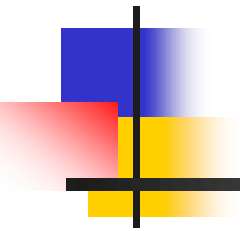
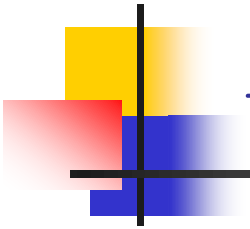


임베디드 소프트웨어
(Introduction to Embedded OS)

인하대학교 컴퓨터공학과

가을, 2018



목차

- 임베디드 시스템과 운영체제
- 운영체제 Review
- µC/OS-II – the first program

실시간 임베디드 시스템 (1)

- 실시간 시스템
  - 정해진 시간 내에 시스템이 결과를 출력하는 시스템
  - 주어진 작업을 빨리 처리하는 것이 아니고 정해진 시간을 넘어서는 안 된다는 뜻임
  - 결과 산출에 걸리는 시간에도 적시성 (timeliness)을 가지며 외부 자극에도 예측 가능한(predictable) 방식으로 반응.

- 적시성(timeliness)이란?
  - 데드라인(deadline: 반응에 요구되는 시간의 한계 값) 이내에 논리적으로도 정확한 출력 값을 산출해 내는 것
    - Ex) DVD player : 영상 한 frame을 1/30초에 하나씩 decoding (복호화) 해서 화면에 뿌려주어야 함 ➔ deadline : 33 milisecond

실시간 임베디드 시스템 (2)

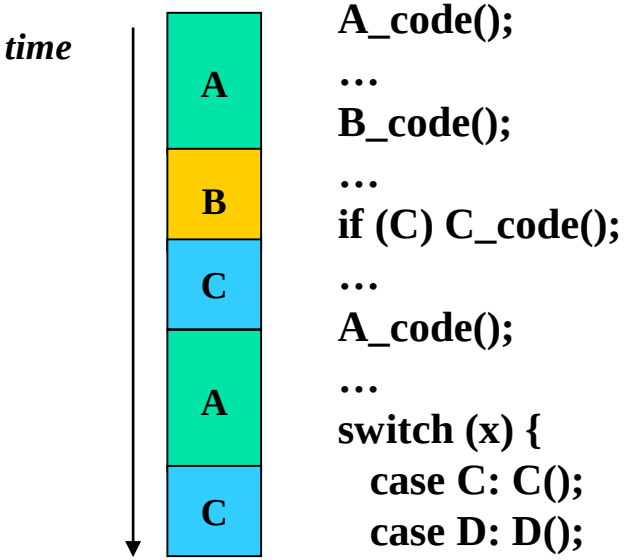
- 실시간 시스템의 분류
  - 경성 실시간 시스템 (hard real-time system)
    - 제어작업이 데드라인을 어기는 경우 시스템에 심각한 영향을 주는 속성을 지닌 시스템
    - 예) 원자력 발전소, 항공기, 우주 왕복선 등의 제어시스템
  - 연성 실시간 시스템 (soft real-time system)
    - 데드라인 만족시키는데 있어서 어느 정도 융통성을 갖는 (즉, 정해진 범위를 넘는 시간 지연이 발생하더라도 시스템 전체의 error가 되지 않는) 시스템
    - 예) DVD 재생기컴퓨터, 정보기기, 네트워크 관련기기 등
- 실시간 시스템 .vs. 임베디드 시스템 ?
  - 임베디드 시스템은 대부분 실시간적인 요소 내포

실시간 임베디드 시스템 (3)

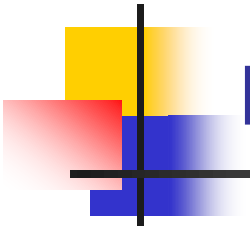
- 경성 실시간 시스템의 일반적인 단점
  - 불완전한 기능의 운영체제
    - 실시간성 보장을 위해 task의 생성/실행/종료의 자유도를 제약함
  - 많은 RTOS에서 thread 모델로서 프로그램 실행
    - 메모리 보호를 받지 못하는 프로그램 수행
    - 한 개의 응용 프로그램(task)의 버그가 전체 시스템을 다운으로 유도
    - ROMable 운영체제
  - 특정회사 혹은 기능에 따라 개발
    - 소스 프로그램의 비공개
    - 초기구입 비용 및 사용료가 고가로 비경제적

초기의 임베디드 시스템 SW

- 초기의 임베디드 시스템(=실시간 시스템)
  - 간단한 Firmware가 주류
  - 간단하고 단순한 순차적인 작업에 관련,순차적인 프로그램으로 충분하였음
  - 8bit, 16bit 마이크로프로세서(마이크로콘트롤러) 사용




```
A_code();
…
B_code();
…
if (C) C_code();
…
A_code();
…
switch (x) {
   case C: C();
   case D: D();
   ...
```



Embedded SW Complexity

- **Size of Typical Embedded System**
  - 1985 13 kLOC
  - 1989 21 kLOC # 44 % per year
  - 1998 1 MLOC
  - 2000 2 MLOC
  - 2008 16 MLOC ≈ Windows NT 4.0
  - 2010 32 MLOC ≈ Windows 2000

OS 탑재 필요성

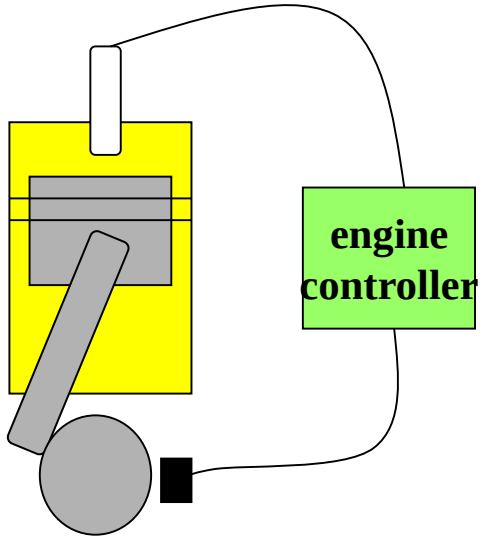
- 시스템 고기능화
  - 시스템의 규모가 커짐에 따른 Multi Tasking 기능 요구 (Network이나 multimedia가 시스템의 기본으로 자리 잡음)
  - 다양한 디바이스 수용과 관리(Networking, GUI, Audio, Video)
  - 기능성이 부가되고 기능이 많아지고 복잡해짐
  - 순차적인 프로그램 작성이 불가능하여 운영체제가 도입됨

- 빠른 개발과 유지보수 용이
  - 새로운 요구 상황에 맞게 프로그램을 수정, 다운로딩이 용이
  - 새롭게 생겨나는 고품질의 창조형 서비스를 적시에 제공

*그러나 임베디드 시스템의 특성상 실시간성을 지원해야 하므로 일반 운영체제로는 한계가 있어, 실시간 운영체제라는 특별한 **OS**가 개발됨*

임베디드 시스템에서의 운영체제

- Multiple-process의 필요성 :
  - 프로세스(혹은 thread)를 이용하여 시간적 복잡성을 해결
    - multiple rates : multimedia, automotive
    - asynchronous input : user interfaces, communication systems
  - Example : Engine control
  - Tasks:
    - spark control
    - crankshaft sensing
    - fuel/air mixture
    - oxygen sensor
    - Kalman filter
    - State machine



임베디드 시스템 OS

- **Real Time OS**
  - VxWorks: HONDA의 Asimo에 사용
  - pSOS: 삼성전자 피처폰, VxWorks 통합
  - VRTX, uC/OSII

ASIMO
VxWorks

우리별 2호
Nucleus plus

- **일반 임베디드 OS**
  - Windows CE (pocket PC): HP iPAQ series
    - Intel x86 계열 지원, MS 회사의 각종 환경 지원(ActiveX, Win32 API등)
  - 3Com Palm
    - Motorola 드래곤볼 CHIP 기반의 운영체제, 수년간 개발로 안정적인 OS와 대양한 응용 프로그램 제공(sony클레오)
  - Symbian
    - 모바일 통신회사들이 결성한 단체에서 모바일 단말기에 사용한 목적의 운영체제: smart phone을 목표로 GUI 환경에 멀티미디어 서비스 제공
  - Embedded Linux: PDA 단말기

임베디드 운영체제 - RTOS

- pSOS
  - 1980년대 인테그레이티드 시스템즈에서 개발된 임베디드 디바이스용 실시간 커널
  - pSOSystem
    - pSOS를 이용한 OS이며 커널을 중심으로 여러 개의 소프트웨어 컴포넌트로 구성됨
    - 멀티태스킹 RTOS로 256개의 태스크를 가지며 우선순위를 가지는 선점형 스케쥴링 방식
  - pRISM+
    - pSOSystem 개발자를 위한 통합 개발환경
  - BSP(Board Support Package)
    - 디바이스 드라이버와 칩 및 각종 디바이스의 정보를 가짐
    - 하드웨어가 변경될 경우 시스템 프로그램에 큰 변경 없이 BSP만 수정하여 쉽게 처리 가능

임베디드 운영체제 - RTOS

- Nuclesus PLUS
  - Accelerated Technology에서 개발된 RTOS
  - 소스 코드의 제공
  - 휴대폰 단말기, PDA, 우리별 1호, 2호에도 탑재됨

- REX (Real-time EXecutive)
  - 퀄컴에서 만든 RTOS
    - CDMA 휴대폰에 탑재

우리별 2호
Nucleus plus

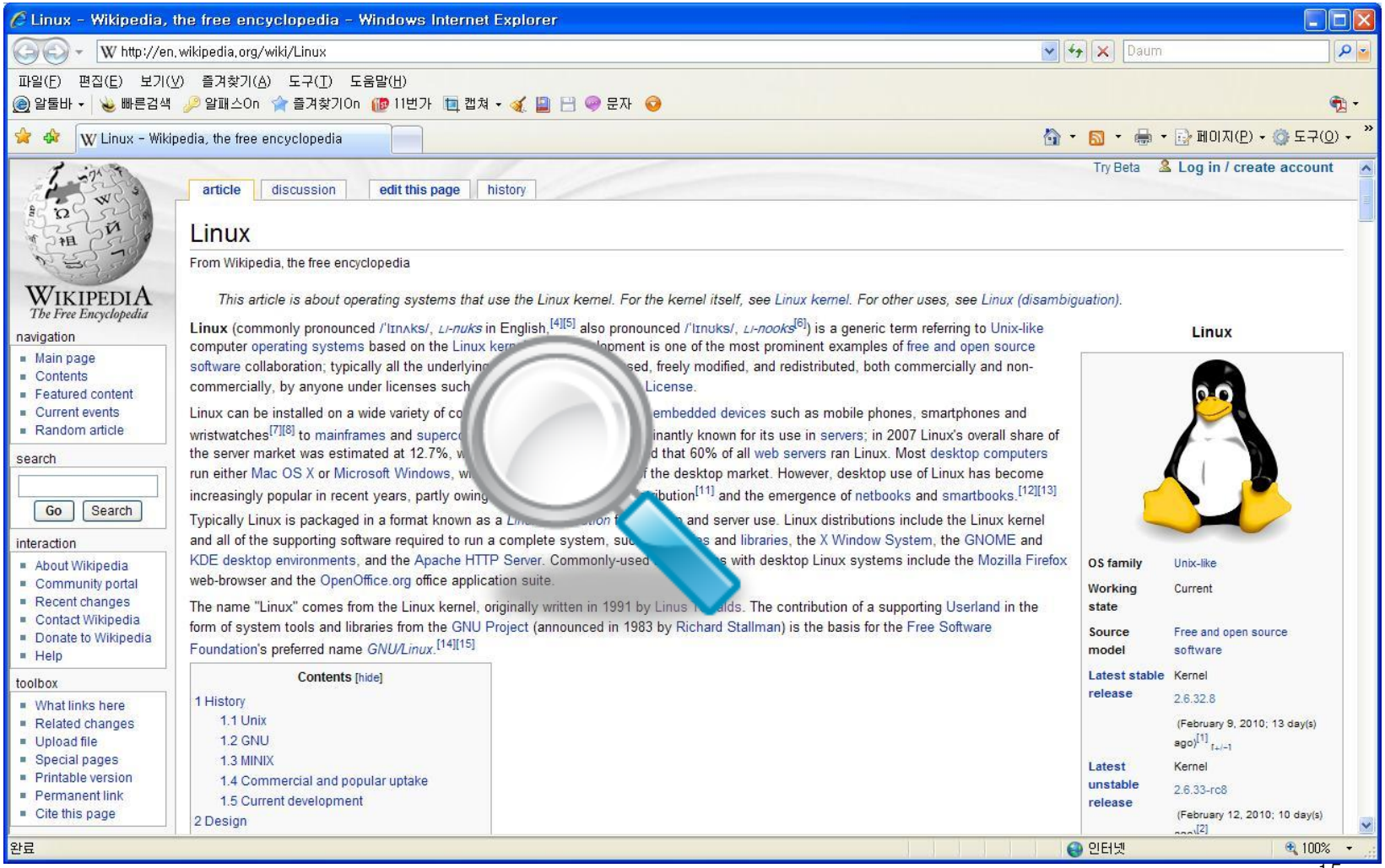
임베디드 운영체제 - WinCE

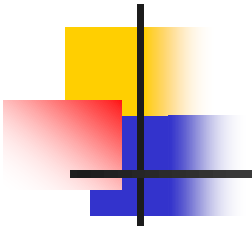
- 윈도우 CE
  - MS사에서 임베디드 시스템을 위하여 제공하는 운영체제
  - 기존의 데스크 탑 PC와 동일한 윈도우 환경 제공
  - 데스크 탑 윈도우 및 응용 프로그램과의 호환성 우수
  - 프로그램 개발 환경이 아주 우수. (특히, GUI 개발 환경 우수)
  - 실행 환경에서 요구되는 H/W 사양이 높고, 가격이 비쌈
  - MS사에서 제공되는 라이브러리에 종속적

임베디드 리눅스

- 정의 :
  - 저 성능의 프로세서와 소용량의 메모리를 가진 임베디드 시스템용으로 개발된 Linux

- 임베디드 리눅스 장점
  - 기능성과 확장성이 우수 (리눅스 이용에 따른 장점) :
    - 단위 모듈로 설계
    - 완전한 운영체제 (file system, network 지원, 각종 응용 SW 수행)
  - PowerPC, ARM, MIPS 등 다양한 CPU Platform지원함
  - Open Source
    - 로열티가 없으므로 경제성이 높음
  - 사용자 층이 넓어 오류 수정이 빠르고 안정성이 우수
    - 지속적 upgrade, Kernel 2.8.x, OS 10.0 (?)
    - Engineer 확보 유리
  - 기존의 데스크 탑 개발 환경과 동일하여 개발이 용이함
    - Cross-compiler, debegger, GUI (Qt/E, Qutopia) 등
  - 많은 장치의 드라이버를 제공: *JTAG, PCMCIA, PDA LCD, I/O, IDE*
  - 다양한 개발 환경의 제공: Cross-compiler, debeger, GUI (Qt/E, Qtopia) 등


Linux – Wikipedia, the free encyclopedia – Windows Internet Explorer
http://en.wikipedia.org/wiki/Linux
Linux – Wikipedia, the free encyclopedia
Try Beta
Log in / create account
article
discussion
edit this page
history
Linux
From Wikipedia, the free encyclopedia
This article is about operating systems that use the Linux kernel. For the kernel itself, see Linux kernel. For other uses, see Linux (disambiguation).
Linux
navigation
Main page
Contents
Featured content
Current events
Random article
search
Go
Search
interaction
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Donate to Wikipedia
Help
toolbox
What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Cite this page
Contents [hide]
1 History
1.1 Unix
1.2 GNU
1.3 MINIX
1.4 Commercial and popular uptake
1.5 Current development
2 Design
OS family
Unix-like
Working state
Current
Source model
Free and open source software
Latest stable release
Kernel
2.6.32.8
Latest unstable release
Kernel
2.6.33-rc8
완료
인터넷
100%



- 1. Unix-like computer operating systems based on the Linux kernel
- 2. GNU General Public License
- 3. Typically Linux is packaged in a format known as a Linux distribution
- 4. Linux distributions include the Linux kernel and all of the supporting software required to run a complete system, such as utilities and libraries, the X Window System, the GNOME and KDE desktop environments,
- 5 The name "Linux" comes from the Linux kernel, originally written in 1991 by Linus Torvalds.

임베디드 리눅스 (계속.)

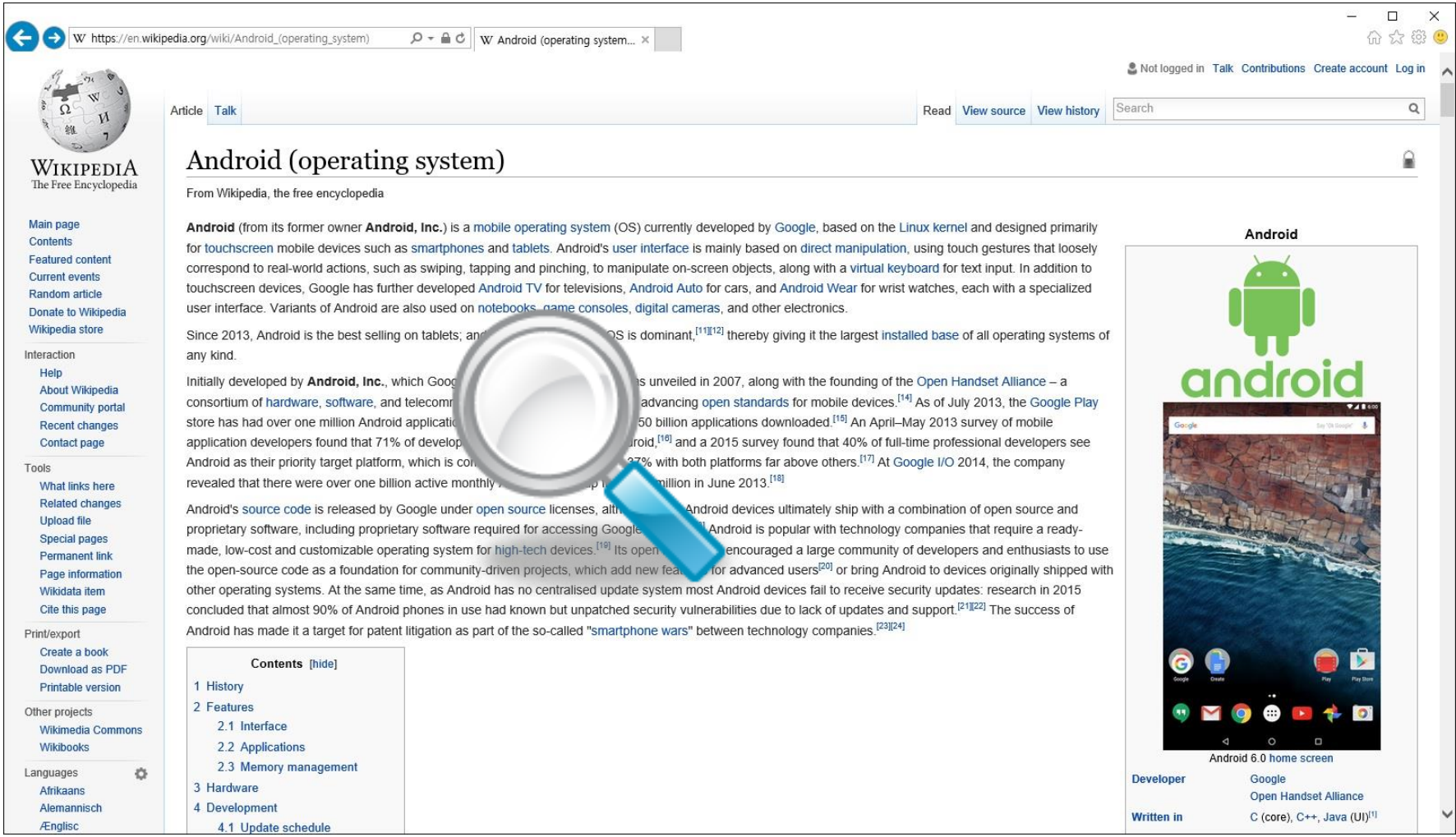
- 임베디드 리눅스 단점
  - 기존의 RTOS보다 많은 메모리를 요구함
  - 범용 OS로 설계되어 Real-Time을 지원하지 못함
  - GUI 환경을 개발하기 어려움
  - 제품화하기 위한 솔루션 구성이 어려움
  - 많은 업체들과 개발자들이 독자적으로 개발하고 있어 표준화가 어려움

- Linux가 실시간성을 지원하지 못하는 이유
  - High Interrupt Latency
  - High Scheduler Latency
  - Various OS Service Time (메모리 할당, IPC, 비예측성 operation, 가상메모리, 시스템 콜 등)

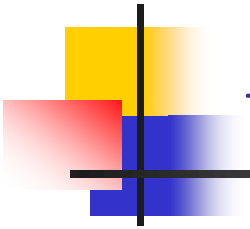
➔ Embedded Linux에서는 위 어려움을 없애거나 문제점을 줄여야 한다.

임베디드 운영체제 – 최근각광받는 OS

- iOS (iphone OS)
  - ipod, iphone, ipad에 사용하기 위해 Apple 사에서 개발한 임베디드 운영체제
  - Mac OS X 10.5 기반(Unix 계열)
  - Core OS, Core Service, Media, User Interface(코코아)의 네 개의 layer로 구성됨
  - 애플 고유의 SDK(software development kit)와 아이폰 simulator 제공
- Android
  - Open Handset Alliance의 모바일 시스템용 운영체제의 표준을 위해 Google 에 의해 개발됨
  - Linux 커널 (모놀리틱 커널), 휴대전화 및 모바일 디바이스용 미들웨어 응용 프로그램
  - Java Virtual machine을 포함함 (Java 기반 응용 프로그램 개발/사용가능)
  - 안드로이드 SDK 및 emulator제공


https://en.wikipedia.org/wiki/Android_(operating_system)
Android (operating system...
Not logged in Talk Contributions Create account Log in
Article Talk
Read View source View history
Search
WIKIPEDIA
The Free Encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page
Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page
Print/export
Create a book
Download as PDF
Printable version
Other projects
Wikimedia Commons
Wikibooks
Languages
Afrikaans
Alemannisch
Ænglisc
Android (operating system)
From Wikipedia, the free encyclopedia
Android
android
Android 6.0 home screen
Developer
Google
Open Handset Alliance
Written in
C (core), C++, Java (UI)[1]
Contents [hide]
1 History
2 Features
2.1 Interface
2.2 Applications
2.3 Memory management
3 Hardware
4 Development
4.1 Update schedule

- **Android** (from its former owner **Android, Inc.**) is a mobile operating system (OS) currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.



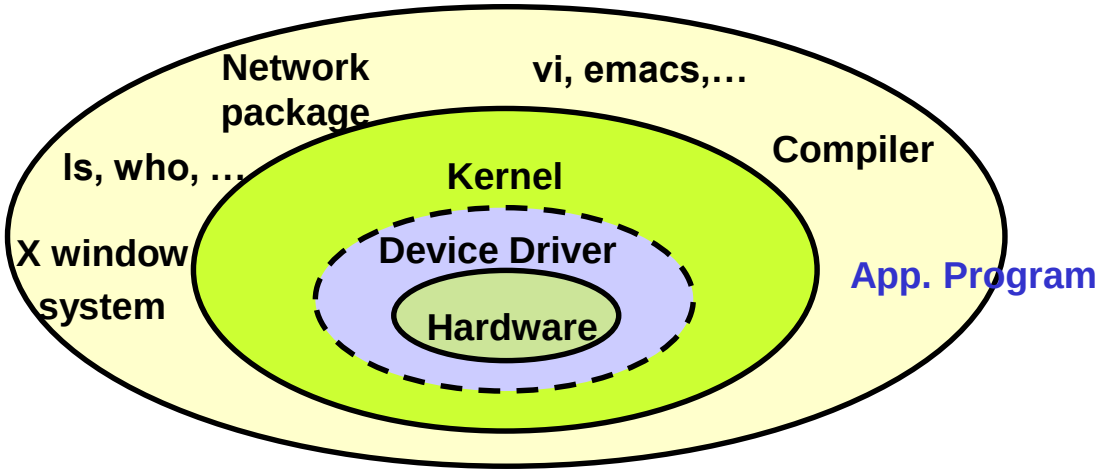
목차

- 임베디드 시스템과 운영체제
- 운영체제 Review
- µC/OS-II – the first program

운영체제 Review

- 운영체제란?
  - 컴퓨터 시스템의 전반적인 동작을 제어하고 조정하는 시스템 프로그램들의 집합. (하드웨어와 프로그램의 인터페이스)
  - 자원 관리자 (Resource Manager) : who gets the CPU? when I/O takes place? how much memory is allocated?
  - 응용프로그램에게 자원에 대한 서비스 제공 (Computing Environments)

- 자원의 종류
  - 물리적인 자원: 처리기, 메모리, 디스크, 터미널, 네트웍, ...
  - 추상적인 자원: 태스크, 세그먼트/페이지, 파일, 드라이버, 통신 프로토콜, 패킷, 보안, ...
  - 가장 중요한 자원은?



운영체제의 주요 개념

- 태스크
  - 수행중인 프로그램 (an instance of a running program)
  - 프로그램의 수행 환경 (an execution environment of a program)
  - 스케줄링 단위 (scheduling entity)
  - 제어 흐름과 주소 공간의 집합 (a control flow and address space)

- 멀티 태스킹
  - 여러 개의 태스크를 동시에 실행시키는 것
  - 일반 OS 에서의 태스크
    - 각 태스크들은 대분분 무관한 프로그램임
  - 임베디드 시스템에서의 태스크
    - 하나의 큰 응용 프로그램을 논리적으로 나눈 것
      - 기능상 매우 밀접한 관계
      - 태스크 사이에 이루어지는 작업들이 많다.
    - 응용 프로그램을 실행을 위해 여러 기능들이 동시 실행 요구
      - 순차적이 아닌 동시 실행의 필요성이 있다.

운영체제의 주요 개념-태스크 스케줄링

- 스케쥴러(Scheduler)
  - OS의 핵심기능
  - 다음 번에 어떤 태스크를 실행해야 하는 지를 결정하는 코드 부분
  - 태스크 선택 정책 : 우선순위 기반의 스케쥴링
    - FIFO(First In First Out), Round-robin 등
  - 선점(Preemptive)
    - 어떤 태스크가 수행되고 있을 때 커널이 중간에 그 태스크의 수행을 중지 시키고 다른 태스크의 기능을 수행시키는 기능
    - 선점형 커널 / 비선점형 커널
  - 다른 태스크로 실행이 넘어갈 때 문맥전환(context switching) 발생

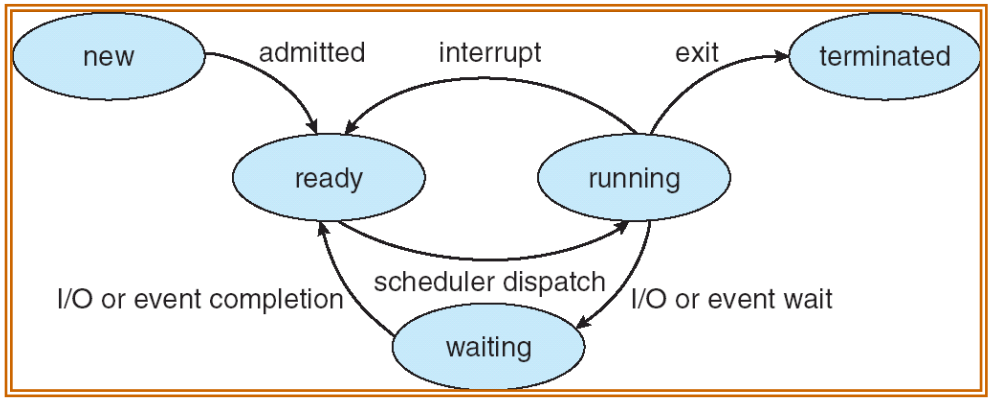
- Context switching
  - 일단 현재 수행 중인 태스크 상황 하에서의 시스템 상태(문맥)를 TCB(Task Control Block)이라는 특정한 자료구조에 저장하고, 다음에 새로운 태스크의 문맥을 가져와 시스템 상태를 복원한 후에 실행하는 것
  - Context switching은 overhead이기 때문에 짧을 수록 효율적 임
    - thread의 개념을 통해 이를 보완하는 방법

운영체제의 주요 개념

- Interrupt service
  - Asynchronous event를 CPU에 알리는 방법
  - Interrupt는 외부에서 들어오는 중요한 신호로서 시간에 민감한 경우가 있기 때문에 interrupt latency가 짧은 것이 좋음.
  - ISR자체도 짧은 것이 좋은데 그 이유는 ISR 자체가 길어지면 interrupt nesting이 되기 쉽기 때문.
  - ISR에서는 보통 그에 상응하는 태스크 수준의 service routine을 부르고(HISR) 끝나도록 구성.
  - HISR에서는 마치 태스크처럼 존재해서 수행이 된다.

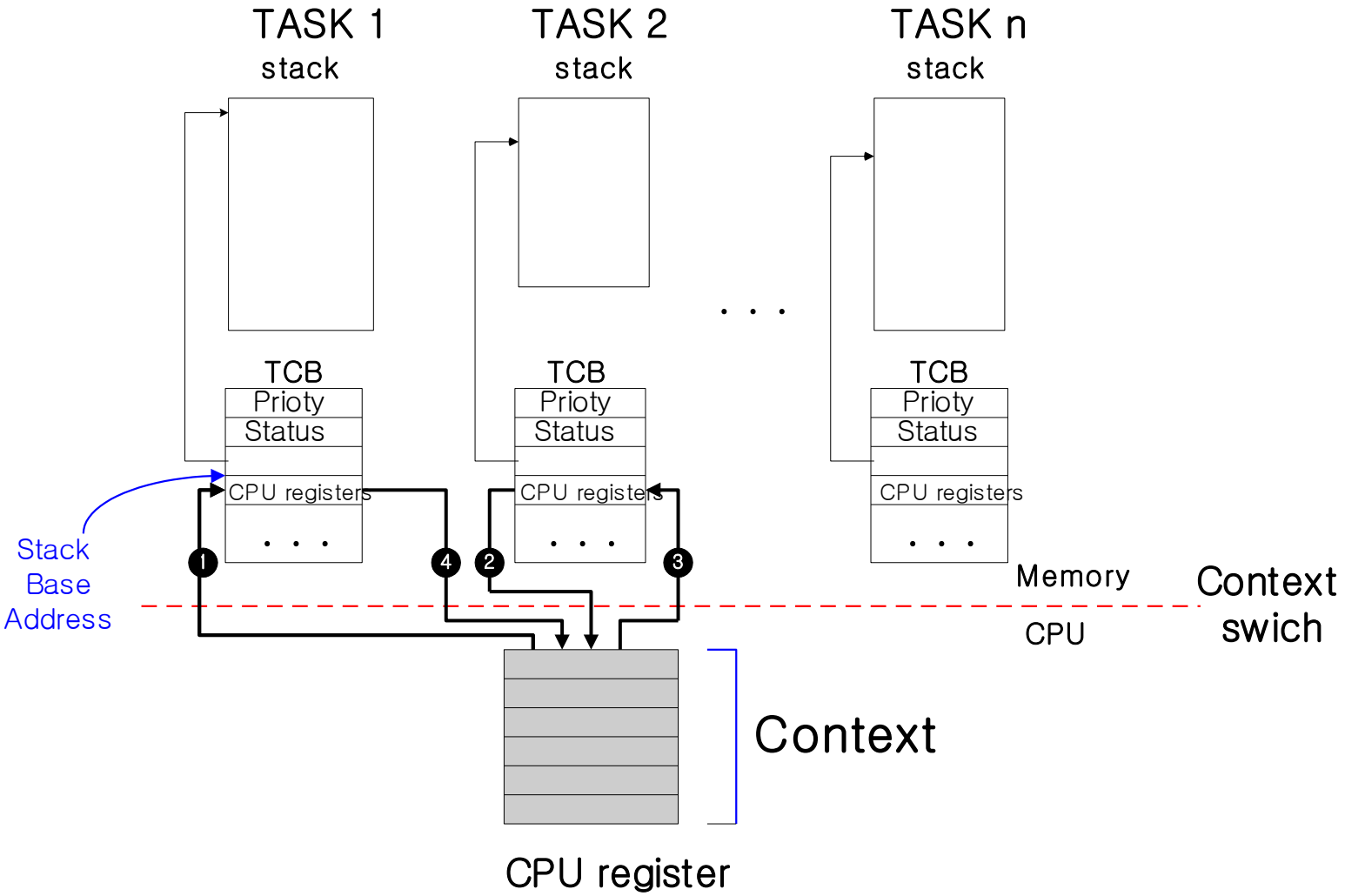
스케줄러 (1)

- 태스크
  - 여러 개가 독립적으로 존재하면서 CPU 실행 시간을 얻기 위해 서로 경쟁하는 실행 코드 (프로세스+쓰레드)
  - 우선순위 부여
  - 문맥 (context)
    - CPU 레지스터와 스택 영역 할당
    - TCB (Task Control Block)에 저장
  - 스케줄러에 의해서 스케줄링



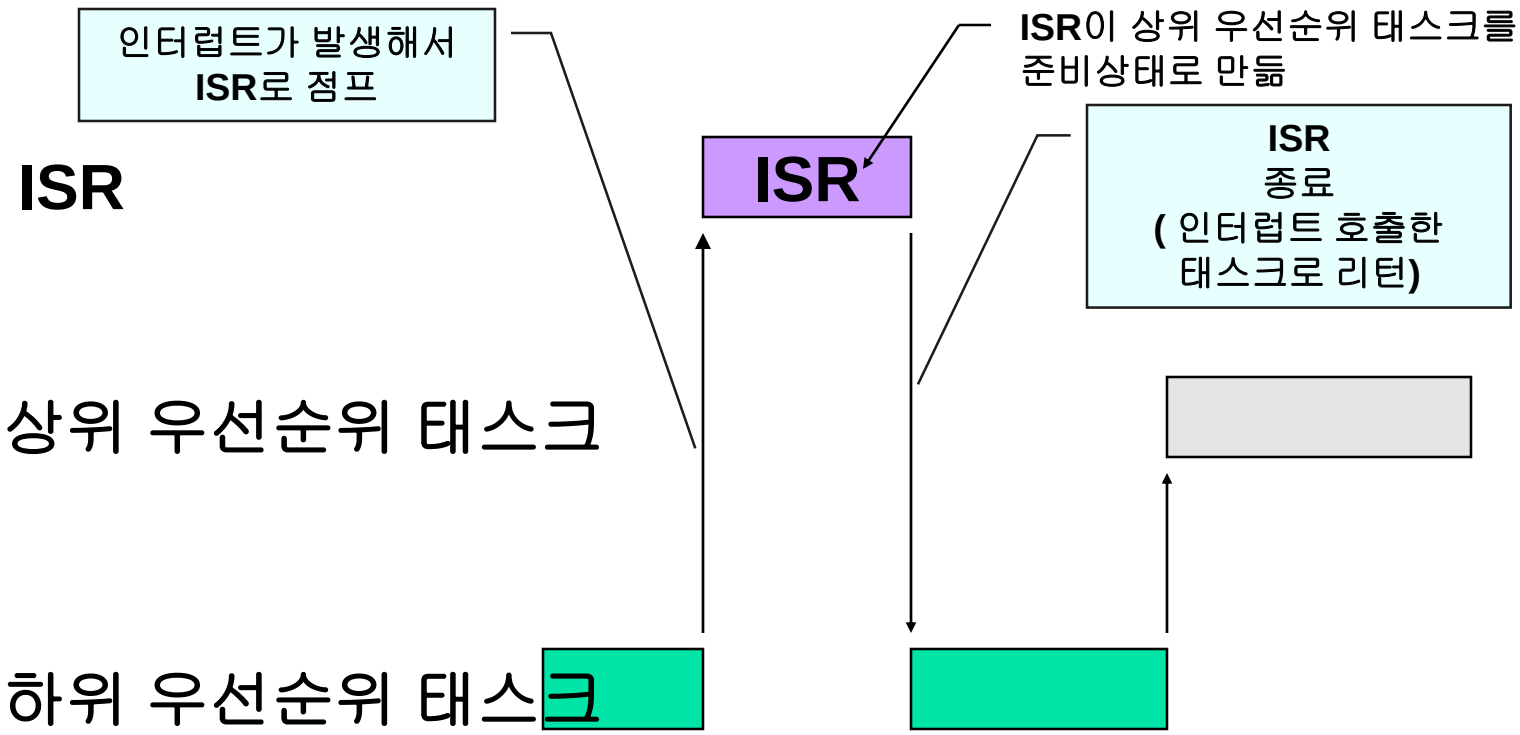
스케줄러 (2)

- 문맥 전환


TASK 1
stack
TASK 2
stack
TASK n
stack
TCB
Prioty
Status
CPU registers
TCB
Prioty
Status
CPU registers
TCB
Prioty
Status
CPU registers
Stack
Base
Address
1
4
2
3
Memory
CPU
Context
swich
Context
CPU register

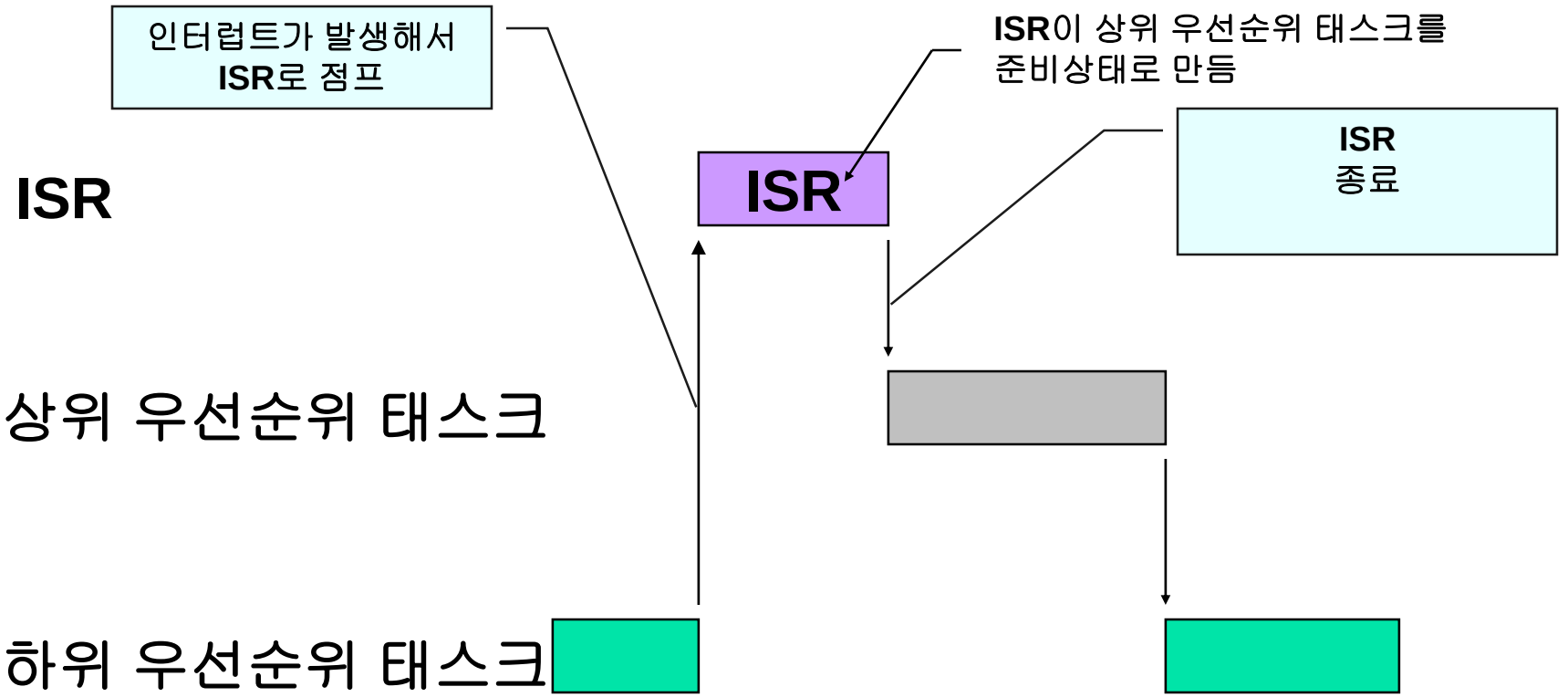
스케줄러 (3)

- 비선점형 커널



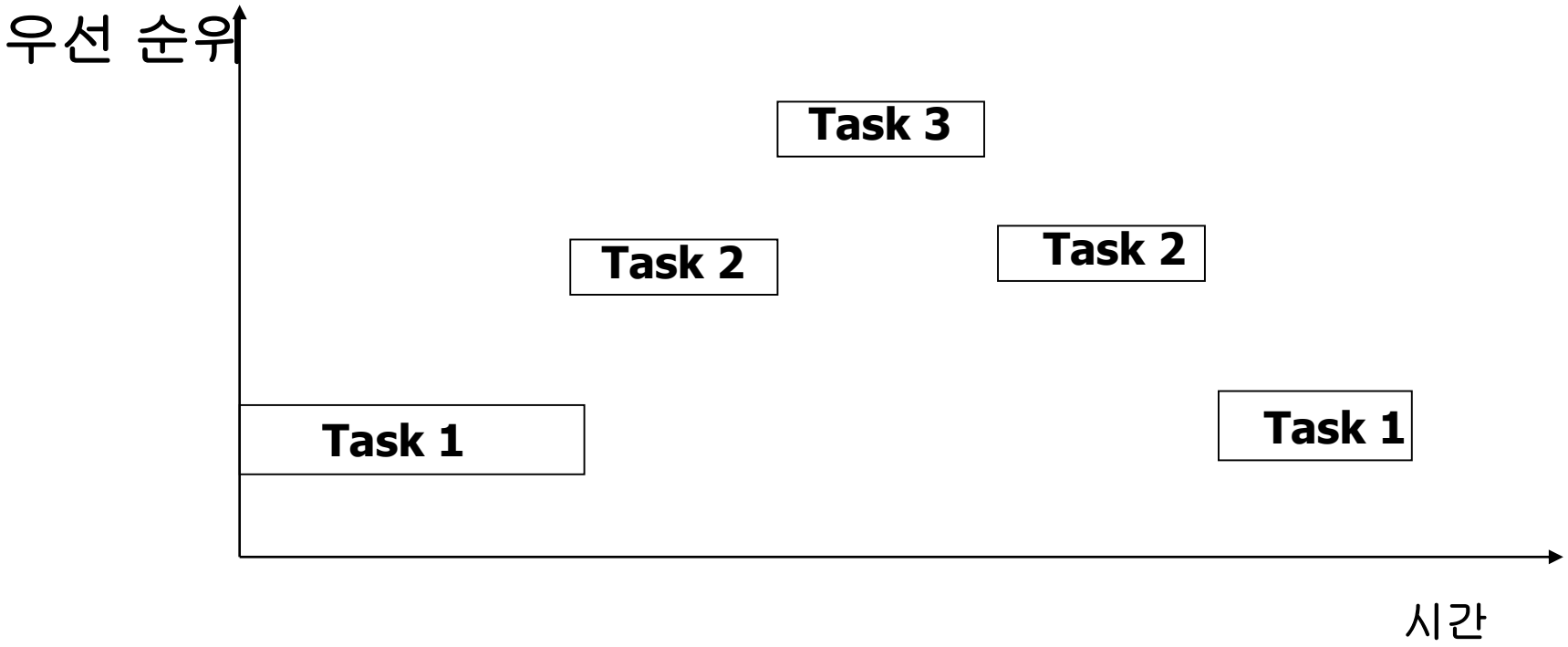
스케줄러 (4)

- 선점형 커널


인터럽트가 발생해서
ISR로 점프
ISR이 상위 우선순위 태스크를
준비상태로 만듬
ISR
종료
ISR
ISR
상위 우선순위 태스크
하위 우선순위 태스크

스케줄러 (5)

- 우선 순위 기반 선점형 스케줄링



운영체제의 주요 개념-상호배제

- Mutual exclusion (상호배제)
  - 두 개의 태스크가 동시에 하나의 공유자원에 접근하려고 할 때 한 태스크에게 자원 사용에 대한 배타적 권리를 보장하는 것
  - Critical section : 공유자원을 access하는 일련의 코드부분
    - 다른 태스크에 의해서 중단되어서는 안 되는 일련의 명령 혹은 코드 블록
- 상호배제 기법
  - 인터럽트 발생을 방지
    - Critical section에 들어가기 전에 인터럽트를 disable시키고 (CLI) 빠져 나오면서 인터럽트를 다시 enable 시키는 방법 (STI)
    - 단일 CPU의 경우 단순하게 사용 가능
  - semaphore 이용
    - semaphore를 얻지 못하면 공유자원을 얻을 수 없으며 일단 semaphore를 얻으면 공유자원을 마음 놓고 쓸 수 있다.
    - 다른 태스크를 위해서 공유자원을 다 쓰면 semaphore를 풀어야 한다.
    - Semaphore가 0 이면 waiting한다.

상호배제-Semaphore

- Semaphore
  - 공유변수 사용
  - mutual exclusion을 만들어 공유자원을 제어
  - 태스크 사이의 동기화에 사용 가능
  - 사용 방법
    - 공유자원에 해당하는 semaphore를 만든다.
    - 공유자원을 사용하기 직전에 해당 semaphore 얻는다.
      - 변수 값이 0이 아닌 경우 그 변수 값을 1 감소 시킨다.
      - 0이면 양수가 될 때까지 기다린다. (SLEEP상태)
    - 공유자원을 다 쓰면 그 값을 1 증가 시킨다.
  - 종류
    - Binary semaphore : 공유자원의 변수가 0,1인 경우
    - Counting semaphore : 1이상의 값

상호 배제-example

- 공유 리소스 문제
  - HHeelloo, WWororlldd
- 인터럽트 비활성화 및 활성화

```
void Function (void)
{
   OS_ENTER_CRITICAL(); // 인터럽트 비활성화
   // 공유 데이터 엑세스
   OS_EXIT_CRITICAL(); // 인터럽트 활성화
}
```

상호 배제-example (cont.)

- 스케줄러 비활성화와 활성화

```
void Function (void)
{
   OSSchedLock(); // 스케줄링 비활성화
   // 공유 데이터 엑세스
    OSSchedUnLock(); // 스케줄링 활성화
}
```

- 세마포어

```
void Function (void)
{
.
WAIT();
Out_Printer("Hello, World");
SIGNAL();
.
}
```

상호 배제-example (cont.)

- 세마포어 은닉

```
int CommSendCmd(char *cmd, char *response, int timeout)
{
    Acquire port's semaphore;
    Send command to device;
    Wait for response(with timeout);
    if (timed out) {
        Release semaphore;
        return (error code);
    } else {
        Release semaphore;
        return (no error);
    }
}
```

- 교착 상태(Deadlock)

운영체제의 주요 개념-태스크간 통신

- Task communication
  - 태스크간에 통신하는 방법
    - Global variable을 쓰는 방법 / message passing 방법
  - Global variable
    - exclusive access를 해야 하며, ISR이 포함된 인터럽트를 disable해야 한다.
    - Task 사이에서는 인터럽트를 disable하는 방법 외에 semaphore를 사용할 수 있다.
  - message passing
    - Mailbox, queue, pipe – message의 크기에 따라 결정됨

- Task synchronization
  - 태스크 간의 동기화 기능
  - Semaphore/event flag/signal 등을 사용

운영체제의 주요 개념-재진입코드

- Reentrancy code
  - 인터럽트, 선점의 개념과 연관
  - 하나의 함수를 여러 태스크가 동시에 수행 가능
  - ‘Reentrant하다’
    - 태스크 1이 함수 1을 수행하다가 태스크 2로 제어가 넘어가고 태스크 2가 이 함수를 호출해도 함수 1이 제대로 동작하는 것
    - 전역변수를 사용하지 말 것 - 전역 변수는 shared resource이기 때문에 mutual exclusive 하도록 만들어 주지 못하는 문제가 생긴다.
    - Reentrant하지 못한 code는 공유하지 않거나, 공유해야 하는 경우에는 semaphore를 쓴다거나 동일 우선 순위를 갖는 태스크 사이에서 round-robin을 하지 않아야 한다.

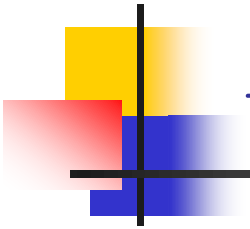
재진입 가능 함수

재진입 가능함수

```
void swap(int* x, int* y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

재진입 불가능함수

```
int temp;

void swap(int* x, int* y)
{
     temp = *x;
    *x = *y;
    *y = temp;
}
```



목차

- 임베디드 시스템과 운영체제
- 운영체제 Review
- µC/OS-II – the first program

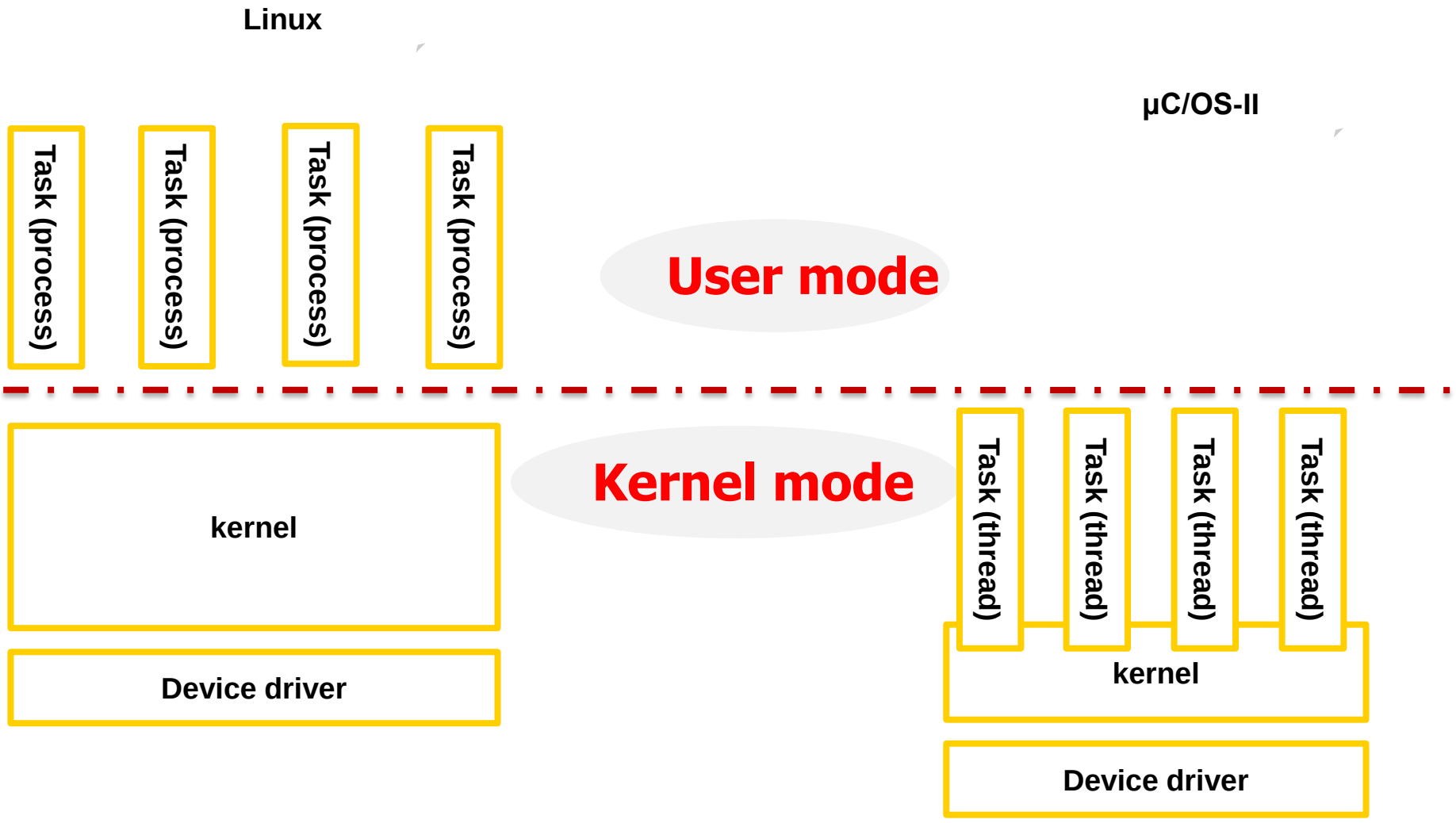
µC/OS-II 개요

- Micro-Controller Operating Systems, Version 2
  - Jean J. Labrosse가 만든 실시간 운영체제 (1992년)
  - 학교나 개인의 교육과 같은 비상업적 목적에 한해 자유로이 사용 가능한 공개소스
    - 소스코드의 수정 및 커널의 내부 구조를 이해하기 용이
  - 상업적 용도 : 각종 장비 개발이 가능
    - Avionics, medical devices, mobile handsets, consumer electronics 등
    - 상업적인 목적에 사용될 경우 라이센스를 따로 얻어야 함
  - 공식 사이트:
    - http://www.micrium.com/page/products/rtos/os-ii
  - 현재는 version 3까지 나와있음

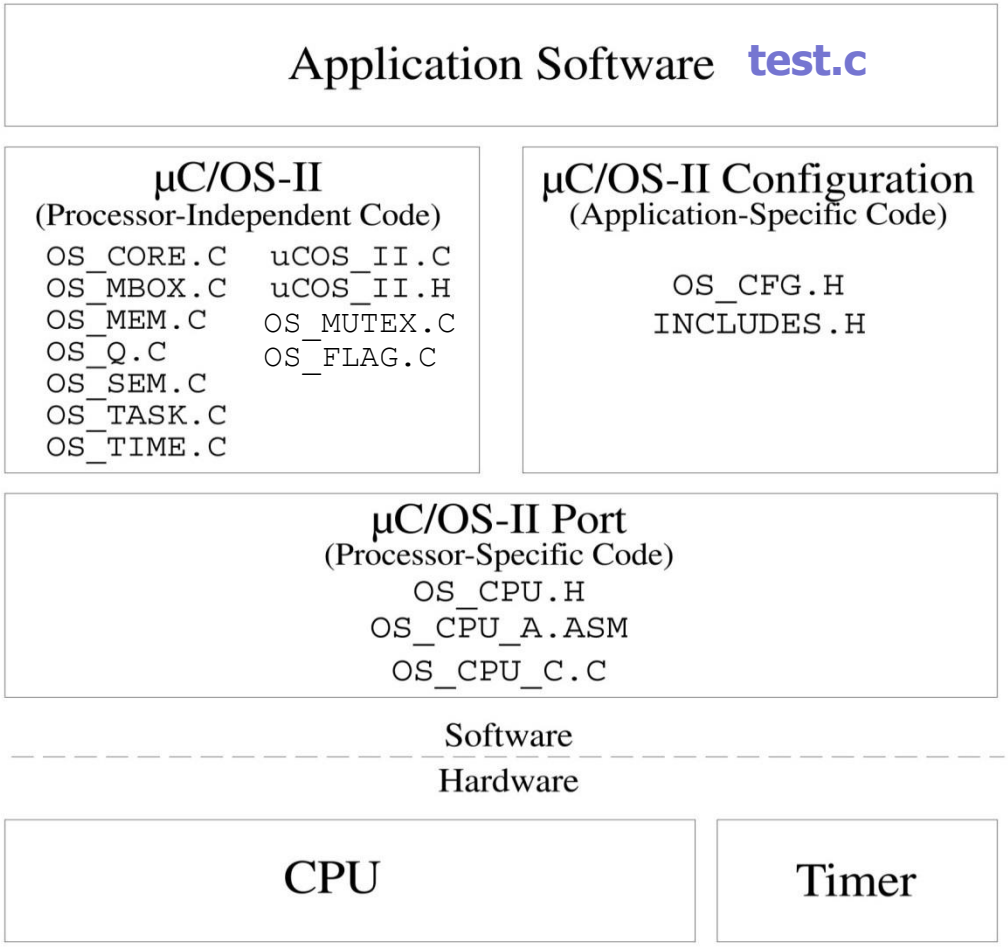
특징

- Source code
  - 깔끔한 코드
  - Very Small (20Kb의 공간만 있어도 실행가능)
- Portable
  - 커널 코드의 대부분이 이식 가능한 ANSI C를 기반 (5500 line에 불과)
    - 일부 마이크로프로세서에 국한된 부분은 어셈블러로 코딩
  - 8Bit, 16Bit, 32Bit 및 64Bit, DSP로도 Porting 가능
- Reliable
  - Safety-critical system 에도 사용할 수 있는 안전한 운영체제
    - FAA(Federal Aviation Administration) 승인 (2000년 7월)
- ROMable
  - 해당 C 컴파일러, 어셈블러, 링커, 로더가 있으면 내장이 용이
- Preemptive real-time kernel (선점형 실시간 코드)
  - 높은 우선 순위 작업이 먼저 수행
- Multitasking
  - 최대 64개의 태스크 지원
  - 일부 task는 OS에서 사용하도록 정해져 있음

일반 운영체제와의 큰 차이점

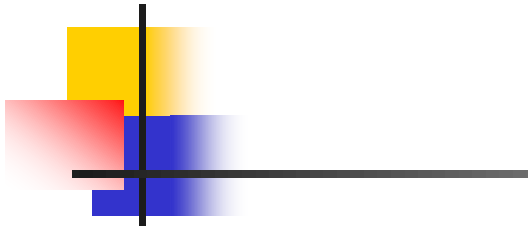


The µC/OS-II File Structure



First program –소스 구조

- uC/OS-II source code
  - \SOFTWARE 루트디렉토리
  - \SOFTWARE\BLOCKS PC관련기능처리
  - \SOFTWARE\TO 유틸리티
  - \SOFTWARE\uCOS-II 메인디렉토리
  - \SOFTWARE\uCOS-II\EX1_x86L 예제 1
  - \SOFTWARE\uCOS-II\EX2_x86L 예제 2
  - \SOFTWARE\uCOS-II\EX3_x86L 예제 3
  - \SOFTWARE\uCOS-II\EX4_x86L 예제 4
  - \SOFTWARE\uCOS-II\Ix86L x86 리얼모드 포팅
  - \SOFTWARE\uCOS-II\Ix86L-FP x86 FP 리얼모드 포팅
  - \SOFTWARE\uCOS-II\SOURCE uC/OS-II 코드



- 소스 코드 내용
  - C:\Software\uCOS-II\EX1_x86L\BC45\MAKE
    - MAKEAPP.BAT : Create EX1.EXE file
    - MAKECLEAN.BAT : Remove temporary files
    - EX1.MAK : Makefile
    - EX1.LIN : Linking files
  - C:\Software\uCOS-II\EX1_x86L\BC45\SOURCE
    - INCLUDE.H : Header file configuration
    - OS_CFG.H : OS configuration
    - EX1.C : Example # 1 source code

test.c

```
#include "includes.h"                                                    (1)

#define  TASK_STK_SIZE    512       /* 각 Task의 스택 크기(WORD 항목 단위)      */
#define  N_TASKS          10        /* 생성할 Task 수                          */

OS_STK   TaskStk[N_TASKS][TASK_STK_SIZE];     /* Task 스택                  */
OS_STK   TaskStartStk[TASK_STK_SIZE];
char     TaskData[N_TASKS];                    /* 각 Task에 넘겨줄 전달인자   */
OS_EVENT      *RandomSem;                    /* 세마포어  */
```

필요한 모든 헤더 파일을 마스터 헤더 파일인 '**INCLUDES.H**'에 포함되어 있음

```
void main (void)
{
    PC_DispClrScr(DISP_FGND_WHITE + DISP_BGND_BLACK);      /* DOS 화면 지우기 */
    OSInit();                                            /* uC/OS-II 초기화  (1)*/
    PC_DOSSaveReturn();                                    /* DOS 환경 저장 */
    PC_VectSet(uCOS, OSCtxSw);                        /* uC/OS-II 문맥전환 벡터 설정 */
    RandomSem = OSSemCreate(1);                        /* Random Number용 세마포어 */
    OSTaskCreate(TaskStart, (void *)0, (void *)&TaskStartStk[TASK_STK_SIZE-1],
                 0);                                                          (2)
    OSStart();                                           /* 멀티태스킹 시작 (3) */
}
```

(2) 멀티태스킹을 시작하기 전에 최소한 하나 이상의 **Task**를 생성해야만 함

- 이 예제에서는 이 **Task**의 함수를 **TaskStart()**라고 명명

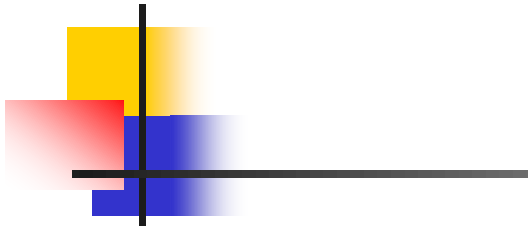
(3) 멀티태스킹을 시작하기 위해 **OSStart()**함수를 호출해서 **uC/OS-II**로 제어를 넘김

- **OSStart()**를 호출하기 전에 반드시 **1**개 이상의 **Task**를 생성해야만 함

```
void TaskStart (void *data)
{
    UBYTE  i;
    char   s[100];
    WORD   key;

    data = data;                                          /* 컴파일러 경고 방지 */

    OS_ENTER_CRITICAL();
    PC_VectSet(0x08, OSTickISR);                    /* uC/OS-II 클럭 틱 ISR 설치 */
    PC_SetTickRate(OS_TICKS_PER_SEC);               /* 틱 주기 재설정            */
    OS_EXIT_CRITICAL();

    OSStatInit();                                 /* uC/OS-II 통계모듈 초기화  (1) */
    TaskStartCreateTasks();                                                  (2)

    for (;;) {
       if (PC_GetKey(&key) == TRUE) {             /* 키가 눌렸는지 확인          */
            if (key == 0x1B) {                    /* 예, ESCAPE 키인지 확인      */
                PC_DOSReturn();                   /* DOS로 복귀                  */
            }
        }
        OSCtxSwCtr = 0;
        OSTimeDlyHMSM(0, 0, 1, 0);                /* 1초 지연                    */
    }
}
```

```
static void TaskStartCreateTasks ( void )
{
    INT8U  i;

    for ( i = 0; i < N_TASKS; i++ )    {
       TaskData [ i ] = '0' + i ;
       OSTaskCreate ( Task , ( void * ) & TaskData[ i ] ,
                          &TaskStk [ i ] [ TASK_STK_SIZE - 1], i + 1 ) ;    (1)
    }
}
```

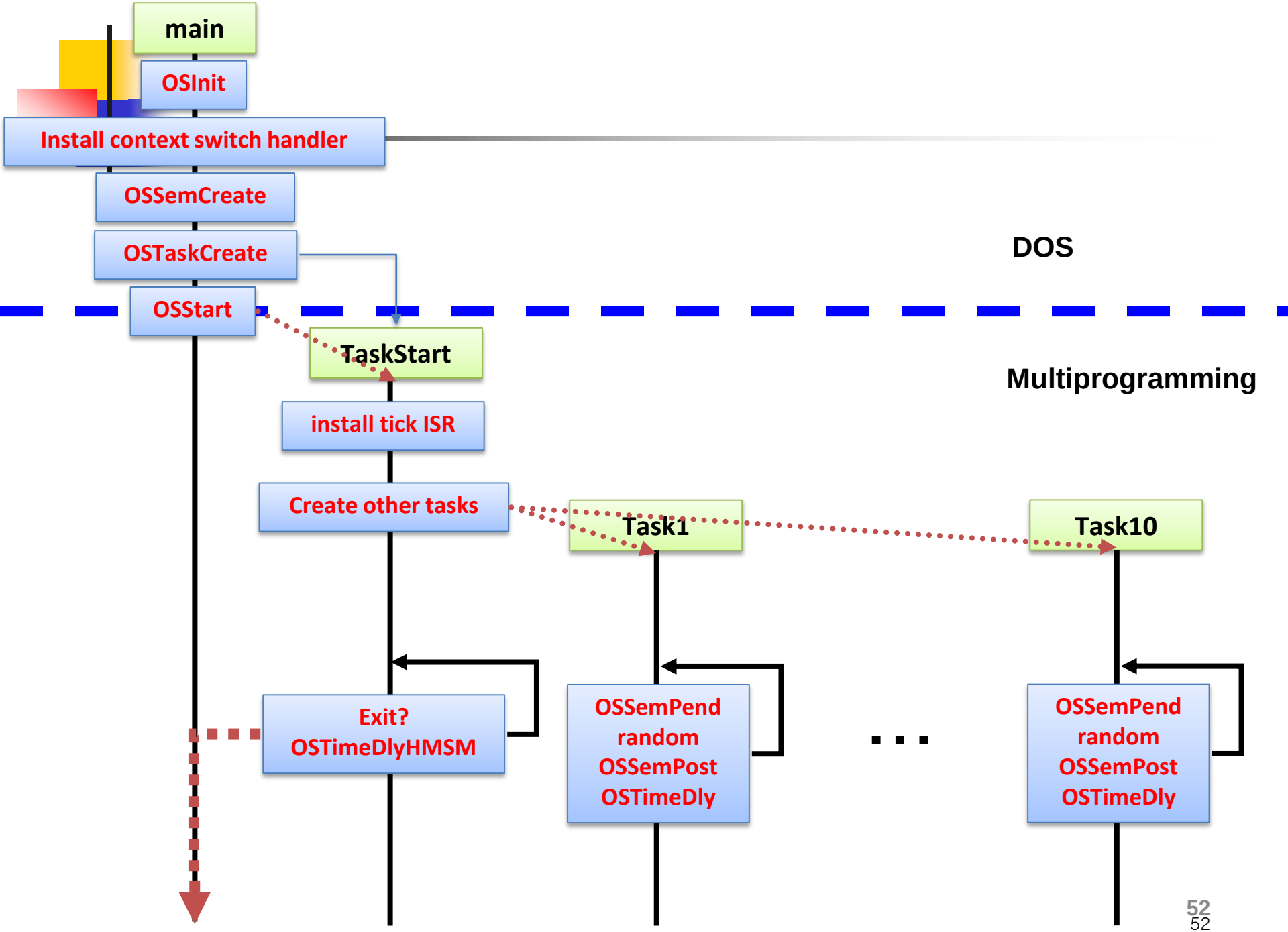
- **for** 루프는 **Task()**라는 **N_TASKS**개의 동일 **Task**를 초기화
  <각 **Task**는 **1**부터 **10**까지의 고유한 우선순위 숫자를 가짐**>**
- 우선순위 숫자가 낮을수록 우선순위가 높음



```
void Task (void *data)
{
  INT8U x;
  INT8U y;
  INT8U err;

  for (;;) {
    OSSemPend(RandomSem, 0, &err); /* 랜덤 넘버 함수를 사용하기 위해 세마포어 획득 */
    x = random(80);                /* 태스크 번호를 표시할 X 좌표 결정           */
    y = random(16);                /* 태스크 번호를 표시할 Y 좌표 결정           */
    OSSemPost(RandomSem);          /* 세마포어 반납                              */
    PC_DispChar(x, y + 5, *(char *)data, DISP_FGND_LIGHT_GRAY);
    OSTimeDly(1);                  /* 1 클럭 틱 지연                    (1)      */
  }
}
```

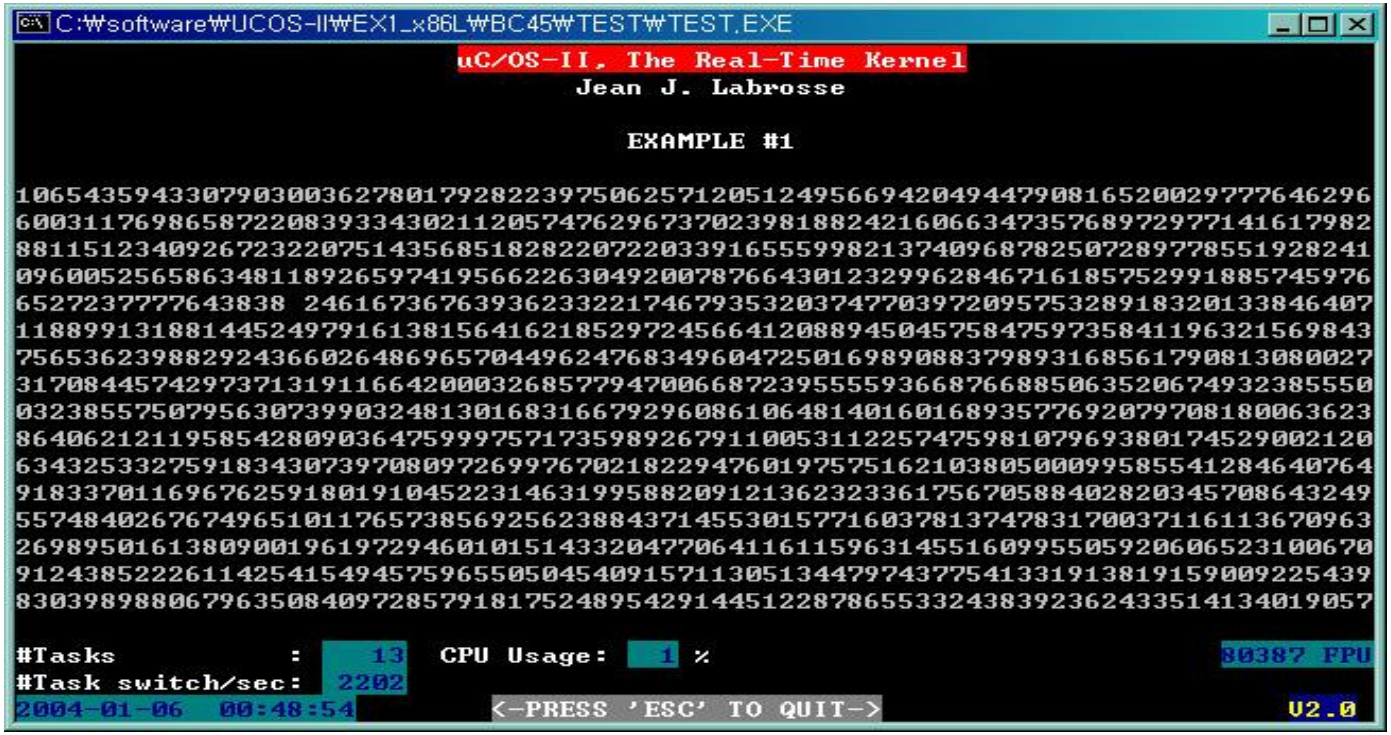
Semaphores

- A semaphore consists of a wait list and an int eger counter.
  - OSSemPend():
    - Counter--
    - If the value of the semaphore < 0, then the task is blocke d and moved to the wait list immediately.
    - A time-out value can be specified.
  - OSSemPost():
    - Counter++
    - If the value of the semaphore ≧ 0, then a task in the wait list is removed from the wait list.
    - Reschedule if needed.


main
OSInit
Install context switch handler
OSSemCreate
OSTaskCreate
OSStart
DOS
TaskStart
Multiprogramming
install tick ISR
Create other tasks
Task1
Task10
Exit?
OSTimeDlyHMSM
OSSemPend
random
OSSemPost
OSTimeDly
OSSemPend
random
OSSemPost
OSTimeDly

52

- Example 1 -multitasking
  - 기본적인 멀티태스킹 수행 과정을 보여줌
  - 10개의 Task가 화면상 임의 위치에 각각 0에서 9사이의 숫자를 표시



Example 1: Multitasking

- Summary:
  - µC/OS-II is initialized and started by calling OSInit() and OSStart(), respectively.
  - Before µC/OS-II is started,
    - The DOS status is saved by calling PC_DOSSaveReturn().
    - A context switch handler is installed by calling PC_VectSet().
    - One user task must be created first!
  - Shared resources can be protected by semaphores.
    - OSSemPend(), OSSemPost().