라이브러리를 사용하여 어휘 분석기를 생성하도록 한다.

JFlex 파일의 구조는 (그림 3)과 같이 %%를 경계로 세 부분(사용자 코드 부분, 옵션과 선언 부분, 어휘 규칙 부분)으로 구성된다. 사용자 코드 부분은 어휘 분석기에서 필요한 초기화 코드가 자바 코드로 작성된다. 옵션과 선언 부분은 어휘분석기의 이름이

나 유니코드 지원 같은 옵션 처리와 OIL 을 검사하여 토큰을 분리 처리 그리고 %{와 }% 안에 작성되는 토큰 처리에 관한 자바 코드로 구성된다. 어휘 규칙 부분은 옵션과 선언 부분에서 분리한 토큰을 구문 분석기로 보내고 OIL 해석 과정 중 정의 되지 않은 토큰은 에러 처리 한다.

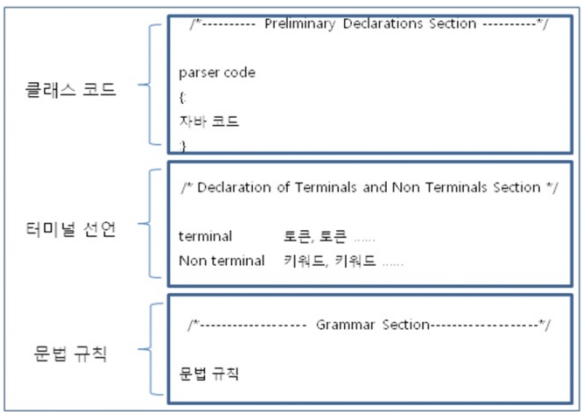


(그림 3) JFlex 파일의 구조

2.2. 구문 분석기

구문 분석기는 어휘 분석기에서 보낸 토큰들을 중간 코드로 변환한다 본 연구에서 CUP(ver 0.11a)[6][7] 라이브러리를 사용하여 구문 분석기를 생성하도록 한다.

CUP 파일의 구조는 (그림 4)와 같이 세 부분(클래스 코드 부분, 터미널 선언 부분, 문법 규칙 부분)으로 구성된다. 클래스 코드 부분은 구문 분석기 클래스를 자바 코드로 작성한다. 터미널 선언 부분은 어휘 분석기에서 정의한 토큰들과 구문 분석기에서 사용하는 키워드들을 터미널로 선언한다. 문법 규칙 부분은 선언된 터미널들로 이루어져 있으며 실제로 터미널에 따라 중간 코드를 변환 되는 해석작업이 이루어진다



(그림 4) CUP 파일의 구조

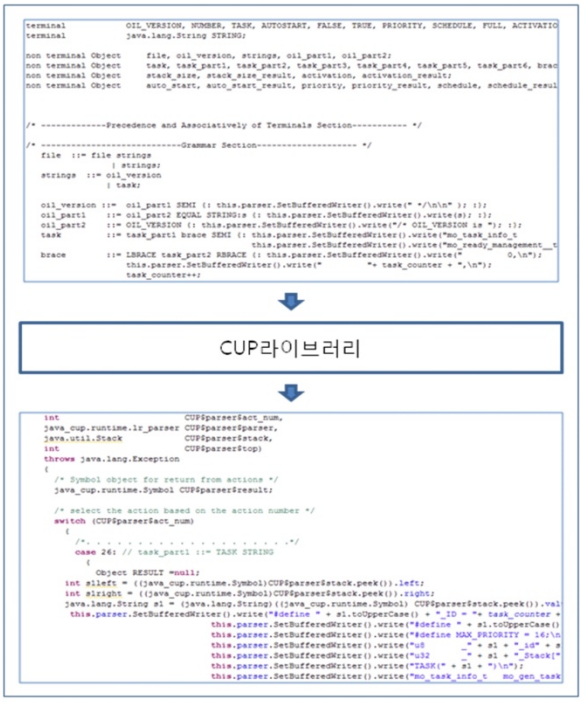
3. 구현 및 실험

3.1. 구현

(그림 5)와 같은 JFlex 파일을 작성하고 JFlex 라이브러리를 이용하여 자바 파일로 변환된 어휘 분석기를 생성한다.

(그림 5) 어휘 분석기 생성

(그림 6)와 같은 CUP 파일을 작성하고 CUP 라이브러리를 이용하여 자바 파일로 변환된 구문 분석기를 생성한다.



(그림 6) 구문 분석기 생성

3.2. 실험

설계한 어휘 분석기와 구문 분석기의 타당성을 검토하기 위하여 다음과 같은 단계로 실험한다.

[단계 1] OIL 파일을 작성한다.

```
    USEPARAMETERACCESS = FALSE;
    USERESSCHEDULER = FALSE;
};

TASK HighTask
{
    AUTOSTART = FALSE;
    PRIORITY = 3;
    SCHEDULE = FULL;
    ACTIVATION = 1;
    STACKSIZE = 512;
};

TASK LowTask
{
    AUTOSTART = FALSE;
    PRIORITY = 3;
    SCHEDULE = NON;
```

(그림 7) OIL 파일

[단계 2] 어휘 분석기에 의해 분리된 토큰을 확인한다.

```
USEPARAMETERACCESS      ;           ;
=                       PRIORITY    }
FALSE                   =           ;
;                       3           TASK
USERESSCHEDULER         ;           LowTask
=                       SCHEDULE    {
FALSE                   =           AUTOSTART
;                       FULL        =
}                       ;           FALSE
;                       ACTIVATION  ;
TASK                    =           PRIORITY
HighTask                1           =
{                       ;           3
AUTOSTART               STACKSIZE   ;
=                       =           SCHEDULE
FALSE                   512         =
                                    NON
```

(그림 8) 분리된 토큰

[단계 3] 구문 분석기에 의해 생성된 중간 코드 파일을 확인한다.

```
    int USEPARAMETERACCESS = NULL;
    int USERESSCHEDULER = NULL;

mo_task_info_t  mo_gen_task[MAX_TASK]   =
{
        {
            (u32)HIGHTASK_ID,
            ((u32)_HighTask_stack)+HIGHTASK_STACK_SIZE-4,
            (u32)HIGHTASK_ID,
            ((u32)_HighTask_stack)+HIGHTASK_STACK_SIZE-4,
            AUTOSTART(0),
            MO_PRIORITY(3),
            MO_PRIORITY(3),
            SUSPENDED,
            BASIC_TASK,
            PREEMPT_TASK,
            MAX_ACTIVATION(1),
            ACTIVATION(0),
            HIGHTASK_ID,
            WAIT_EVENT(NULL),
            SET_EVENT(NULL),
            OWNER_EVENT(NULL),
            NONE_RESOURCE,
        },
        {
            (u32)LOWTASK_ID,
            ((u32)_LowTask_stack)+LOWTASK_STACK_SIZE-4,
            (u32)LOWTASK_ID,
            ((u32)_LowTask_stack)+LOWTASK_STACK_SIZE-4,
            AUTOSTART(0),
            MO_PRIORITY(3),
            MO_PRIORITY(3),
            SUSPENDED,
            EXTEND_TASK,
```

(그림 9) 중간 코드 파일

4. 결론

시스템 생성기는 OIL 언어로 작성된 파일을 C 언어의 중간 코드 파일로 변환하기 위한 도구로써 OSEK/VDX OS 개발에 필요한 요소이다. 본 연구에서는 OSEK/VDX OS 개발에 필요한 시스템 생성기를 보다 쉽게 구현할 수 있는 기법을 제시하였다.

제시한 기법은 시스템 생성기를 어휘 분석기와 구문 분석기로 구분하여 설계하고 JFlex 와 CUP 라이브러리를 이용하여 어휘 분석기와 구문 분석기를 구현함으로써 개발 시간을 단축할 수 있었다. 또한 추가/수정할 OIL 속성이 있는 경우 JFlex 파일과 CUP 파일만을 추가/수정함으로써 어휘 분석기와 구문 분석기를 새로 만들 수 있기 때문에 유지 보수가 매우 편리함을 확인하였다.

참고문헌

[1] 자동차 소프트웨어 융합, KIPA, 2009
[2] OSEK/VDK Operating System Specification Version 2.2.3, 2005. 2. (http://www.osek-vdx.org)
[3] OSEK/VDX Operation System Version 2.2.3, 2005. (http://www.osek-vdx.org)
[4] OSEK Implementation Language Specification Version 2.5, 2005. (http://www.osek-vdx.org)
[5] JFLex (http://jflex.de/)
[6] CUP (http://www2.cs.tum.edu/projects/cup/)
[7] CUP User's Manual.(http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html)