1. 운영체제의 자원들

운영체제가 관리해야 할 자원은 크게 물리적인 자원과 추상적인 자원으로 구분할 수 있다.

(1) 물리적인 자원: CPU, 메모리, 디스크, 터미널, 네트워크 등 시스템을 구성하고 있는 요소들과 주변장치
(2) 추상적 자원: cpu를 추상화시킨 task, 메모리를 추상화시킨 세크먼트와 페이지, 디스크를 추상화시킨 파일, 네트워크를 추상화시킨 통신 프로토콜, 패킷.

따라서 리눅스 커널 내부에는 태스크 관리자, 메모리 관리자, 파일시스템, 네트워크 관리자, 디바이스 드라이버 관리자 등 크게 5가지 부분으로 나뉜다.

(1) 태스크 관리자: 태스크의 생성, 실행, 상태 전이, 스케쥴링, 시그널 처리, 프로세스 간 통신
(2) 메모리 관리자: 물리 메모리 관리, 가상 메모리 관리, 세그멘테이션, 페이징
(3) 파일 시스템: 파일의 생성, 접근 제어, inode 관리, 디렉터리 관리, 수파 블록 관리
(4) 네트워크 관리자: 소켓 인터페이스, TCP / IP 등의 통신 프로토콜 서비스 제공
(5) 디바이스 드라이버: 디스크나 터미널, cd, 네트워크 카드 등과 같은 주변 장치를 구동하는 드라이버들

2. 리눅스 커널 컴파일

실행 파일 생성법 :

```
$vi hello.c
$gcc -O -o hello hello.c
$ls
hello    hello.c
$./hello
Hello Linux
```

hello.c ->

```
#include <stdio.h>
int main() {
        printf(“Hello Linux\n”);
        return 0;
}
```

리눅스 커널 또한 hello 실행 파일을 만드는 것 처럼 만들 수 있다. 다만 커널을 만들 때는 많은 소스 파일들을 기반으로 컴파일 해야 하기 때문에 make 유틸리티를 사용하게 된다.

3. 리눅스 커널 제작 과정

(1) 커널 구성:

커널 구성이란 새로 만들어질 리눅스 커널에게 현재 시스템에 존재하는 하드웨어 특성, 커널 구성 요소, 네트워크 특성 등의 정보를 알려주는 과정이다. 보통 이 과정은 매우 복잡하며, 자신이 가지고 있는 시스템의 하드웨어 정보들에 대한 사전 지식이 필요하다.
커널 구성을 수행하는 방법은 make config, make menuconfig, make xconfig 등의 방법이 있다.

(2) 커널 컴파일:

커널 구성이 완료되면 그 다음 단계는 커널 컴파일이다. 이 단계는 커널 소스 파일을 이용해 실행 가능한 커널을 만드는 과정이다. 커널 버전 2.6 이후부터는 단순히 “make” 만 타이핑해도 된다.

(3) 커널 인스톨:
커널 인스톨이란 생성된 커널로 시스템이 부팅될 수 있도록 만드는 과정이다. 구체적으로 커널 인스톨은 생성된 커널 이미지를 루트 파일 시스템으로 복사, 모듈 인스톨, 그리고 부트 로더 수정 등의 과정으로 이루어진다.

그냥 **컴파일러**로 **컴파일**하면 되지 왜 굳이 `Makefile`을 만들고 `make`명령을 실행해야 하나?

1. 각 파일에 대한 반복적 명령의 자동화로 인한 시간 절약
2. 프로그램의 종속 구조를 빠르게 파악 할 수 있다.
3. 단순 반복 작업 및 재작성을 최소화

`Makefile`은 다음과 같은 구조를 가진다.

-목적파일(Target) : 명령어가 수행되어 나온 결과를 저장할 파일

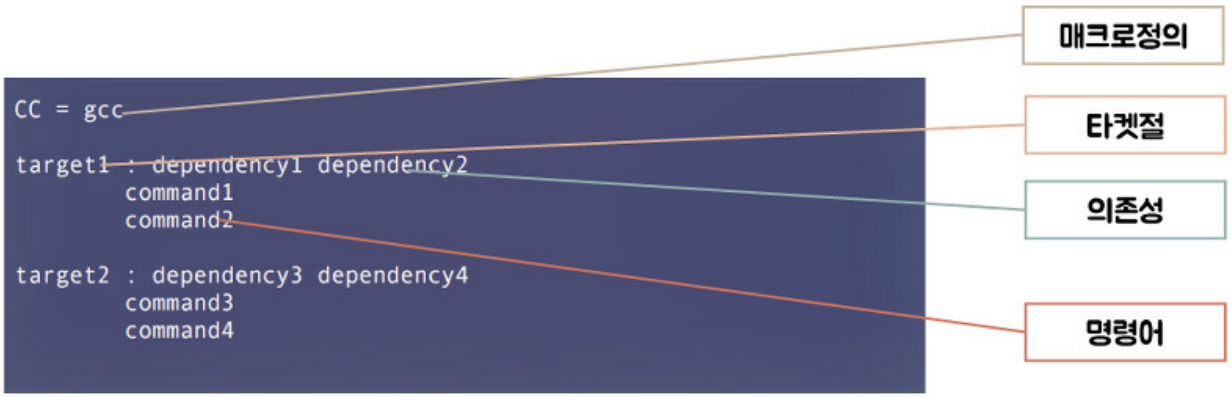
-의존성(Dependency) : 목적파일을 만들기 위해 필요한 재료

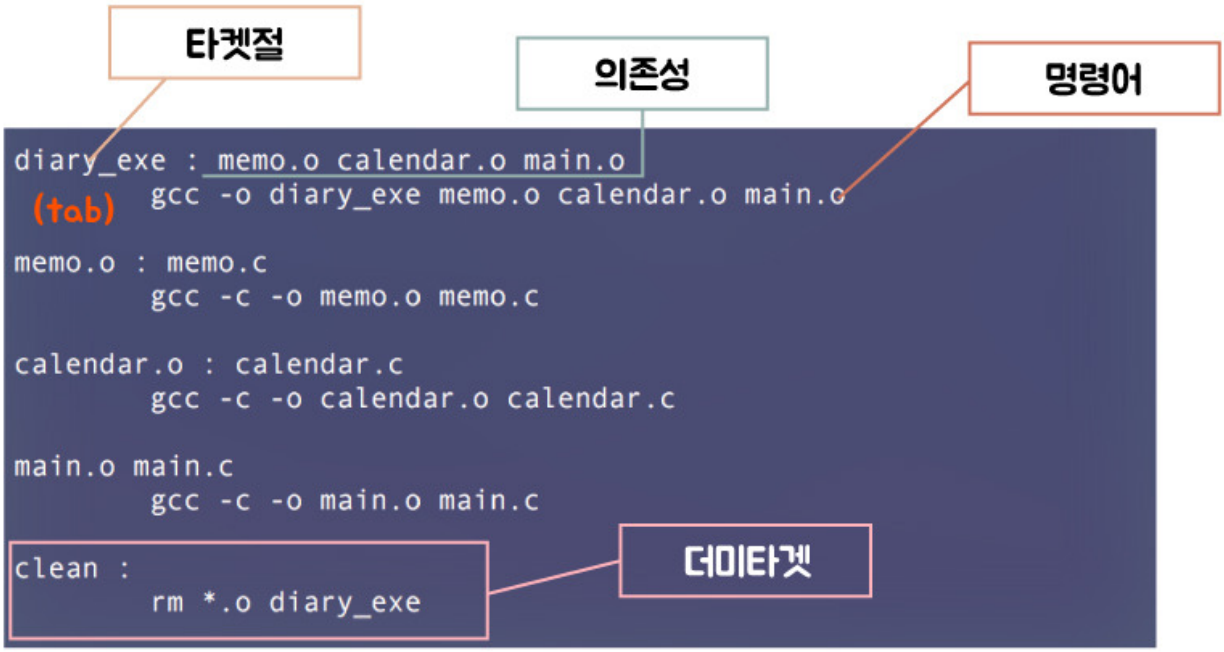
-명령어(Command) : 실행 되어야 할 명령어들

-매크로(macro) : 코드를 단순화 시키기 위한 방법

Makefile의 기본구조

위의 구성에서 말한 요소들은 실제 `Makefile` 코드에서 다음과 같이 배치됩니다.





더미 타겟에서 make clean 입력시 현재 디렉토리의 모든 **object 파일**들과 생성된 실행파일인 **diary_exe**를 rm 명령어로 제거해 줍니다.