

# System Programming

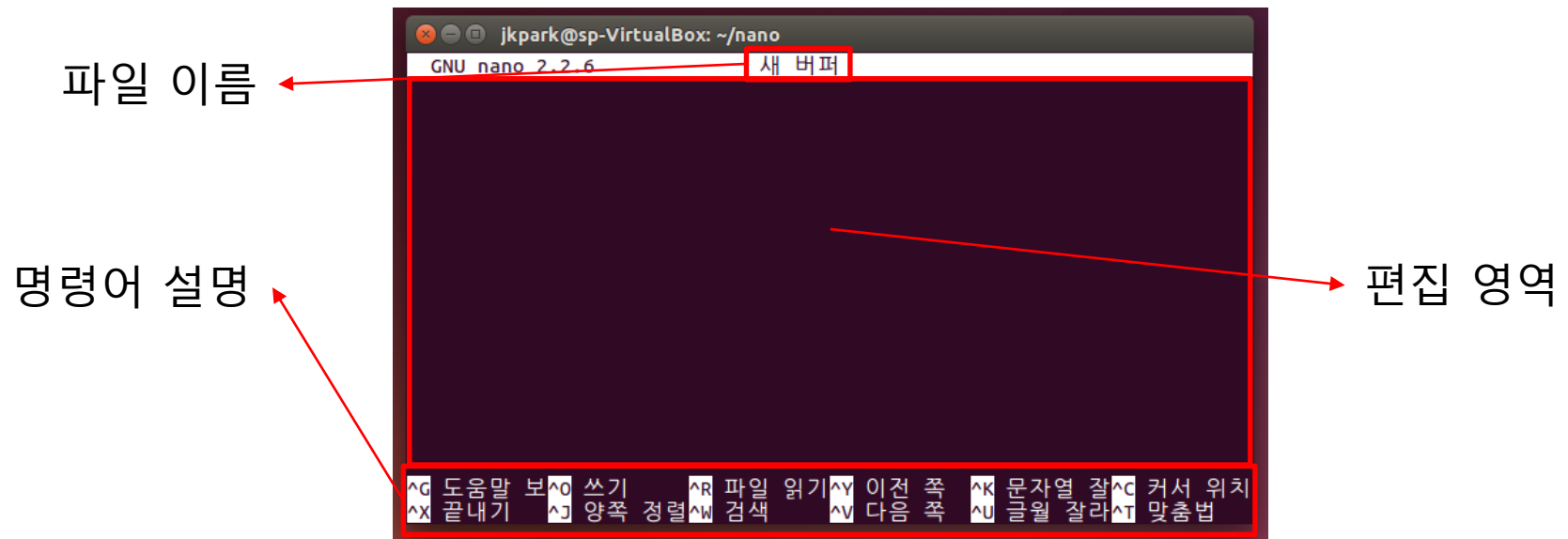
Hyun-Wook Jin  
System Software Laboratory  
Department of Computer Science & Engineering  
Konkuk University  
jinh@konkuk.ac.kr

# Nano

- 매우 기본적인 기능만 제공하는 텍스트 에디터
- 사용이 쉽고 간편함
- Vi 에디터와 달리 플러그인 설치 불가능
- 가볍고 빠름

# Nano

- 터미널에서 nano 를 입력하여 실행
  - 예) \$ nano : 파일을 지정하지 않고, nano 실행
  - 예) \$ nano main.c : 읽거나 쓰기 위한 파일을 지정하며 nano 실행



# Nano

- 단축키

- Ctrl 키는 ^, Alt 키는 M- 기호로 표현
- 기본적인 단축키
  - Ctrl + O (^O) : 파일 저장
  - Ctrl + R (^R) : 현재 파일에 다른 파일 읽어서 삽입
  - Ctrl + X (^X) : 프로그램 종료
  - Ctrl + Y (^Y) : 이전 쪽
  - Ctrl + V (^V) : 다음 쪽
  - Ctrl + W (^W) : 검색
  - Alt + } (M-}) : 들여쓰기
  - Alt + { (M-{) : 내어쓰기
- 복사 및 붙여넣기
  - Alt + ^ (M-^) : 현재 행 복사
  - Ctrl + K (^K) : 행 잘라내기
  - Ctrl + U (^U) : 붙여넣기
  - Ctrl + 6 (^6) : 블록 지정(텍스트 범위 선택)
  - Alt + 6 (M-6) : 지정한 블록을 복사

# Nano

- 단축키

- 검색

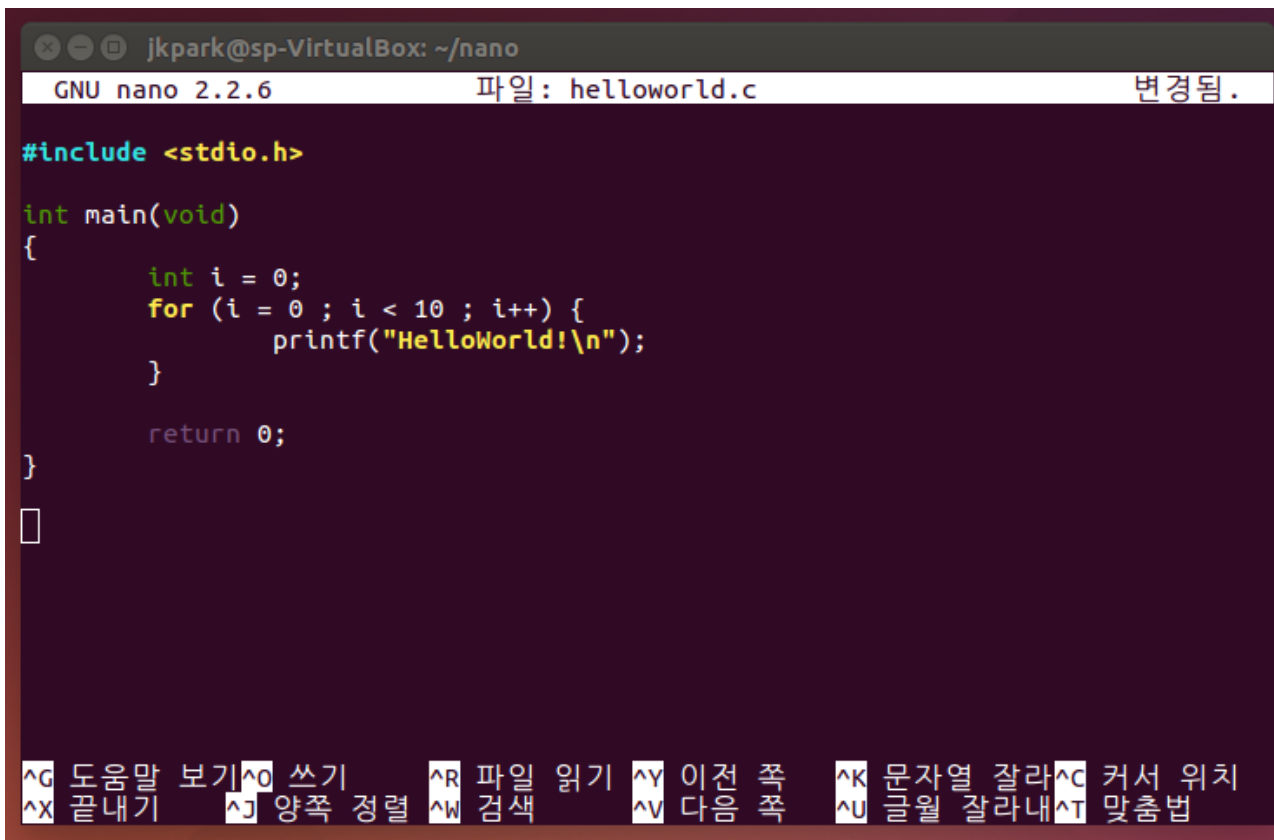
- Ctrl + W (^W) : 내용 검색
    - Alt + W (M-W) : 이전 검색 반복

- 이동

- Ctrl + A (^A) : 현재 행의 처음으로 이동
    - Ctrl + E (^E) : 현재 행의 마지막으로 이동
    - Alt + ] (M-]) : 쌍이 되는 괄호로 이동
    - Alt + - (M--) : 커서 이동 없이, 화면을 위로 이동
    - Alt + + (M-+) : 커서 이동 없이, 화면을 아래로 이동

# Challenge #1

- Nano 에디터를 사용하여 아래와 같이 파일 작성하고, 단축키 사용해보기



```
jkpark@sp-VirtualBox: ~/nano
GNU nano 2.2.6                파일: helloworld.c                변경됨.

#include <stdio.h>

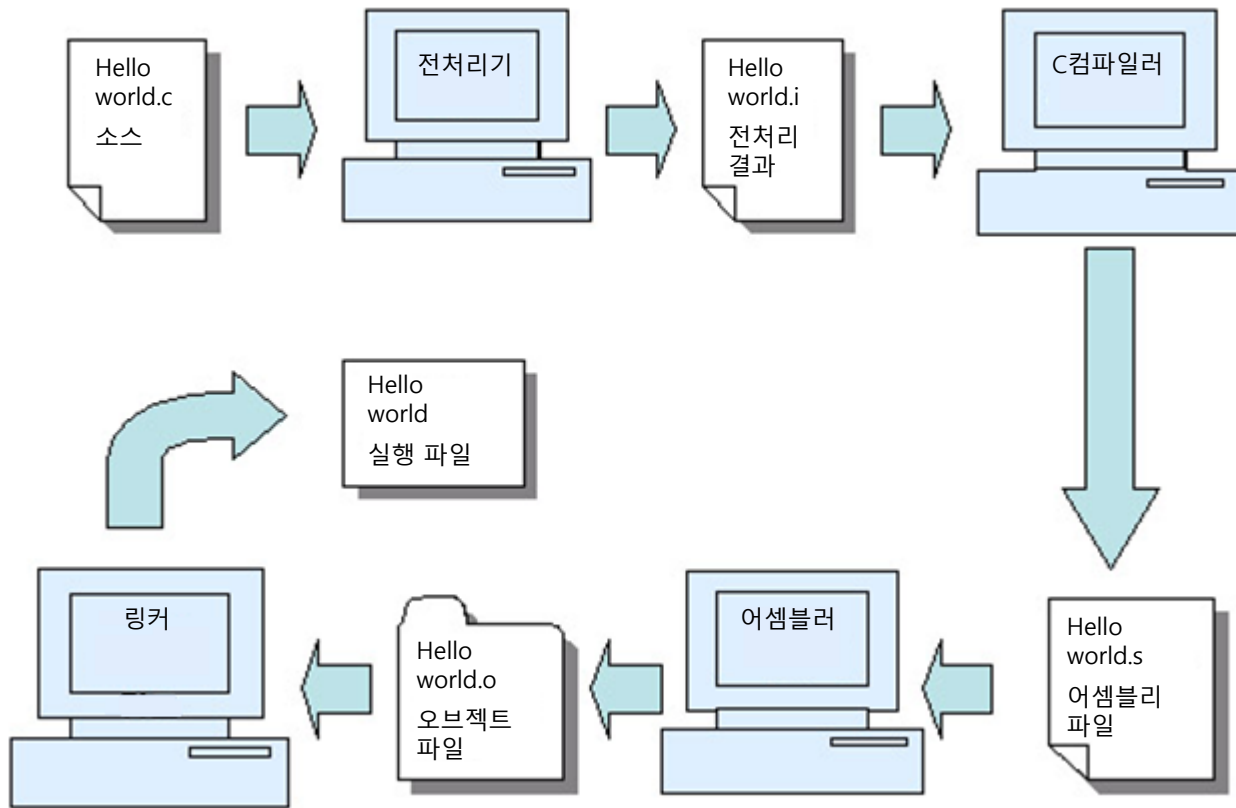
int main(void)
{
    int i = 0;
    for (i = 0 ; i < 10 ; i++) {
        printf("HelloWorld!\n");
    }

    return 0;
}
□
```

^G 도움말 보기 ^O 쓰기 ^R 파일 읽기 ^Y 이전 쪽 ^K 문자열 잘라내기 ^C 커서 위치  
^X 끝내기 ^J 양쪽 정렬 ^W 검색 ^V 다음 쪽 ^U 글줄 잘라내기 ^T 맞춤법



# 컴파일 과정



# Assembly 언어란?

- 컴퓨터 언어의 3단계

- 기계어

00000000h: 10 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00 00  
00000010h: 00 00 00 00 00 00 00 00 3E 00 03 00 FE FF 09 00

- 어셈블리어

- 고급언어

```
push    %ebp
mov     %esp, %ebp
push    %ebx
sub     $0x4, %esp
call    8048468 <_fini+0xc>
```

- 어셈블리어

- 인간이 이해할 수 있는 문자열

- 기계어와 1:1 대응

```
int main(int argc, char * argv[])
{
    int num, exp, res;
    num = atoi(argv[1]);
    exp = atoi(argv[2]);
}
```



# GCC

- GNU 컴파일러 모음(GNU Compiler Collection, GCC)는 GNU 프로젝트의 일환으로 개발되어 널리 쓰이고 있는 컴파일러
- GCC는 C만을 지원했던 컴파일러로 이름도 "GNU C Compiler"였지만, 후에 C++, java, fortran, ada 등 여러 언어를 컴파일 할 수 있게 되면서, 현재의 이름으로 바뀜
- `$ gcc -v` : gcc 버전 정보를 알려줌
- C언어 컴파일 방법
  - `$ gcc [파일이름] ex) $ gcc helloworld.c`
    - a.out 파일이 생성됨

# GCC 옵션

- **-o : output option**
  - 출력(output) file명을 정하는 option
    - 예) \$ gcc -o helloworld helloworld.c
    - 예) \$ gcc helloworld.c -o helloworld
- **-D : macro option**
  - gcc의 command line에서 macro를 define 할 수 있도록 하는 옵션
    - 예) \$ gcc -o helloworld -DMAXLEN=255 helloworld.c
  - -DMAXLEN=255 는 #define MAXLEN 255와 같음

# GCC 옵션

- -E

- gcc의 컴파일 과정 중에서 C preprocessing까지만 처리하고 나머지 단계는 처리하지 말라는 것을 지시
  - 예) \$ gcc -E -o helloworld.i helloworld.c

- -S

- 전처리된 파일을 어셈블리 파일로 컴파일까지만 수행
  - ex) \$ gcc -S helloworld.c
- helloworld.s 생성

# GCC 옵션

- -C
  - 컴파일(compile) 작업만 할 경우
  - 오브젝트 파일 생성 ex) helloworld.o
    - ex) `$gcc -c helloworld.c`
- -I
  - `#include` 문장에서 지정한 헤더 파일이 들어있는 곳을 정하는 option
  - `$gcc -c file이름.c -I<디렉토리>`
    - ex) `$gcc -c helloworld.c -linclude`

# GCC 옵션

- -g
  - 디버깅 모드로 컴파일
  - 오류가 발생했을 경우, 소스코드 단위로 Tracing 할 수 있도록 추가적인 데이터를 삽입
  - gdb 프로그램으로 디버깅할 때 보다 수월하게 디버깅 가능
    - ex) \$ gcc -g helloworld.c



# 실행 파일 실행

- ./실행파일명 ex) ./helloworld

```
umja@umja-desktop: ~  
파일(F) 편집(E) 보기(V) 터미널(T) 도움말(H)  
umja@umja-desktop: ~$ ls  
dasn2          hel loworl d      sysprog  문서      음악  
data1ab        hel loworl d. c   sysprog2  바탕화면  템플릿  
data1ab_fullset  install - tl - 20150912  공개     비디오  
examples.desktop  install - tl - unx.tar.gz  다운로드 사진  
umja@umja-desktop: ~$ ./hel loworl d  
hel lo worl d  
umja@umja-desktop: ~$
```

# Challenge #2

- Challenge #1 에서 작성한 소스코드를 gcc를 사용하여 컴파일하고 실행해보자

```

jkpark@sp-VirtualBox: ~/nano
GNU nano 2.2.6          파일: helloworld.c          변경됨.

#include <stdio.h>

int main(void)
{
    int i = 0;
    for (i = 0 ; i < 10 ; i++) {
        printf("HelloWorld!\n");
    }

    return 0;
}

```

[illegible]



# Challenge #3

- Challenge #1 에서 작성한 소스코드를 가지고,  
gcc -S 를 사용하여 어셈블리어로 변환해보자  
gcc -c 를 사용하여 오브젝트 파일을 만들어보자

```
jkpark@jkpark-VirtualBox: ~/nano
jkpark@jkpark-VirtualBox:~/nano$ ls -al
합계 20
drwxrwxr-x  2 jkpark jkpark 4096  9월  28 02:33 .
drwxr-xr-x 20 jkpark jkpark 4096  9월  28 02:26 ..
-rw-rw-r--  1 jkpark jkpark  125  9월  28 02:27 helloworld.c
-rw-rw-r--  1 jkpark jkpark 1528  9월  28 02:33 helloworld.o
-rw-rw-r--  1 jkpark jkpark  567  9월  28 02:27 helloworld.s
jkpark@jkpark-VirtualBox:~/nano$
```

```
GNU nano 2.2.6          파일: helloworld.s

.file "helloworld.c"
.section .rodata
.LC0:
.string "HelloWorld!"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl $0, -4(%rbp)
movl $0, -4(%rbp)
jmp .L2

[ 35 행을 읽었습니다. ]
^G 도움말 보기 ^O 쓰기 ^R 파일 읽기 ^Y 이전 쪽 ^K 문자열 잘라내기 ^C 커서 위치
^X 끝내기 ^J 양쪽 정렬 ^W 검색 ^V 다음 쪽 ^U 글줄 잘라내기 ^T 맞춤법
```



# Objdump

- GNU 바이너리 유틸리티의 일부
- 바이너리 파일들의 정보를 보여주는 프로그램
- 역어셈블러로 사용 가능



# Objdump

- Option

옵션   긴 옵션	설명
-d   --disassemble	오브젝트 파일을 기계어로 역어셈블
-D   --disassemble-all	모든 섹션을 대상으로 역어셈블
--[no-]show-raw-insn	코드와 바이트열 제거/출력
--prefix-address	코드의 주소를 심볼에서의 상대주소로 표시
-j section   --section=section	특정 섹션 지정
-l   --line-numbers	각각의 코드에 대응하는 소스코드의 행에 관한 정보 출력
-S   --source	행 번호에 해당하는 소스코드가 그 위치에 삽입되어 출력

# Example 1

- Objdump

- gcc -g -c ex4.c
- objdump -d -S ex4.o

```

1 #include <stdio.h>
2
3 int main(){
4
5     unsigned int ret ;
6
7     __asm__ __volatile__ ("rdtsc" : "=A"(ret));
8
9     printf("clock time : %d \n",ret);
10
11     return 0;
12 }

```

ex4.c

```

sp@sp: ~/inline$ objdump -d -S ex4.o
ex4.o:          file format elf32-i386


Disassembly of section .text:

00000000 <main>:
#include <stdio.h>

int main(){
0: 55                push    %ebp
1: 89 e5             mov     %esp, %ebp
3: 83 e4 f0          and     $0xffffffff0, %esp
6: 83 ec 20          sub     $0x20, %esp

    unsigned int ret ;

    __asm__ __volatile__ ("rdtsc" : "=A"(ret));
9: 0f 31             rdtsc
b: 89 44 24 1c       mov     %eax, 0x1c(%esp)

    printf("clock time : %d \n",ret);
f: b8 00 00 00 00    mov     $0x0, %eax
14: 8b 54 24 1c       mov     0x1c(%esp), %edx
18: 89 54 24 04       mov     %edx, 0x4(%esp)
1c: 89 04 24          mov     %eax, (%esp)
1f: e8 fc ff ff ff   call    20 <main+0x20>

    return 0;
24: b8 00 00 00 00    mov     $0x0, %eax

29: c9               leave
2a: c3               ret

```

# Example 2

## • Objdump

- gcc -g -c ex5.c
- objdump -d -S ex5.o

```

1 #include <stdio.h>
2
3 void main(void){
4
5     int x = 4;
6     int y = 6;
7     int ret = 0;
8     ret = max(x, y);
9 }
10
11 int max(int x, int y)
12 {
13     if(x > y)
14         return x;
15     else
16         return y;
17 }

```

ex5.c

```

0000003b <max>:
int max(int x, int y)
{
    if(x > y)
3b: 55                push    %ebp
3c: 89 e5             mov     %esp, %ebp
    return x;
3e: 8b 45 08          mov     0x8( %ebp), %eax
41: 3b 45 0c          cmp     0xc( %ebp), %eax
44: 7e 05             jle     4b <max+0x10>
    else
46: 8b 45 08          mov     0x8( %ebp), %eax
49: eb 03             jmp     4e <max+0x13>
    return y;
4b: 8b 45 0c          mov     0xc( %ebp), %eax
4e: 5d                pop     %ebp
4f: c3                ret

```

# Challenge #4

- Challenge #1의 코드를 gcc -S 로 변환된 어셈블리어와 objdump 가지고 오브젝트 파일을 통해서 얻어낸 어셈블리어를 비교해보기
  - gcc의 -g 옵션을 가지고 만들어진 오브젝트 파일을 objdump 사용하여 어셈블리어 얻어보기

```

jkpark@jkpark-VirtualBox: ~/nano
GNU nano 2.2.6          파일: helloworld.s

.file "helloworld.c"
.section .rodata
.LC0:
.string "HelloWorld!"
.text
.globl main
.type main, @function

main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl $0, -4(%rbp)
movl $0, -4(%rbp)
jmp .L2
    
```

35 행을 읽었습니다.

도움말 보기, 쓰기, 파일 읽기, 이전 쪽, 문자열 잘라내기, 커서 위치, 끝내기, 양쪽 정렬, 검색, 다음 쪽, 글꼴 잘라내기, 맞춤법

```

jkpark@jkpark-VirtualBox: ~/nano
helloworld.c helloworld.o helloworld.s
jkpark@jkpark-VirtualBox:~/nano$ objdump -d -S helloworld.o

helloworld.o:          file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
0: 55                      push    %rbp
1: 48 89 e5                mov     %rsp,%rbp
4: 48 83 ec 10             sub     $0x10,%rsp
8: c7 45 fc 00 00 00 00    movl    $0x0,-0x4(%rbp)
f: c7 45 fc 00 00 00 00    movl    $0x0,-0x4(%rbp)
16: eb 0e                  jmp     26 <main+0x26>
18: bf 00 00 00 00         mov     $0x0,%edi
1d: e8 00 00 00 00         callq   22 <main+0x22>
22: 83 45 fc 01            addl    $0x1,-0x4(%rbp)
26: 83 7d fc 09            cmpl    $0x9,-0x4(%rbp)
2a: 7e ec                  jle     18 <main+0x18>
2c: b8 00 00 00 00         mov     $0x0,%eax
31: c9                      leaveq   %eax
32: c3                      retq

jkpark@jkpark-VirtualBox:~/nano$
    
```