

CaOS

(Calcium Operating System)

CaOS v0.05 manual

2007.9.5 v0.05 문서 작성

김기오
www.asmlove.co.kr

CaOS v0.05 manual

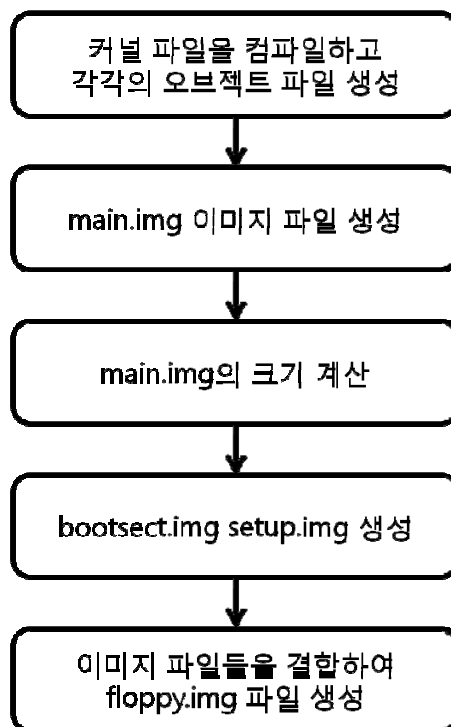
1. CaOS 소개

리눅스 커널을 공부하려고 몇번 시도해봤지만 둔한 머리로 책만 읽어봐서는 안된다는 결론만 얻었습니다. 그래서 2004년 1월 Calcium Operating System 이라는 거창한 이름으로 프로젝트를 시작했습니다. 리눅스 커널을 공부하고 핵심되는 기능을 어떻게든 간단하게라도 손으로 만들어봐서 잘 이해해보자는 취지로, 제 지식의 튼튼한 뼈대를 세울 수 있는 칼슘 같은 프로젝트가 되라고 해서 칼슘이라는 이름을 붙였습니다.

IA-32 기반 펜티엄 이상 프로세서에서 동작하도록 만들고 있고 VMware v5.5 이상에서 테스트해보았습니다. 싱글 프로세서에서 동작하고 4MB 이상의 물리 메모리를 필요로 합니다. 파일 시스템을 만들지 않았으므로 하드디스크는 필요하지 않고 플로피 드라이브만 필요합니다.

2. Makefile

커널 이미지가 만들어지는 과정과 전체적인 동작 방식을 알기 위해서는 Makefile을 이해하는 것이 필요합니다. 대략적인 컴파일 과정은 다음 그림과 같습니다.



다음은 Makefile 파일의 내용입니다.

최적화 옵션 -O는 사용하지 않는다. 최적화 옵션을 사용하면 커널이 부팅되지 않는다.

INCLUDE=.

CFLAGS=-Wall -I\$(INCLUDE)

커널에 포함되지는 않지만 부트로더와 커널 이전의 프로세서 설정에 사용되는 파일

BOOT_SRCS = bootsect.asm setup.asm

커널을 이루는 파일들

C_SRCS = main.c kprintf.c io.c init_idt.c init_irq.c except_handler.c keyboard.c string.c timer.c
sched.c process.c oops.c user_tasks.c page.c memory.c page_alloc.c init.c char_device.c syscall.c
user_syscall.c

ASM_SRCS = head.asm exception.asm irq.asm syscall_handler.asm

BOOT_IMG = boot.img

C_OBJS = \$(C_SRCS:.c=.o)

ASM_OBJS = \$(ASM_SRCS:.asm=.o)

순수 커널로만 이루어진 이미지

MAIN_IMG = main.img

all:\$(MAIN_IMG) \$(BOOT_IMG)

생성된 모든 이미지 파일을 결합해서 vmware에서 사용할 플로피 디스크의 이미지를 만든다. 부트로더 이미지 bootsect.img가 처음에 결합되어야 시스템이 부팅되면서 부트로더가 실행될 수 있다. 부트로더를 이미지의 끝 부분에도 결합하는 이유는 커널의 크기를 섹터 크기 512에 맞춰주기 위해서이다. 만약 커널 이미지 main.img가 4106바이트이면 9개의 섹터를 읽어야 커널을 모두 읽게 된다. 그런데 int 13h로 이미지를 읽으면 이미지의 크기가 섹터크기로 나뉘지지 않기 때문에 8개의 섹터까지만 읽게 되고 나머지 데이터들은 읽을 수가 없게 된다. 따라서 512바이트 크기의 부트로더를 커널 이미지 끝에 연결해주어서 512바이트의 섹터 크기 단위로 나누어질 수 있도록 한다.

```
cat bootsect.img setup.img main.img bootsect.img > floppy.img
```

```
nm main.tmp > text.txt
```

```
$(MAIN_IMG):$(ASM_OBJS) $(C_OBJS) $(USER_OBJS)
```

커널 코드가 메모리에 저장되는 가상 메모리를 0xC010 0000으로 설정했다. 이것은 페이징 설정에서 정하기 나름이다. 커널을 구성하는 어셈블리 파일과 C 파일을 링크해서 하나의 오브젝트 파일로 만든다.

```
ld -o main.tmp -Ttext 0xc0100000 $(ASM_OBJS) $(C_OBJS)
```

어셈블리 소스들과 C 소스들이 합쳐서 만들어진 main.tmp 파일에서 각종 정보들을 제외한 코드와 데이터만을 추출하면 순수 바이너리 이미지 main.img 파일이 만들어진다.

```
objcopy -j .text -j .rodata -j .data -O binary main.tmp $@
```

C 파일을 컴파일한다.

```
%.o : %.c
```

```
gcc $(CFLAGS) -c $<
```

어셈블리 파일을 컴파일 할 때는 반드시 ELF 오브젝트 파일로 어셈블해야 한다. 그래서 C 소스의 오브젝트 파일과 링크할 수 있다.

```
%.o : %.asm
```

```
nasm -f elf -o $@ $<
```

ls -l 명령으로 main.img 파일의 파일 크기를 출력해서 몇개의 섹터인지 계산한다. 계산된 섹터 수를 bootsect.asm과 setup.asm의 상수로 전달한다.

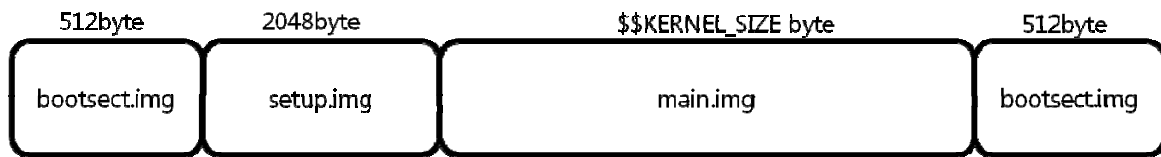
```
$(BOOT_IMG):$(BOOT_SRCS)
```

```
@KERNEL_SIZE=`ls -l main.img | awk '{ print $5 }';` \W
KERNEL_SIZE=`expr \W( $$KERNEL_SIZE + 512 \W) / 512`; \W
echo "Kernel size(sectors) = " $$KERNEL_SIZE; \W
nasm -DKERNEL_SIZE=$$KERNEL_SIZE -o bootsect.img bootsect.asm
@KERNEL_SIZE=`ls -l main.img | awk '{ print $5 }';` \W
KERNEL_SIZE=`expr \W( $$KERNEL_SIZE + 512 \W) / 512`; \W
echo "Kernel size(sectors) = " $$KERNEL_SIZE; \W
nasm -DKSIZE=$$KERNEL_SIZE -o setup.img setup.asm
```

clean:

```
rm -f *.o *.img *.tmp
```

3. floppy.img



4. 부팅과정

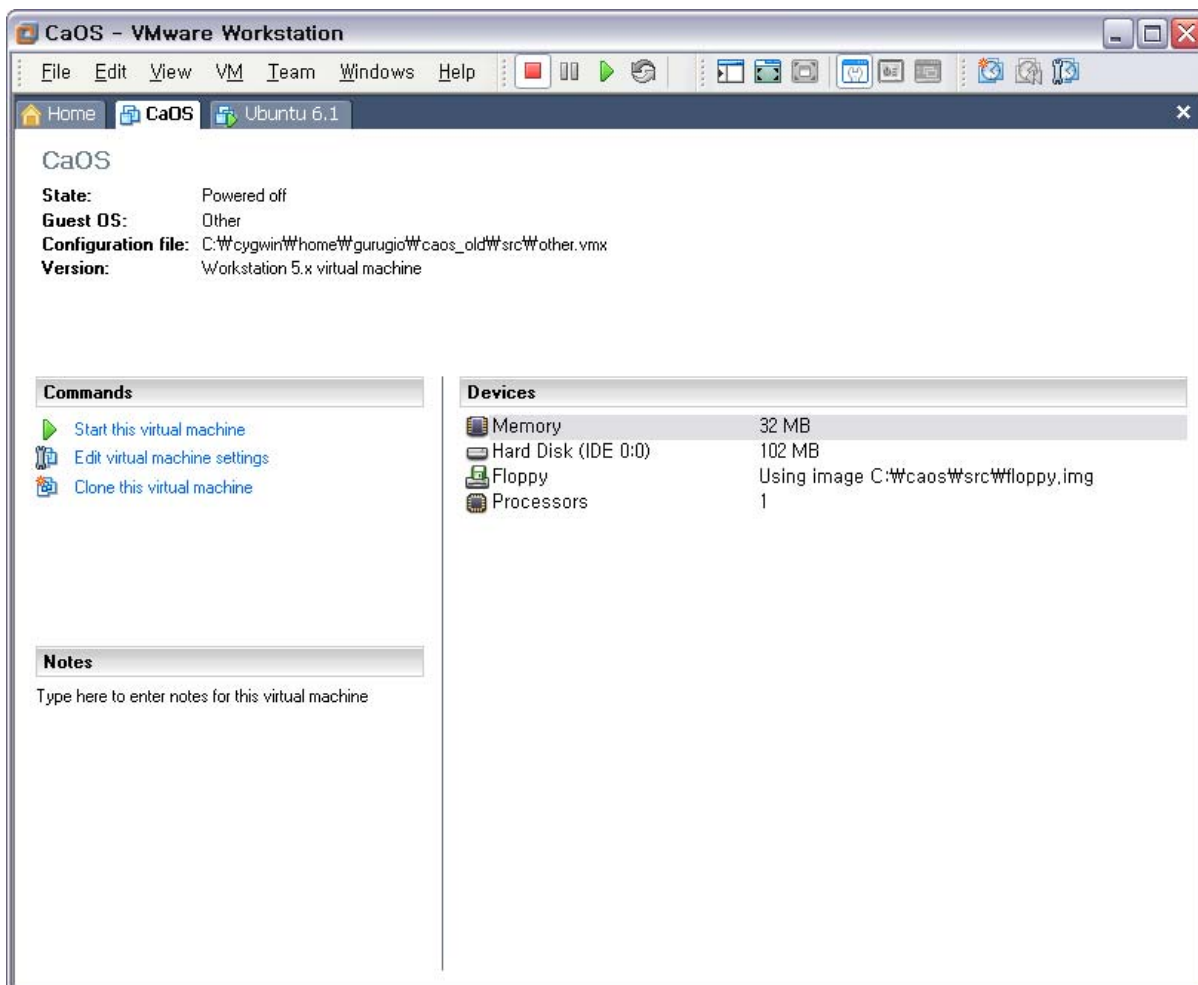
- ① bootsect.asm: setup.asm과 main.img을 메모리에 읽어오고 setup.asm으로 점프
- ② setup.asm
 - A. TSS 초기화
 - B. GDT 초기화
 - C. 보호모드 진입
 - D. 메모리 크기 확인
 - E. 커널을 물리메모리 0x100000 (1MB)로 복사
 - F. 임시 페이지 디렉토리 (swapper_pg_dir)와 페이지 테이블(pg0) 설정
 - G. 커널로 점프 (커널이 물리메모리 0x100000 에 있으므로 가상 메모리는 0xC0100000 이 된다)
- ③ head.asm: 커널이 임시로 사용할 스택 설정 후 start_kernel 함수 실행
- ④ main.c: 주변 장치 설정, 메모리 관리, 태스크 관리, 스케줄러 실행 등의 처리를 하고 유저 모드 init 태스크 실행

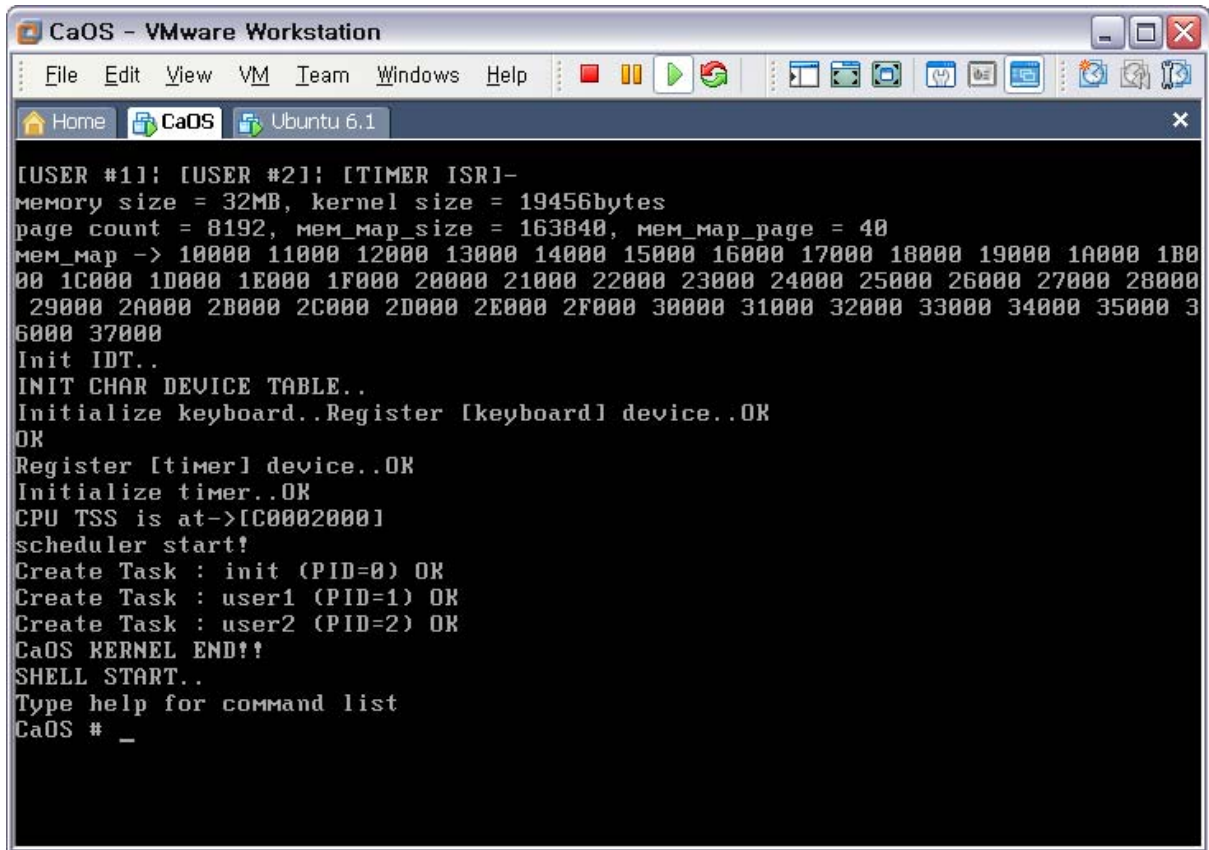
5. VMware 사용법

- 프로젝트 파일의 압축 해제
- src 디렉토리로 이동
- make clean;make 명령 실행
- floppy.img 파일이 생성됨

```
root@gurugio-desktop: /home/gurugio/caos/src
root@gurugio-desktop:/home/gurugio/caos/src# make clean;make
make: Warning: File `Makefile' has modification time 1.4e+04 s in the future
rm -f *.o *.img *.tmp
make: warning: Clock skew detected. Your build may be incomplete.
make: Warning: File `Makefile' has modification time 1.4e+04 s in the future
nasm -f elf -o head.o head.asm
nasm -f elf -o exception.o exception.asm
nasm -f elf -o irq.o irq.asm
nasm -f elf -o syscall_handler.o syscall_handler.asm
gcc -Wall -I. -c main.c
gcc -Wall -I. -c kprintf.c
gcc -Wall -I. -c io.c
gcc -Wall -I. -c init_idt.c
gcc -Wall -I. -c init_irq.c
gcc -Wall -I. -c except_handler.c
gcc -Wall -I. -c keyboard.c
gcc -Wall -I. -c string.c
gcc -Wall -I. -c timer.c
gcc -Wall -I. -c sched.c
gcc -Wall -I. -c process.c
gcc -Wall -I. -c oops.c
gcc -Wall -I. -c user_tasks.c
gcc -Wall -I. -c page.c
gcc -Wall -I. -c memory.c
gcc -Wall -I. -c page_alloc.c
gcc -Wall -I. -c init.c
gcc -Wall -I. -c char_device.c
gcc -Wall -I. -c syscall.c
gcc -Wall -I. -c user_syscall.c
ld -o main.tmp -Ttext 0xc0100000 head.o exception.o irq.o syscall_handler.o main.o kprintf.o io.o
init_idt.o init_irq.o except_handler.o keyboard.o string.o timer.o sched.o process.o oops.o user
_tasks.o page.o memory.o page_alloc.o init.o char_device.o syscall.o user_syscall.o
objcopy -j .text -j .rodata -j .data -O binary main.tmp main.img
Kernel size(sectors) = 38
Kernel size(sectors) = 38
cat bootsect.img setup.img main.img bootsect.img > floppy.img
nm main.tmp > text.txt
make: warning: Clock skew detected. Your build may be incomplete.
root@gurugio-desktop:/home/gurugio/caos/src#
```

- floppy.img를 플로피 이미지로 지정해서 가상 머신 실행





```
CaOS - VMware Workstation
File Edit View VM Team Windows Help
Home CaOS Ubuntu 6.1
[USER #1]: [USER #2]: [TIMER ISR]-
memory size = 32MB, kernel size = 19456bytes
page count = 8192, mem_map_size = 163840, mem_map_page = 40
mem_map -> 10000 11000 12000 13000 14000 15000 16000 17000 18000 19000 1A000 1B000 1C000 1D000 1E000 1F000 20000 21000 22000 23000 24000 25000 26000 27000 28000 29000 2A000 2B000 2C000 2D000 2E000 2F000 30000 31000 32000 33000 34000 35000 36000 37000
Init IDT..
INIT CHAR DEVICE TABLE..
Initialize keyboard..Register [keyboard] device..OK
OK
Register [timer] device..OK
Initialize timer..OK
CPU TSS is at->[C0002000]
scheduler start!
Create Task : init (PID=0) OK
Create Task : user1 (PID=1) OK
Create Task : user2 (PID=2) OK
CaOS KERNEL END!!
SHELL START..
Type help for command list
CaOS # _
```

- h를 입력하면 도움말을 볼 수 있음
- 화면 상단에 2개의 유저 태스크와 타이머 인터럽트 서비스가 주기적으로 실행되고 있음을 알리는 메시지 출력됨

6. 파일 리스트

파일들을 기능에 따라 분류하면 다음과 같다. 각각의 기능을 분석할 때 해당 파일들을 확인한다. (작업 상황에 따라 약간 변동이 있을 수 있습니다.)

- 부트로더
 - bootsect.asm
- 프로세서 초기화:
 - except_handler.c
 - exception.asm
 - gdt.c
 - gdt.h
 - head.asm
 - idt.h
 - init_idt.c
 - init_irq.c

- irq.asm
- irq.h
- segment.h
- selector.inc
- setup.asm
- 태스크 관리
 - process.c
 - process.h
 - processor.h
 - sched.c
 - sched.h
 - user_syscall.c
 - user_syscall.h
 - user_tasks.c
 - user_tasks.h
 - init.c
 - init.h
 - syscall.c
 - syscall.h
 - syscall_handler.asm
- 메모리 관리
 - memory.c
 - memory.h
 - page.c
 - page.h
 - page_alloc.c
 - page_alloc.h
 - paging.c
 - pgtable.h
- 주변 장치
 - timer.c
 - timer.h
 - keyboard.c
 - keyboard.h
 - kprintf.c
 - console.h
- 기타 유틸리티
 - string.c
 - string.h

- a20.inc
- inline_asm.h
- io.c
- io.h
- list.h
- oops.c
- oops.h
- 커널 시작 지점
 - main.c