그래픽 기반 시뮬레이션을 활용한 상호배제 교육방법 연구

이영숙† · 남영호††

요 약

병행 프로세스들은 공유 자원을 동시에 읽거나 쓰려고 할 때 서로 경쟁하게 된다. 이때 상호배제, 교착상태, 기아라는 세 가지 제어 문제가 발생한다. 병행성은 이해하기 어려운 운영체제 분야의 주제이다. 현재 대부분의 운영체제 교재에 포함되어 있는 병행 프로그램들은 의사코드로 기술되어 있어, 학습자들은 병행 프로그램들을 실행해 볼 수 없고, 병행 프로그램들의 복잡한 실행 행위를 파악하기 어려워 상호배제 원리를 이해하기 어렵다. 본 연구의 목적은 그래픽 기반 언어인 SDL과 MSC를 이용하여 상호배제 교육방법을 제안하는 것이다. 이를 위해 SDL로 병행 프로그램들을 작성하고 MSC로 시뮬레이션하여 병행 프로그램이 상호배제 요구조건들을 충족시키는지 검증하였다. 설문조사 결과, 이 교육방법이 의사코드 기반 교육방법에 비해 더 효과적인 것으로 나타났다.

주제어 : 병행성, 상호배제, SDL, MSC

A Study on Instruction Method for Mutual Exclusion Using Simulation Based on Graphic

Lee Young-Suk† · Nam Young-Ho††

ABSTRACT

Concurrent processes come into conflict with each other when they are competing for the use of the same resources. In the case of competing processes three control problems must be faced: mutual exclusion, deadlock, and starvation. The concurrency is a subject rather difficult to understand. In addition, because concurrent programs included in most of OS texts are described by pseudocode and are not being able to execute directly, almost learners are difficult to understand behaviour of concurrent programs. The purpose of this study is to propose instruction method for mutual exclusion using SDL and MSC base on graphic. In order to verify the effectiveness of the proposed materials, we compare with materials based on pseudocode. The results indicated that the proposed materials is more effective than materials based on pseudocode in teaching-learning for mutual exclusion mechanisms.

Keyword : Concurrency, Mutual exclusion, SDL, MSC

† 정 회 원: 진주산업대학교(경상대학교 교육대학원 컴퓨터교육전공 석사)
†† 정 회 원: 경상대학교 교수(교신저자)
논문접수: 2010년 07월 13일, 심사완료: 2010년 09월 29일

1. 서 론

교육방법은 광의의 의미에서 교육의 과정이며, 협의의 의미에서 교육 내용의 제시 형태 혹은 원리이며, 교육방법의 가장 중요한 역할은 학습자가 학습하는 과정을 지원하는 데에 있다. 즉 학습자가 정보, 기능, 사고방식, 가치 등을 습득할 수 있도록 도와주는 것이며, 학습자가 스스로 학습할 수 있도록 안내하는 역할을 한다. 이러한 역할을 하는 교육방법은 학습 내용을 조직하고 전달하는 방법을 포함한다. 따라서 교육방법은 학습자의 학습 성취를 좌우하는 중요한 조건이 된다[1].

컴퓨터 학습 목표와 관련된 지식과 기능은 인지적 영역에서부터 심체적, 정의적 영역에 이르기까지 다양하다. 더욱이, 인지적·심체적 활동이 상호 연관되어, 타이핑 능력, 정보 통신기기를 다루는 기계적 능력, 프로그래밍 언어나 응용 소프트웨어의 사용을 통한 논리적인 사고 능력이 포함되므로, 학습자의 효과적인 학습 성취를 위하여 각 영역의 학습 내용이 적절히 조직되고 제시되는 방법과 전략이 고안되어야 한다[2].

컴퓨터 학습 영역 중에서 운영체제는 컴퓨터 시스템 자원의 관리에 관한 기본 철학을 다루는 분야로서, 컴퓨터 관련 분야를 전공하기 위해서는 필수적으로 이수해야 하는 중요한 영역이다.

운영체제는 운영체제의 구조, 프로세스 관리, 기억장치 관리, 보조기억장치 관리, 입출력 관리 등으로 구성되어 있다. 그 중에서도 프로세스 관리의 병행성(concurrency) 주제는 다른 주제와 비교해서 학습 난이도가 높은 주제이다. 이 주제는 상호배제(mutual exclusion), 임계영역(critical section), 교착상태(deadlock) 등을 포함하고 있으며, 병행 처리에 익숙하지 못한 학습자들에게는 어려운 학습내용들이다. 특히 상호배제 문제는 공유 자원을 접근하는 병행 프로그램뿐만 아니라 우리 일상생활의 자원 공유에서도 발생할 수 있다. 상호배제 보장을 위한 프리미티브(primitive)를 소프트웨어적으로 해결하기 위한 의사코드(pseudocode) 형식의 교수-학습 방법[4][5]은 텍스트 형식의 설명만을 사용해서는 효과적인 학습을 기대할 수 없다[3]. 김일민[3]은 이러한 문제점을 해결하기 위해 자바 언어 기반 병행프로그램을 이용하여 상호배제 프리미티브를 소프트웨어적으로 구현하여 교수-학습에 적용하였다. 이 교육방법 역시 원시 프로그램 수준에서 상호배제 관련 프로그램들의 다양한 다중프로그래밍 인터리빙이나 병행처리 되는 문맥을 파악할 수 없어 상호배제 원리 이해가 어렵다. 따라서 단일처리기 또는 다중처리기 환경에서 상호배제 구현 프로그램들이 실행되는 패턴을 원시 프로그램 수준에서 시뮬레이션할 수 있는 적절한 프로그래밍 언어와 실행 환경을 이용하는 교육방법이 상호배제 원리 이해에 효과적일 것이다.

본 연구의 목적은 상호배제 원리 학습을 위해 프로세스들의 병행성을 원시 프로그램 상에서 직접 제어할 수 있고 원시 프로그램 수준에서 병행 실행 결과를 확인할 수 있는 SDL(Specification and Description Language)과 MSC(message sequence charts)를 활용한 그래픽 기반 시뮬레이션 교육방법을 제안하는 것이다.

2. 배경 지식

2.1 병행성과 상호배제

현대 운영체제 설계 기술의 핵심은 병행성이다. 병행성은 프로세스 간 통신, 자원에 대한 공유와 경쟁 등 다양한 이슈들을 포함한다. 병행성은 다중처리기와 분산 시스템과 같이 다수의 처리기가 있는 시스템뿐만 아니라 단일처리기 다중프로그래밍 시스템에서도 발생되는 이슈이다[4].

운영체제는 병행 프로세스를 지원하기 위해 필요한 기본 요구조건인 상호배제를 강제하는 능력을 반드시 제공하여야 한다. 즉 여러 프로세스 또는 쓰레드가 코드, 자원, 그리고 데이터를 공유하고자 할 때, 한 번에 하나의 프로세스만이 공유 객체에 접근할 수 있도록 허용해야 한다.

많은 운영체제 교재에서는 본 연구의 주제인 상호배제를 보장하기 위한 해결안들을 소개하고 있으며, 각 교재들은 해결안들이 충족해야할 다양한 요구조건들을 제안하고 있다[4][5]. 본 연구에서는 많은 교재에서 제안하고 있는 공통적인 다음의 네 가지 요구조건들만을 고려하고자 한다.

◆ 조건 1(Enforce mutual exclusion): 상호배제가

강제되어야 한다.

- 조건 2(No interfere): 임계영역이 아닌 곳에서 수행이 멈춘 프로세스는 다른 프로세스의 수행을 간섭해서는 안 된다.
- 조건 3(No deadlock): 임계영역에 접근하고자 하는 프로세스의 수행이 무한히 미루어져서는 안 된다. 즉 교착상태 및 기아상태가 일어나지 않아야 한다.
- 조건 4(No delay): 임계영역이 비어 있을 때, 임계영역에 진입하려고 하는 프로세스는 즉시 임계영역에 들어갈 수 있어야 한다.

상호배제의 모든 요구조건들을 만족시키는 접근방법에는 여러 가지가 있다. 그 중 한 가지 접근방법은 모든 책임을 병행 수행하려는 프로세스들이 담당하는 것이다. 즉 프로그래밍 언어 또는 운영체제의 특별한 지원 없이, 프로세스 간 협력을 통해 직접 상호배제를 보장하는 것이다. 이 방법을 소프트웨어적인 접근 방법이라 한다. 이 방법은 수행 부하가 크고 잘 설계되지 않을 경우 오동작의 가능성도 높지만, 그럼에도 불구하고 학습자들은 이 방법을 자세히 분석할 필요가 있다. 이를 통해 병행 처리의 복잡도와 상호배제 원리를 더욱 잘 이해할 수 있기 때문이다. 가장 많이 알려진 소프트웨어적인 접근 방법에는 Dekker와 Peterson의 해결안이 있다.

상호배제를 다루는 대부분의 교재에서는 관련 개념을 설명하기 위해 C 언어 형태의 의사코드로 표현된 알고리즘을 제시한다. 학습자가 상호배제 요구조건 보장을 위한 병행 처리 원리를 이해하기 위해 C언어 프로그램으로 구현하여 실행시킨다 할지라도, 사용자 수준에서는 프로세스들의 병행 처리 행위를 전혀 인지할 수 없어 상호배제 원리를 이해하기 어렵다. 본 논문에서는 Dekker의 해결안에 대해 병행 프로세스의 실행 시 다양한 실행 패턴을 생성시킬 수 있으며, 원시 프로그램 수준에서 병행성을 확인할 수 있는 그래픽 기반 시뮬레이션 교육방법을 제안하고자 한다.

2.2 SDL 언어

SDL은 전기통신 시스템을 명세하고(specifying) 서술하기(describing) 위해 ITU-T에서 개발한 국제표준 언어이다[6]. SDL은 그래픽 표현(SDL/GR)과 텍스트 표현(SDL/PR)의 두 가지 문법 표현을 제공한다.

SDL 시스템의 기본적인 이론 모델은 병렬로 수행하는 일련의 확장된 유한상태기계(extended finite state machine)로 구성된다. 유한상태기계들은 상호 독립적이고 불연속적인 시그널들을 통하여 통신한다. SDL 시스템은 구조(structure), 통신(communication), 행위(behavior), 데이터(data) 그리고 타입 개념(type concept)으로 구성된다.

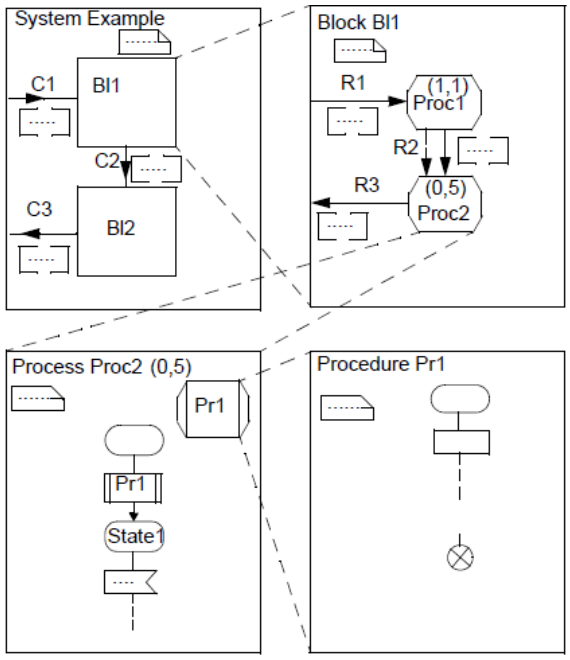

그림 1. SDL 시스템의 계층적 관점

구조는 주요 빌딩 블록들로서 시스템, 블록, 프로세스 그리고 프로시저로 계층적으로 분해된다. 어떤 시스템의 SDL 명세는 상호 연결된 여러 개의 모듈(module)들로 구성되며, SDL에서 이 모듈을 블록(block)이라 한다. 하나의 블록은 순환적으로 블록들의 계층을 형성하는 더 많은 부블록(subblock)들로 분할될 수 있다. 채널은 블록들이 서로 통신하거나 블록들과 환경(environment) 간에 통신하기 위한 통신 경로를 정의한다. 일반적으로 각 채널은 채널을 통해 운송되는 신호들을 저장하는 크기가 무한한 FIFO 큐를 포함한다. 최하위 계층에 있는 블록들의 행위는 하나이상의

통신하는 프로세스들에 의해 서술되어 진다. 프로세스들은 확장된 유한상태기계에 의해 서술되어 진다.

그림 1은 SDL에서 시스템, 블록, 프로세스, 프로시저의 계층적 수준을 나타낸 것이다. SDL 시스템에서 통신은 선택적 시그널 매개변수들을 갖는 비동기적 시그널들에 의해 이루어지며, 동기적 통신을 위해서는 원격 프로시저 호출(RPC)을 사용하여야 한다.

그림 2는 Bl1 블록내에 있는 두 프로세스가 통신하는 방식을 나타낸 것이다.

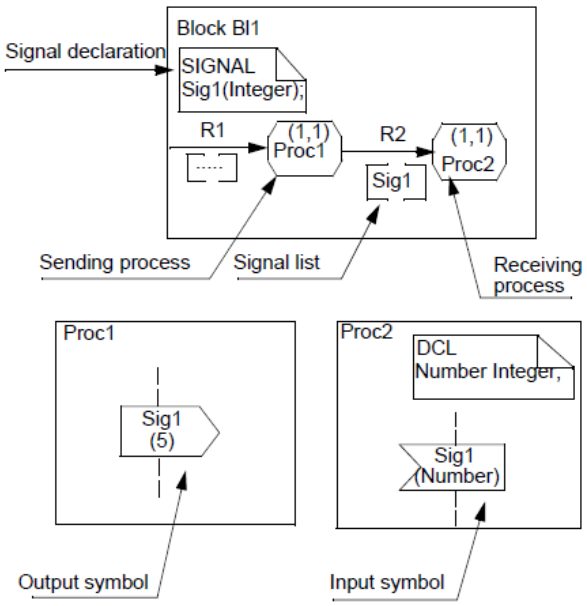

그림 2. 두 프로세스간 신호의 전송

SDL에서는 총괄 데이터를 지원하지 않는다. 이것은 프로세스들 간 또는 프로세스와 외부 간 정보는 반드시 시그널로 전송되어야 한다는 것이다. 시그널들은 비동기적으로 전송된다. 즉, 전송측 프로세스는 수신측 프로세스로부터 응답을 기다리지 않고 실행을 계속한다.

SDL 시스템에서 동적인 행위는 프로세스에 기술되어 진다. 시스템/블럭 계층은 단지 시스템 구조의 정적인 서술이다. SDL에서 프로세스들은 시스템 시작시에 생성 또는 시스템 수행 중에 동적으로 시작될 수 있고 종료될 수 있다. 프로세스에는 한 개 이상의 인스턴스가 존재할 수 있다. 각 인스턴스는 유일한 프로세스 식별자(Pid)를 가진다. 독자적이고 병행적으로 동작하는 프로세스들과 프로세스 인스턴스들은 SDL을 실시간 언어로 만드는 개념이다.

표 1. SDL 명세 기호와 설명

기호	기호 이름	설 명
	블록	시스템 레벨 명세에서 블록 구조를 표현하기 위해 사용된다.
[ ]	채널	시스템 레벨 명세에서 신호의 송수신 방향을 표현하기 위해 사용된다.
	프로세스	블록 레벨 명세에서 프로세서 구조를 표현하기 위해 사용된다.
[ ]	신호 경로	블록 레벨 명세에서 신호의 송수신 방향을 표현하기 위해 사용된다.
	상태	프로세스 명세에서 특정한 상황을 나타내며, 신호를 받음으로써 전이가 일어난다.
	입력	입력 기호 안에 표시된 신호를 자신 또는 다른 프로세스로부터 받는다.
	출력	출력 기호 안에 표시된 신호를 자신 또는 다른 프로세스로 보낸다.
	판단	프로세스 흐름의 분기를 결정한다.
	동작	할당문을 비롯하여 필요한 모든 명령문을 기술한다.
	시작	프로세스의 시작 상태를 나타낸다.
	우선순위 입력	한 상태에 두 개 이상의 입력 기호가 명세되어 있을 때, 우선순위 입력 기호 안에 표시된 신호를 먼저 받는다.
	종료	프로세스의 종료를 나타낸다.
	텍스트	시스템의 구조나 프로세스 행위 명세에서 필요한 신호, 자료구조, 변수 등을 선언하기 위해 사용된다.

표 1은 SDL에서 시스템, 블록, 프로세스 등을 명세하기 위해 사용되는 도식적 표기법의 중요 기호들을 나타낸 것이다. SDL에서 객체의 명세는 순서도(flowchart)를 이용하여 알고리즘을 명세하는 방식과 매우 유사하며, 각 기호는 그래픽 메뉴에서 드래그 드롭 방식으로 쉽게 작성된다.

2.3 MSC 언어

지난 몇년 동안 ITU-T는 형식 언어인 MSC를 표준화하는 노력을 기울여 왔으며, 1992년에 MSC의 첫 번째 버전을 발표하였다[8]. MSC 언어는 SDL 시스템의 동적 행위를 기술함에 있어서 SDL을 보완한다. MSC의 그래픽 표현은 복잡

한 동적 행위를 이해하기 쉽도록 명확하고 모호하지 않는 방식으로 표현하는데 적합하다[7].

MSC는 SDL 시스템을 실행함으로써 생성될 수 있으며, SDL 명세에서 생성된 추상 통신 트리의 한 노드로부터 다른 노드로 하나 이상의 메시지 추적을 기술한다. 기본적으로 하나의 인스턴스로부터 다른 인스턴스로 메시지들을 전송함으로서 정보 전송을 수행한다. 이들 메시지들은 SDL 명세에서 하나의 프로세스로부터 전송되어 다른 프로세스에서 소비되는 시그널들이다. 인스턴스들은 하나의 SDL 시스템, 블록 또는 프로세스와 같은 명세의 어떤 부분에 대응한다.

MSC 언어는 동일한 의미를 갖는 그래픽 표현(MSC-GR)과 텍스트 표현(MSC-PR) 두 가지 표현을 지원한다. MSC는 개발하고자 하는 시스템의 요구사항들을 정의하는 목적의 문서 생성, SDL 설계를 시작하기 전에 다수의 동적 케이스를 식별하고 문서화함으로써 설계 단계의 지원, 이해하기 쉽고 이후에 참조에 대해 검증될 수 있는 그래픽 출력으로서 시뮬레이션의 실행 표현, 그리고 대화형 시뮬레이션 동안 SDL 시스템의 실행 궤적의 표현 및 보고서 생성 등의 영역에서 응용되고 있다.

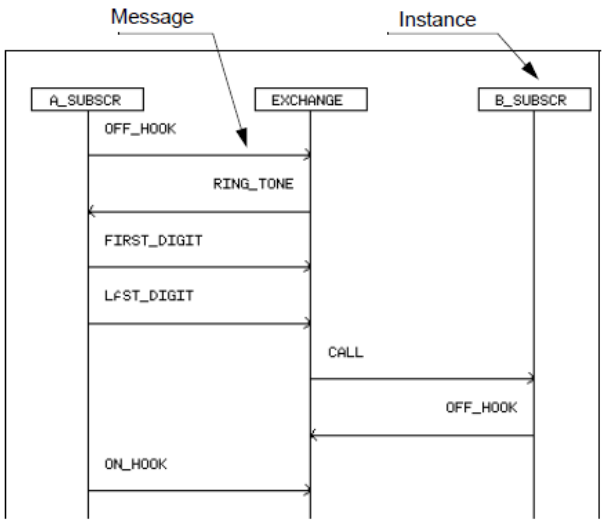

그림 3. MSC 예

그림 3은 MSC의 예를 나타낸 것이다. MSC는 기본적으로 시스템 인스턴스(SDL에서의 블록, 프로세스, 또는 서비스 객체), 환경(외부 세계), 이들 사이의 메시지들로 구성된다. 인스턴스는 클래스에 속하는 하나의 객체(object)이고, 클래스는 SDL에서의 프로세스, 블록 또는 서비스가 될 수 있다. 즉, 인스턴스는 클래스의 특성을 가진 하나의 객체이다. 메시지는 두 개의 인스턴스 사이에서 교환되는 정보이다. 각각의 수직축은 각 객체들 간에 교환되는 논리적 시간 순서를 나타낸다.

2.4 SDL Suite

SDL Suite[9]는 ITU-T SDL 표준을 이용하여 기술된 복잡하고 사건-구동 통신 시스템들을 위한 소프트웨어 명세와 개발 능력을 제공하는 실시간(real-time) 소프트웨어 개발 도구이다. 이 도구는 설계, 테스트, 타켓 코드 생성 등을 위한 다양한 기능을 제공한다.

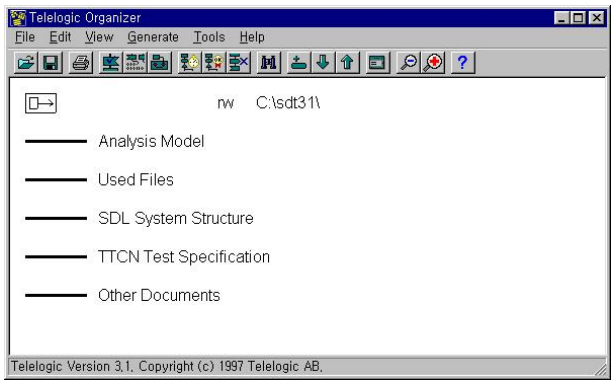

그림 4. SDL Organizer

본 연구에서는 그림 4와 같이 이 시스템에서 가장 기본적인 도구인 SDL 편집기, Text 편집기, SDL 분석기, 코드 생성기, 시뮬레이터만을 이용한다.

그림 5는 SDL로 명세된 시스템을 MSC로 시뮬레이션하기 위한 시뮬레이터의 사용자 인터페이스이다. 시뮬레이터 UI는 SDL 실행 결과의 다양한 정보를 그래픽으로 표현할 수 있도록 여러 가지 선택 사항들을 제공한다.

SDL Suite를 이용하여 시뮬레이션하는 절차는 다음과 같다. 먼저, 그림 4의 SDL Organizer에서 제공하는 SDL 시스템 편집(Edit) 기능을 이용하여 SDL 시스템을 명세한 후, 명세한 시스템의 문법과 의미 오류를 검사한다. 다음으로 MSC 시뮬

레이션을 위해 생성(Generate) 기능을 이용하여 시뮬레이션 파일을 생성한다. 파일 생성이 성공하면 자동으로 그림 5의 시뮬레이터 UI가 자동으로 실행되며, UI에서 제공하는 선택사항들을 적절히 설정한 후 학습자가 환경(Environment) 메뉴를 이용하여 시뮬레이션을 수행한다.

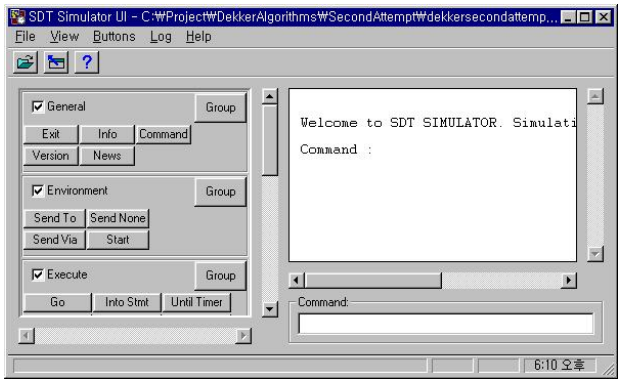

그림 5. 시뮬레이터 UI

본 논문에서는 지면을 고려하여 가장 간단한 형태의 MSC를 생성하여 제시한다.

3. 상호배제 교육방법 개발

Dijkstra는 네덜란드의 수학자 Dekker에 의해 설계된 두 프로세스를 위한 상호배제 해결안을 단계별로 보고하였다. 이러한 접근방법은 병행 프로그램을 개발할 때 만나게 되는 많은 일반 오류들을 접할 수 있는 이점이 있다. 그러나 학습자들은 자원을 공유하는 두 프로세스들이 만들어 낼 수 있는 다양한 패턴의 병행 실행을 의사코드 형태의 원시 프로그램 상에서 상상하기 어렵다. 본 장에서는 이러한 문제를 해결하여 학습자가 원시 프로그램 수준에서 병행 프로세스의 다양한 실행 행위를 시뮬레이션할 수 있는 교육방법을 제시한다.

3.1 첫 번째 해결안

Dekker의 첫 번째 해결안은 상호배제에 대한 어떠한 시도든지 하드웨어 내에서의 기본적인 배제 기법을 이용한다. 이들 중 가장 일반적인 것은 한 메모리 위치에 대해 한 번에 하나의 접근만 허용된다는 제한이다.

프로세스 P0	프로세스 P1

/* PROCESS 0 */
.
.
while (turn != 0)
/* do nothing */ ;
/* 임계영역 */ ;
turn =1;
.
.
	/* PROCESS 1 */
	.
	.
	while (turn !=1)
	/* do nothing */ ;
	/* 임계영역 */ ;
	turn = 0;
	.
	.

그림 6. 첫 번째 해결안의 의사코드

이러한 제약을 이용하기 위하여, 그림 6에서와 같이 turn이라 불리는 전역 메모리 변수를 사용한다. 임계영역을 수행하기를 원하는 하나의 프로세스(P0 또는 P1)는 우선 turn의 내용을 관찰한다. turn의 값이 자신의 프로세스 번호와 같을 경우 프로세스는 임계영역을 수행할 수 있다. 그렇지 않으면, 그 프로세스는 대기해야만 한다. 대기하는 프로세스는 임계영역에 들어갈 수 있을 때까지 반복적으로 turn의 값을 읽는다. 프로세스가 자신의 임계영역에 들어가는 권한을 얻고 임계영역 작업을 마치게 되면, 해당 프로세스는 turn의 값을 다른 프로세스의 번호 값으로 변경해야만 한다.

이 해결안은 상호배제 강제 보장 조건은 충족시키나, 2 가지의 단점이 있다. 첫째는, 프로세스들이 임계영역을 반드시 번갈아 사용해야 하는 점이다. 따라서 전체 수행 속도가 두 프로세스 중 느린 프로세스에 맞추어진다. 예를 들어 P0는 1시간에 1 번만 임계영역을 사용하고 P1은 1시간에 1,000 번의 비율로 임계영역을 사용하기를 원하는 경우, P1은 P0의 속도를 따를 수밖에 없다. 좀 더 심각한 문제는 하나의 프로세스가 실패하는 경우, 다른 프로세스가 영구히 기다려야 하는 점이다.

이것은 한 프로세스가 임계영역 안에서 실패하든지 또는 밖에서 실패하든지와 무관하게 사실이다.

표 2는 첫 번째 해결안에 대한 상호배제 요구조건의 충족 여부를 분석한 것이다.

표 2. 상호배제 조건 충족 여부

조 건	충족여부
조건 1 : Enforce mutual exclusion	○
조건 2 : No interfere	X
조건 3 : No deadlock	○
조건 4 : No delay	X

* ○ : 충족 X : 불충족

3.2 첫 번째 해결안의 SDL 명세

3.2.1 시스템 및 블록 다이어그램

SDL 기반 시스템 명세에서 시스템과 블록 레벨은 개발하고자 하는 시스템의 구조를 나타내고 프로세스 레벨은 시스템의 행위를 나타낸다.

그림 7은 첫 번째 해결안을 SDL로 명세한 시스템 개요를 나타낸 것이다. 시스템은 하나의 블록과 두 개의 프로세스로 구성되어 있으며, 블록은 두 개의 채널(C0, C1)를 통해 사용자 또는 외부 환경과 통신하며, 각 프로세스는 통신경로(R0, R1, R2, R3, BetP0nP1)를 통해 사용자 그리고 다른 프로세스와 통신한다.

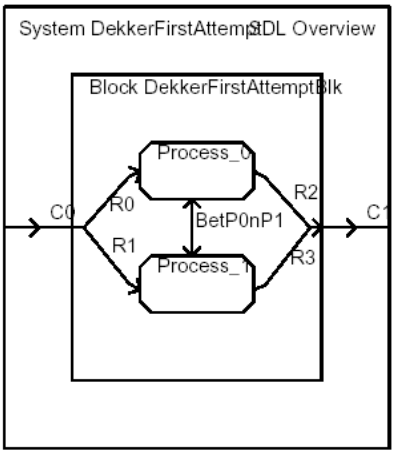

그림 7. 시스템 개요

그림 8은 시스템 구조를 파일 형태의 계층 구조로 나타낸 것이며, 그림 9는 시스템 다이어그램을 나타낸 것이다. 그림 9에서 알 수 있다시피, 채널들을 통해 사용자로부터 블록으로, 블록으로부터 사용자로 신호들이 전달되고 있다. 이러한 신호들은 정보를 포함할 수 있고 단지 사건의 발생을 나타낼 수 있으며, 사용자가 프로세스의 행위를 관찰할 수 있도록 하기 위해 설계된 것이다.

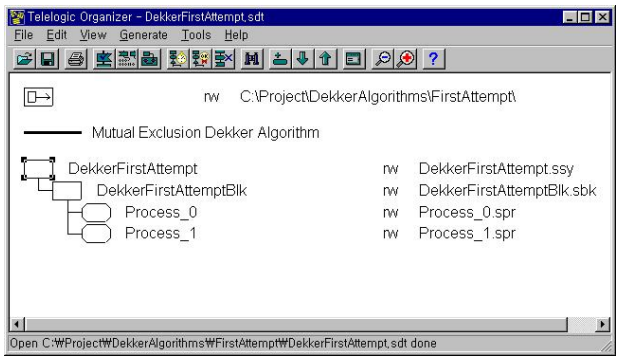

그림 8. 계층 구조로 표현된 시스템 구조

그림 9에서 BeginP0, BeginP1, StopP0, StopP1 등은 단지 사건의 발생을 통지하지만, EnteredCS, ExitedCS 신호들은 사건의 발생뿐만 아니라 사건을 발생시킨 프로세스의 식별자 정보를 전송한다. BeginP0, BeginP1은 프로세스 P0, P1의 실행을 시작시키는 신호이며, StopP0, StopP1은 프로세스 P0, P1의 실행을 종료시키는 신호이다. 또한 EnteredCS(SELF)는 프로세스 자신이 임계영역에 진입하였다는 사실을 사용자에게 통보하기 위한 것이며, ExitedCS(SELF)는 프로세스 자신이 임계영역에서 빠져나왔다는 사실을 사용자에게 알리기 위한 것이다.

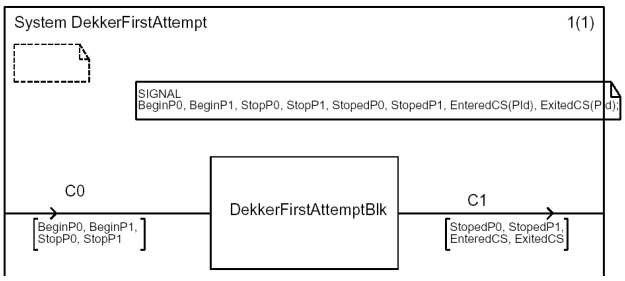

그림 9. 첫 번째 시도 - 시스템 다이어그램

그림 10은 시스템을 구성하는 블록의 구조를 상세히 나타낸 것이다. DekkerFirstAttemptBlk 블록은 P0, P1 두 개의 프로세스를 포함하며, 각 프

로세스들은 그림 11과 그림 12에 기술된 각 프로세스 다이아그램에 대응한다. 그림 10에서 프로세스 P0, P1은 각각 BetP0nP1 신호경로를 통해 chgTurn0, chgTurn1 신호를 이용하여 공유 변수 turn의 변경된 값을 상호 교환한다.

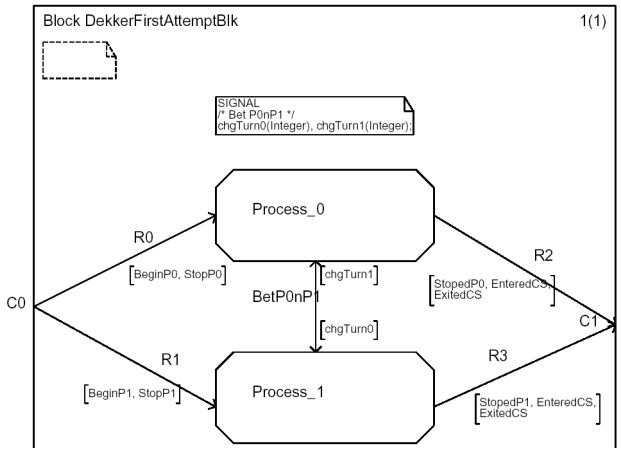

그림 10. 첫 번째 시도 - 블록 다이어그램

3.2.2 프로세스 다이어그램

SDL 명세에서 프로세스는 시스템 명세 단계에서 추상화 수준이 가장 낮은 객체이다. 프로세스는 시스템이 취해야 할 행위를 나타내며, 필요에 따라 서비스 객체 또는 프로시저 객체를 포함할 수 있다. 그림 11은 그림 6의 의사코드에서 프로세스 P0와 동일한 결과를 산출하는 SDL 프로세스 명세이다. 의사코드와 동일하게 공유 변수 turn 값은 초기에 0으로 설정되며, 이것은 프로세스 P0가 먼저 임계영역에 진입할 수 있음을 의미한다.

그림 11의 프로세스 P0의 행위를 설명하면 다음과 같다. P0는 입력 신호(StopP0, BeginP0, chgTurn1)들 중 하나가 발생하기를 Ready 상태에서 기다리고 있다. 만일 BeginP0 신호가 사용자로부터 발생한다면, turn 값이 0인지를 검사한다. 이때 turn 값이 0이라면, P0는 임계영역에 진입하며, 사용자에게 임계영역에 진입한다는 사실을 통보하고(EnteredCS) 바로 임계영역을 빠져나오면서 그 사실을 역시 사용자에게 통보한다(ExitedCS).

그 다음 turn 값을 1로 변경하고 변경된 turn

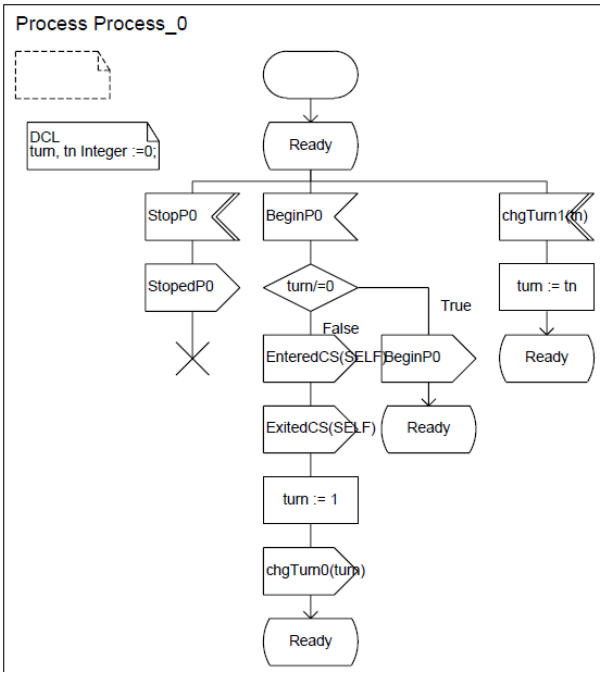

그림 11. 프로세스 P0 다이어그램

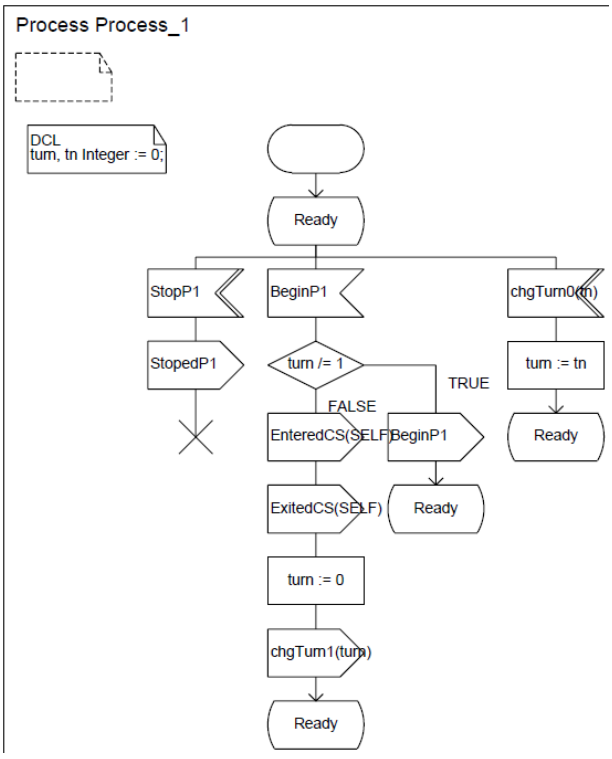

그림 12. 프로세스 P1 다이어그램

값과 그 사실을 그림 12의 프로세스 P1에게 통보(chgTurn0(turn))한 후, 다시 임계영역에 진입할 준비를 한다. 또한 준비 상태에서 프로세스 P1이

공유 변수 turn 값의 변경을 알리는 신호(chgTurn1(turn))를 수신할 수 있다. 이 신호가 도착되면, 자신의 지역변수에 값을 할당하고 준비 상태로 귀환한다.

그림 12의 프로세스 P1도 동일한 논리로 명세되어 있다. 따라서 그림 11과 그림 12는 그림 6의 의사코드로 기술한 해결안과 동일한 행위를 수행한다.

3.3 MSC를 이용한 상호배제 조건 검증

Dekker 첫 번째 해결안은 상호배제 요구조건 4가지 중 조건 1과 조건 3은 충족시키지만 나머지 조건들을 충족시키지 못한다는 것을 표 2에서 분석하여 제시하였다. 본 절에서는 각 조건의 충족 여부를 MSC를 이용한 시뮬레이션을 통해 검증한다.

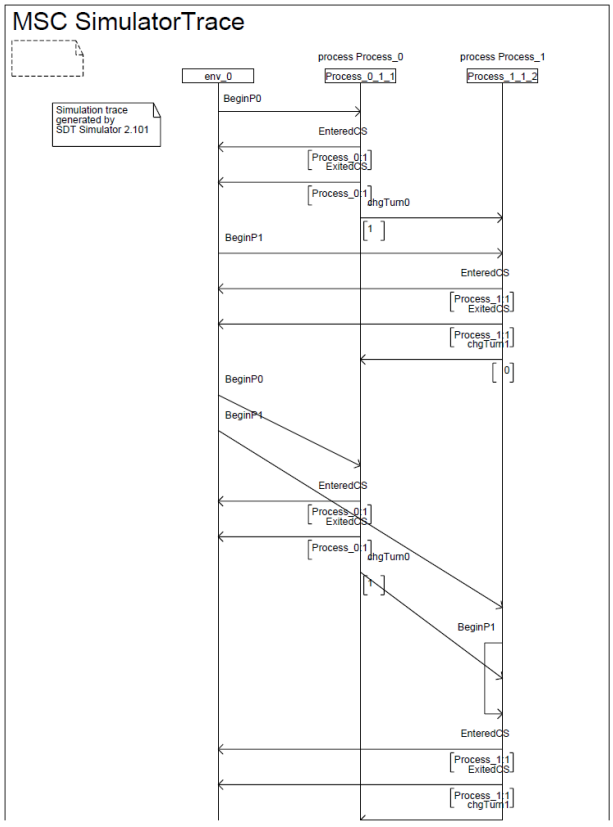

그림 13. 첫 번째 해결안 - 조건 1 검증

3.3.1 요구조건 1의 검증

요구조건 1은 병행 프로세스 간에 상호배제가 보장되어야 한다는 것이다.

그림 13은 SDL로 명세한 첫 번째 해결안이 상호배제 조건 1을 충족하는지를 검증하기 위해 그림 5의 시뮬레이터에서 MSC를 이용한 시뮬레이션 결과를 나타낸 것이다. 그림 13에 의하면 동시에 실행 중인 두 프로세스 중에서 어느 한 순간에 반드시 단 하나의 프로세스만이 임계영역에 진입함을 알 수 있다. 먼저 프로세스 P0가 임계영역에 들어갔다 나온 후에 turn 변수 값을 1로 설정하면 프로세스 P1이 다음으로 임계영역에 진입할 수 있다.

그림 13에서 알 수 있는 것은 첫 번째 해결안은 상호배제 강제를 보장하지만 반드시 두 프로세스가 임계영역을 번갈아 사용해야 한다는 것이다.

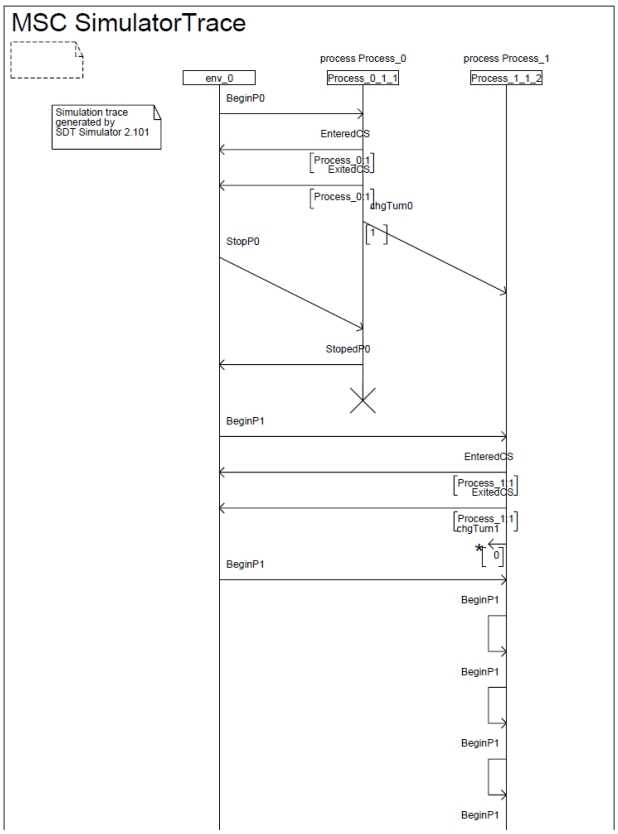

그림 14. 첫 번째 해결안 - 조건 2 검증

3.3.2 요구조건 2의 검증

병행 수행 환경에서 임계영역이 아닌 곳에서 수행이 멈춘 프로세스는 다른 프로세스의 수행을 간섭해서는 안 된다.

그림 11의 P0 프로세스 다이어그램에서 임계영

역을 빠져나온 직후(turn을 1로 변경하기 이전에) 프로세스 수행이 종료되거나 프로세스 P0의 수행이 중단된다면, 그림 12의 프로세스 P1은 turn의 값을 1로 변경할 수 없기 때문에 영원히 임계영역에 진입할 수 없다. 그림 14는 P1의 이 실행 행위를 MSC로 검증한 것이다.

3.3.3 요구조건 3의 검증

병행 수행 환경에서 임계영역에 접근하고자 하는 프로세스의 수행이 무한히 미루어져서는 안된다. 첫 번째 해결안에서 각 프로세스는 임계영역을 반드시 한 번씩 번갈아 들어갔다 나와야 하지만, 각 프로세스가 임계자원을 필요로 한다면 교착상태에 빠질 염려는 없다.

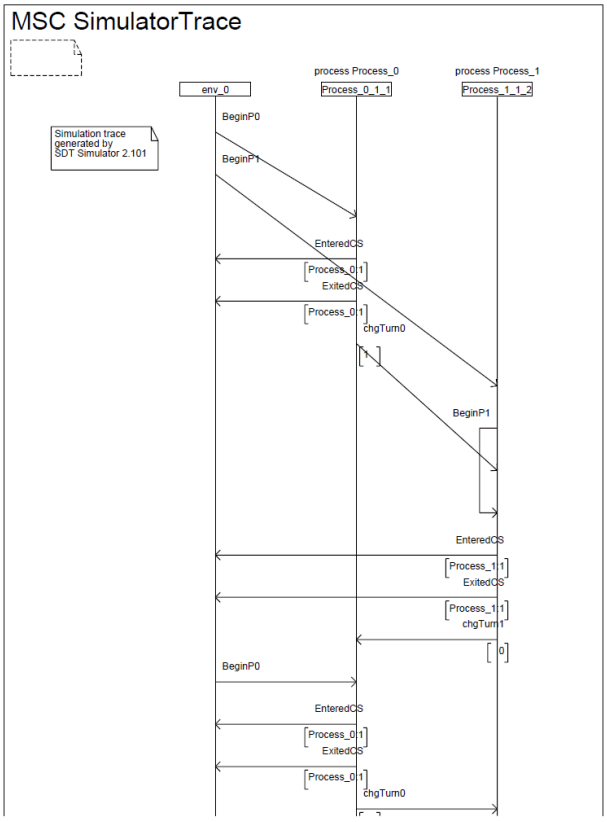

그림 15. 첫 번째 해결안 - 조건 3 검증(계속)

그림 15에서 turn의 값이 0인 경우에, 프로세스 P0가 Ready 상태에서 수행이 중지된 경우 프로세스 P1은 turn의 값이 1이 될 때까지 계속하여 조건 검사를 수행함을 알 수 있다. 그러나 프로세스 P0가 임계자원을 사용한다면, 프로세스 P1은 임계영역에 진입할 수 있으므로 교착상태가 아님을 알 수 있다.

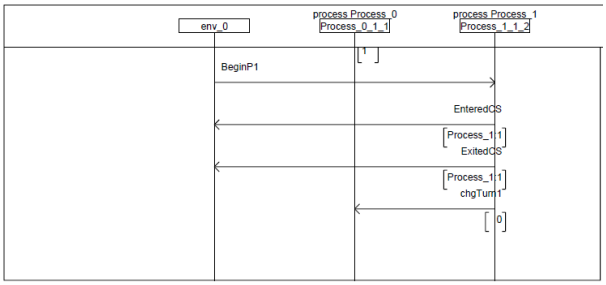

그림 15. 첫 번째 해결안 - 조건 3 검증

3.3.4 요구조건 4의 검증

병행 수행 환경에서 임계영역이 비어 있을 때, 임계영역에 진입하려고 하는 프로세스는 즉시 임계영역에 진입할 수 있어야 한다.

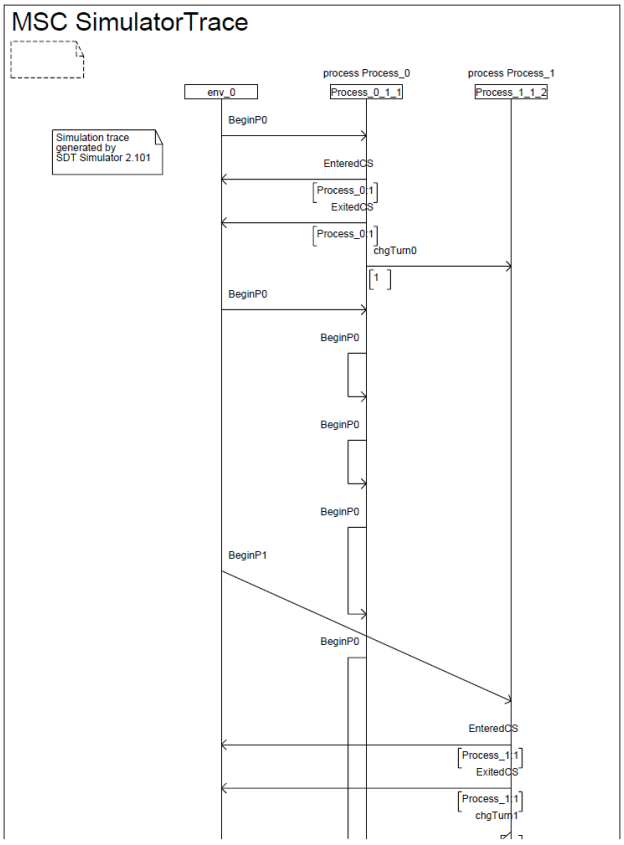

그림 16. 첫 번째 해결안 - 조건 4 검증(계속)

그림 16은 프로세스 P0이 임계영역을 수행한 후, 임계영역이 비어 있는 상태에서 P0이 다시 임

계영역에 진입하려고 시도하지만 임계영역에 진입할 수 없음을 나타낸 것이다. 즉, 상호배제 조건 4를 충족시키지 못함을 증명하고 있다.

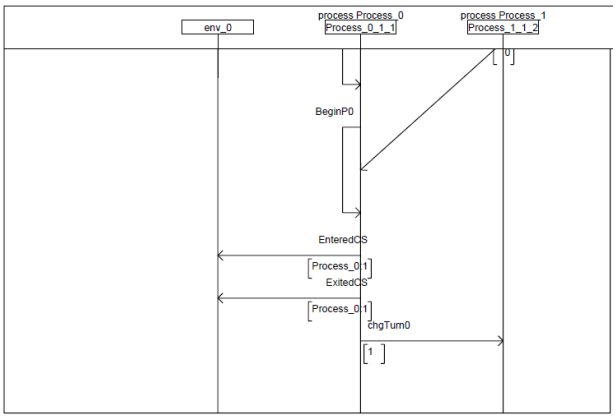

그림 16. 첫 번째 해결안 - 조건 4 검증

4. 평 가

본 연구에서는 의사코드를 통해 상호배제 개념과 원리를 선행 학습한 경험이 있는 대학교 3학년 학생들을 대상으로 본 연구에서 개발한 교수-학습 자료를 이용하여 수업을 진행하였다. 개발한 교수-학습 자료의 효과성 향상을 조사하기 위해, 학생들을 대상으로 수업 집중도, 흥미도, 이해도, 자기주도 학습 가능성 등에 대한 빈도와 백분율을 조사하였다.

4.1 학습자 수업 집중도 조사

표 3은 본 연구에서 개발한 교육방법을 이용하여 상호배제에 관한 개념과 원리 수업이 의사코드를 이용한 수업에 비해 학생 자신이 수업에 더 집중할 수 있었는지에 대한 설문조사 결과이다. 학습자 스스로가 수업에 더 집중할 수 있었다고 응답한 학생이 64.7%로 였으며, 그렇지 않다고 응답한 학생이 11.8%로 비교적 낮았다. 학생 스스로 수업에 더 집중할 수 있었던 이유로는 제시한 교육방법이 시각적 효과가 뛰어나 의사코드를 해석하는 것보다 학습동기를 자극하고, 병행 프로세스 실행 흐름을 직접적으로 확인할 수 있기 때문이라고 응답하였다.

표 3. 수업의 집중도 조사

구 분	빈도(명)	백분율(%)
매우 높아졌다	4	23.5
높아졌다	7	41.2
같다	4	23.5
높아지지 않았다	2	11.8
전혀 높아지지 않았다	0	0
합 계	17	100

4.2 학습자의 지적 호기심 유발 조사

표 4는 의사코드를 이용한 수업보다 학습자의 지적 호기심을 더 유발시키는지에 대한 설문조사 결과이다. 학습자들이 본 교육방법을 이용하여 학습한 후, 스스로 상호배제와 관련된 다양한 문제가 무엇인지에 대해 지적 호기심이 더 유발되었다고 응답한 학습자는 64.7% 였으며, 11.8%는 그렇지 않다고 응답하였다. 지적 호기심이 더 유발되었다고 응답한 이유로는 병행 프로세스를 실제 컴퓨팅 환경에서의 수행과 유사하게 다양한 병행 패턴으로 시뮬레이션하여 그 결과를 직접 확인할 수 있고, 순서도와 유사한 그래픽 기반 프로그램을 수행할 수 있다는 사실 때문이라고 응답하였다.

표 4. 지적 호기심 유발 조사

구 분	빈도(명)	백분율(%)
매우 높아졌다	2	11.8
높아졌다	9	52.9
같다	4	23.5
높아지지 않았다	2	11.8
전혀 높아지지 않았다	0	0
합 계	17	100

4.3 상호배제 원리 이해도 조사

표 5는 의사코드를 이용한 수업에 비하여 상호배제 개념 및 원리를 이해하기 더 쉽게 전달하는가 라는 설문조사 결과이다. 개념 및 원리 이해가 더 쉽게 전달된다고 응답한 학습자는 82.3% 였으며, 그렇지 않다고 응답한 학습자는 5.9%였다. 그 이유로는 순서도와 유사한 병행 프로그램을 이용

하여 다양한 병행 패턴을 직접 실행해 볼 수 있기 때문이라고 응답하였다.

표 5. 상호배제 원리 이해도

구 분	빈도(명)	백분율(%)
매우 높아졌다	4	23.5
높아졌다	10	58.8
같다	2	11.8
높아지지 않았다	1	5.9
전혀 높아지지 않았다	0	0
합 계	17	100

4.4 학습자 주도 학습 가능성 조사

본 교육방법은 의사코드 기반 교육방법에 비해 학생 스스로 학습에 적극 참여할 수 있도록 제작되어 있는가라는 조사에 그렇다고 응답한 학습자는 표 6과 같이 76.5%였으며, 그렇지 않다고 응답한 학습자는 전무하였다. 그렇다고 응답한 이유로는 교수-학습 자료와 실행 소프트웨어가 주어진다면 복습을 위해, 다양한 패턴의 병행 실행 결과를 확인하기 위해 스스로 실행할 수 있다라고 답하였다.

표 6. 학습자 주도 학습의 가능성 조사

구 분	빈도(명)	백분율(%)
매우 높아졌다	6	35.3
높아졌다	7	41.2
같다	4	23.5
높아지지 않았다	0	0
전혀 높아지지 않았다	0	0
합 계	17	100

5. 결 론

운영체제는 컴퓨터 관련 전공자에게는 필수 과목이며, 병행처리는 운영체제 학습에 있어서 가장 어려운 부분이다. 많은 학습자들이 비동기 병행처리 부분을 이해하는 데 어려움을 겪는 이유는 대부분의 교육방법이 의사코드로 제시되어 있기 때문이다. 이러한 의사코드는 학습자가 선 습득한 컴퓨팅 환경을 기준으로 학습자의 두뇌와 연필만으로 실행될 수 있기 때문에, 이러한 교육방법은 실제 컴퓨터상에서 발생할 수 있는 다양한 병행성을 시연할 수 없으며, 아주 제한적인 사건 순서만이 고려된다.

본 논문에서는 IBM Rationale SDL Suite 상에서 특정 공유 자원과 관련된 프로세스 집합의 구조와 각 프로세스의 행위를 나타내는 SDL 기반 프로세스들을 구현하고, MSC를 통해 상호배제 요구조건들을 충족시키는지를 검증할 수 있는 교육방법을 개발하였으며, 설문조사를 통해 다음과 효율성을 입증하였다.

첫째, 제시한 교육방법은 시각적 효과가 뛰어나 의사코드를 해석하는 것보다 학습자들의 학습동기를 자극하고, 병행 프로세스 실행 흐름을 직접적으로 확인할 수 있어 학습자의 수업 집중도를 향상시킬 수 있다.

둘째, 제시한 교육방법은 순서도와 유사한 그래픽 기반 병행 프로세스를 실제 컴퓨팅 환경에서의 수행과 유사하게 다양한 병행 패턴으로 시뮬레이션하여 그 결과를 직접 확인할 수 있어 학습자의 지적 호기심을 유발할 수 있고 상호배제 개념과 원리를 쉽게 이해할 수 있다.

셋째, 학습자는 제시한 교육방법을 이용하여 스스로 학습할 수 있다는 것이다.

따라서 본 논문에서 제시한 SDL과 MSC 기반의 교육방법은 병행성을 기반으로 하는 모든 컴퓨팅 개념과 원리의 교수-학습에 효과적으로 적용될 수 있을 것이다.

참 고 문 헌

[1] 허희옥 외 5명 (2003). **컴퓨터교육방법 탐구**. 서울: 교육과학사.

[2] 이옥화 외 5명 (2000). **컴퓨터교육의 이해**. 영진.com.

[3] 김일민 (2001). 자바스레드를 이용한 운영체제 교육. **컴퓨터교육학회논문지, 4(1)**, 19-26.

[4] William Stallings (2009). *Operating System Internals and Design Principles*. 5th Edition. Pretice Hall.

[5] Abraham Silberschatz, Peter B. Galvin, Greg Gagne (2009). *Operating System Concepts*. 8th Edition. John Wiley & Sons. Inc.
[6] ITU-T (2000). *ITU Recommendation Z.100: The Specification and Description Language(SDL)*. ITU. Geneva.
[7] Anders Olsen, et.al (1995). *Systems Engineering Using SDL-92*. ELSEVIER SCIENCE B. V.
[8] ITU-T (1994). *Message Sequence Charts*. Geneva. ITU-T.
[9] _____. http://www-142.ibm.com/software/products/kr/ko/ratisdlsuit.

이 영 숙

1997 진주산업대학교 전자계산학과(공학사)
2010 경상대학교 교육대학원 컴퓨터교육전공(교육학석사)
1997~현재 진주산업대학교 행정실
관심분야: 컴퓨터교육, 멀티미디어통신
E-Mail: yslee@jinju.ac.kr

남 영 호

1989 경상대학교 전자계산통계학과(이학사)
1991 중앙대학교 전자계산학과(공학석사)
1994 중앙대학교 컴퓨터공학과(공학박사)
1995~현재 경상대학교 컴퓨터교육과 교수
관심분야: 컴퓨터교육, 무선프로토콜 검증
E-Mail: yhnam@gnu.ac.kr