설치 sudo apt install gcc
make

1 유닉스 시스템 프로그래밍 개요

한성대학교 컴퓨터공학부 김진환

학습목표

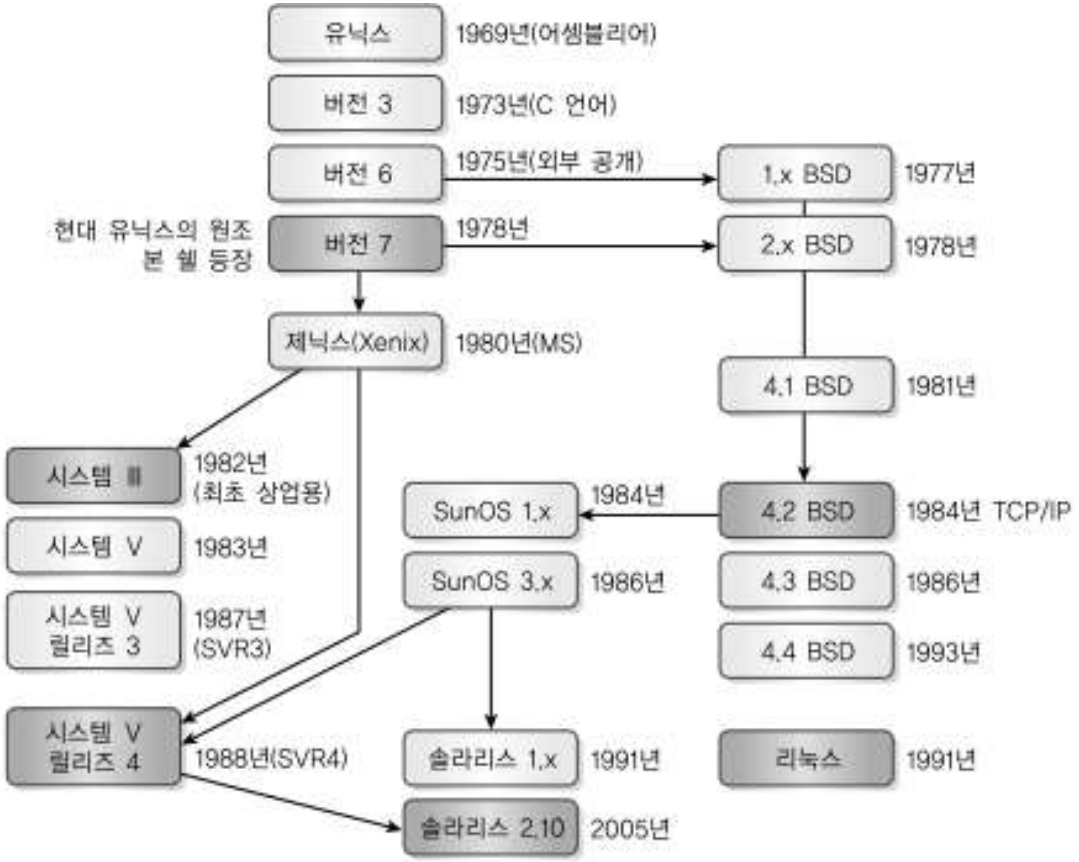
- 유닉스 시스템 관련 표준을 이해한다.
- 유닉스 시스템 프로그래밍이 무엇인지 이해한다.
- 시스템 호출과 라이브러리 함수의 차이를 이해한다.
- 유닉스 시스템의 기본 명령을 사용할 수 있다.
- C 컴파일러와 make 도구를 사용할 수 있다.

목차

- ✓ 유닉스 시스템의 역사
- ✓ 유닉스 시스템 표준
- ✓ 유닉스 시스템 프로그래밍이란
- ✓ 시스템 호출과 라이브러리 함수의 비교
- ✓ 유닉스 기본 명령
- ✓ 컴파일 환경과 Makefile
- ✓ 오류처리함수
- ✓ 동적 메모리 할당 함수
- ✓ 명령행 인자

유닉스 시스템의 역사

- 1969 AT&T산하의 벨연구소에서 켄 톰슨과 데니스 리치가 개발
- 1973 C언어를 이용하여 재개발 -> 고급 언어로 작성한 최초의 운영체제
- 그 후 상용유닉스(시스템V) 계열과 BSD 계열로 분리하여 각각 발전
- 1989 AT&T와 썬마이크로시스템즈가
  두 계열의 장점을 결합하여
  SVR4를 공동개발
  - 이 유닉스가 현재 사용하는
    대부분의 유닉스의 기반임



유닉스 시스템 표준

ANSI C 표준

- 미국 표준협회(ANSI)에서 표준화한 C 언어 명세 : ANSX3.159-1989
- ISO가 이를 받아들여 ISO/IEC 9899:1990으로 발표함(www.iso.org)

POSIX

- 서로 다른 유닉스 시스템 사이에서 상호 이식이 가능한 응용프로그램을 개발하기 위한 표준으로 IEEE에서 제정
- POSIX.1(IEEE Std 1003.1) : C언어 응용 프로그래밍 인터페이스 표준
- POSIX.2(IEEE Std 1003.2) : 표준 쉘과 유틸리티 프로그램 인터페이스 표준

X/Open 가이드

- X/Open은 유럽의 유닉스 제조업체를 중심으로 설립한 단체로 개방형 시스템에 대한 표준을 정의하고 보급하고 있음
- X/Open 이식성 가이드 : XPG3, XPG4
- 1996년 오픈소프트웨어재단과 합병하여 오픈그룹(The Open Group)으로 새 출발함
  - 오픈 그룹이 UNIX에 대한 상표권 소유

시스템V 인터페이스 정의

- SVID : 프로그램과 장치에서 이용할 수 있는 시스템 호출과 C라이브러리 표준 포함

유닉스 시스템 프로그래밍이란

단일 유닉스 규격(SUS)

- 오스틴 그룹이 관리, IEEE와 오픈 그룹의 작업에 기반하여 SUSv3 발표

유닉스시스템 프로그래밍의 정의

- 유닉스에서 제공하는 시스템 호출을 사용해 프로그램을 작성하는 것을 의미

시스템 호출

- 유닉스 시스템이 제공하는 서비스를 이용해 프로그램을 작성할 수 있도록 제공되는 프로그래밍 인터페이스
- 기본적인 형태는 C 언어의 함수 형태로 제공

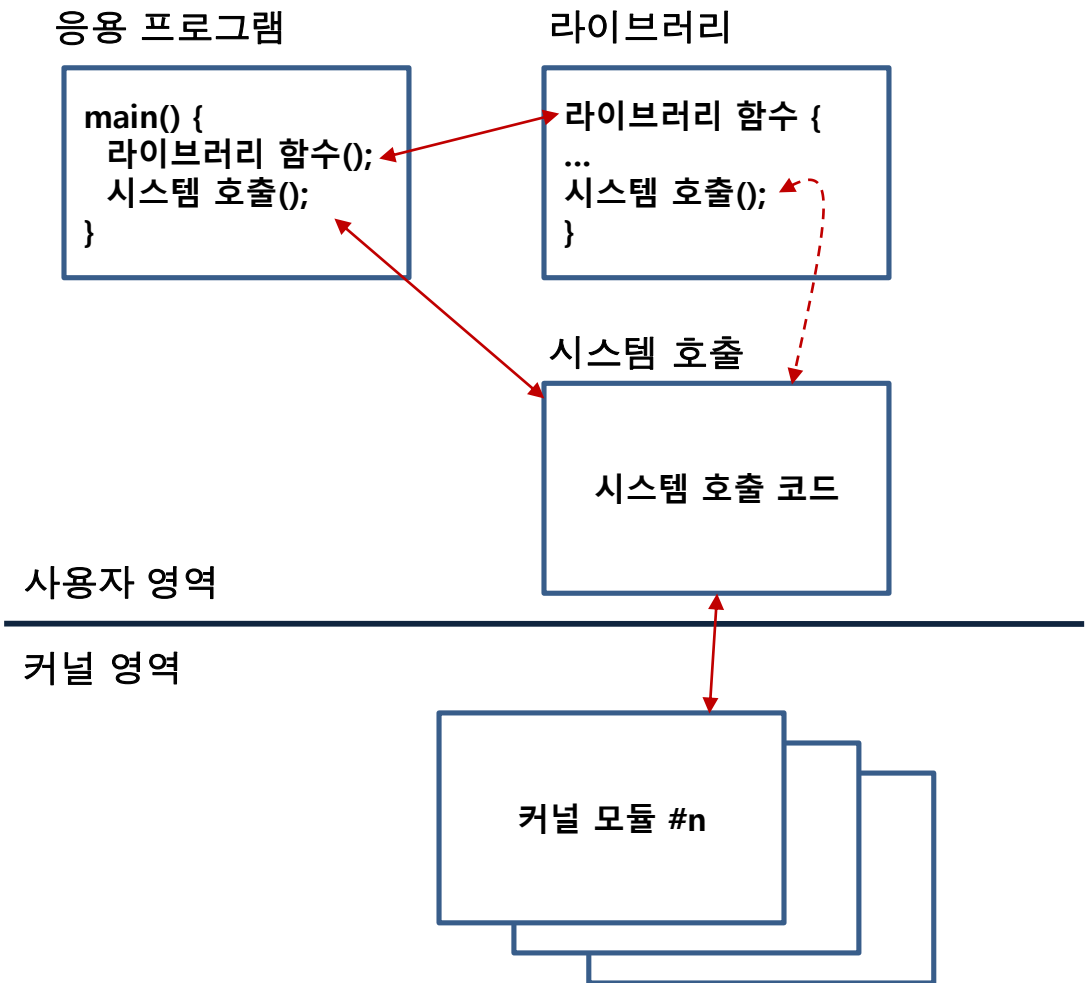
리턴값 = 시스템호출명(인자, ...);

라이브러리 함수

- 라이브러리 : 미리 컴파일된 함수들을 묶어서 제공하는 특수한 형태의 파일
- 자주 사용하는 기능을 독립적으로 분리하여 구현해둠으로써 프로그램의 개발과 디버깅을 쉽게하고 컴파일을 좀 더 빠르게 할 수 있다
- /lib, /usr/lib에 위치하며 lib*.a 또는 lib*.so 형태로 제공

시스템 호출과 라이브러리 함수의 비교[1]

- 시스템 호출 : 커널의 해당 서비스 모듈을 직접 호출하여 작업하고 결과를 리턴
- 라이브러리 함수 : 일반적으로 커널 모듈을 직접 호출안함



시스템 호출과 라이브러리 함수의 비교[2]

- 시스템 호출 : man 페이지가 섹션 2에 속함

```
System Calls                                                    open(2)

NAME
        open, openat - open a file
SYNOPSIS
        #include <sys/types.h>
```

- 라이브러리 함수 : man 페이지가 섹션 3에 속함

```
Standard C Library Functions                                    fopen(3C)

NAME
        fopen - open a stream
SYNOPSIS
        #include <stdio.h>
```

시스템 호출과 라이브러리 함수의 비교[3]

시스템 호출의 오류 처리방법

- 성공하면 0을 리턴, 실패하면 -1을 리턴
- 전역변수 errno에 오류 코드 저장 : man 페이지에서 코드값 확인 가능

[예제 1-1] 시스템 호출 오류 처리하기 ex1_1.c

```
01  #include <unistd.h>     //</usr/include/errno.h> 추가 필요
02  #include <stdio.h>
03
04  extern int errno;
05
06  int main(void) {
07      if (access("unix.txt", F_OK) == -1) {
08          printf("errno=%d\n", errno);
09      }
10
11      return 0;
12  }
```

```
# ex1_1.out
errno=2
```

```
# vi /usr/include/sys/errno.h
......
/*
* Error codes
*/
#define EPERM   1       /* Not super-user */
#define ENOENT  2       /* No such file or directory */
......
```

시스템 호출과 라이브러리 함수의 비교[4]

라이브러리 함수의 오류 처리방법

- 오류가 발생하면 NULL을 리턴, 함수의 리턴값이 int 형이면 -1 리턴
- errno 변수에 오류 코드 저장

[예제 1-2] 라이브러리 함수 오류 처리하기 ex1_2.c

#include <sys/errno.h> ← 이게 있어야 에러 안나고 오류 코드 저장

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  extern int errno;
05
06  int main(void) {
07      FILE *fp;
08
09      if ((fp = fopen("unix.txt", "r")) == NULL) {
10          printf("errno=%d\n", errno);
11          exit(1);
12      }
13      fclose(fp);
14
15      return 0;
16  }
```

읽기모드로 파일을 연다

```
# ex1_2.out
errno=2
```

man fopen에서 확인

유닉스 기본 명령[1]

로그인/로그아웃

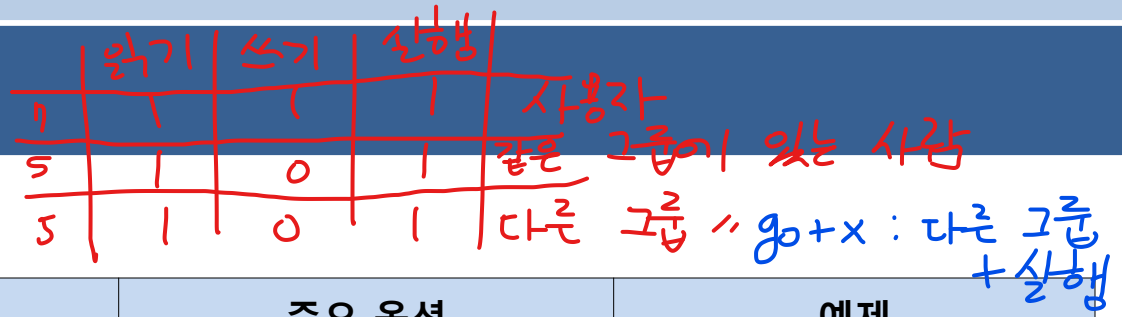
명령	기능	주요 옵션	예제
telnet	유닉스시스템에 접속	–	telnet hanb.co.kr
logout	유닉스시스템에서 접속해제	–	logout
exit		–	exit

프로세스 관련 명령

명령	기능	주요 옵션	예제
ps	현재 실행 중인 프로세스의 정보를 출력	-ef : 모든 프로세스에 대한 상세 정보 출력	ps ps -ef ps -ef | grep ftp
kill	프로세스 강제 종료	-9 : 강제 종료	kill 5000 kill -9 5001

| : 왼쪽의 결과값을 오른쪽에 대입할 수 있다

유닉스 기본 명령[2]



파일/디렉토리 조작 명령

명령	기능	주요 옵션	예제
pwd	현재 디렉토리 경로 출력	–	pwd
ls	디렉토리 내용 출력	-a : 숨김파일출력 -l : 파일 상세정보 출력	ls –a /tmp ls –l
cd	현재 디렉토리 변경	–	cd /tmp cd ~han01
cp	파일/디렉토리 복사	-r : 디렉토리 복사	cp a.txt b.txt cp –r dir1 dir2
mv	파일/디렉토리 이름변경과 이동	–	mv a.txt b.txt A → B 로 mv a.txt dir1 mv dir1 dir2 옮기고 A 삭제
rm	파일/디렉토리 삭제	-r : 디렉토리 삭제	rm a.txt rm –r dir1
mkdir	디렉토리 생성	–	mkdir dir1
rmdir	빈 디렉토리 삭제	–	mkdir dir2
cat	파일 내용 출력	–	cat a.txt
more	파일 내용을 쪽단위로 출력	–	more a.txt
chmod	파일 접근권한 변경	–	chmod 755 a.exe chmod go+x a.exe
grep	패턴 검색 a.txt 에서 abcd 가 어디있는가		grep abcd a.txt

유닉스 기본 명령[3]

vi 편집기 내부 명령

기능	명령	기능	명령
입력모드전환	i,a,o,O	명령모드전환	<Esc>
커서이동	j,k,h,l 또는 방향키	행이동	#G (50G, 143G 등) 또는 :행번호
한글자수정	r	여러글자수정	#s (5s, 7s 등)
단어수정	cw	명령취소undo	u, U
검색하여수정	:%s/aaa/bbb/g	복사	#yy (5yy, 10yy 등)
붙이기	p	커서이후삭제	D(shidt-d)
글자삭제	x, #x(3x,5x 등)	행삭제 (잘라내기)	dd, #dd(3dd, 4dd 등)
저장하고종료	:wq 또는 ZZ	저장않고종료	:q!
행 붙이기	J(shift-j)	화면다시표시	ctrl+l
행번호보이기	:set nu	행번호없애기	:set nonu

유닉스 기본 명령[4]

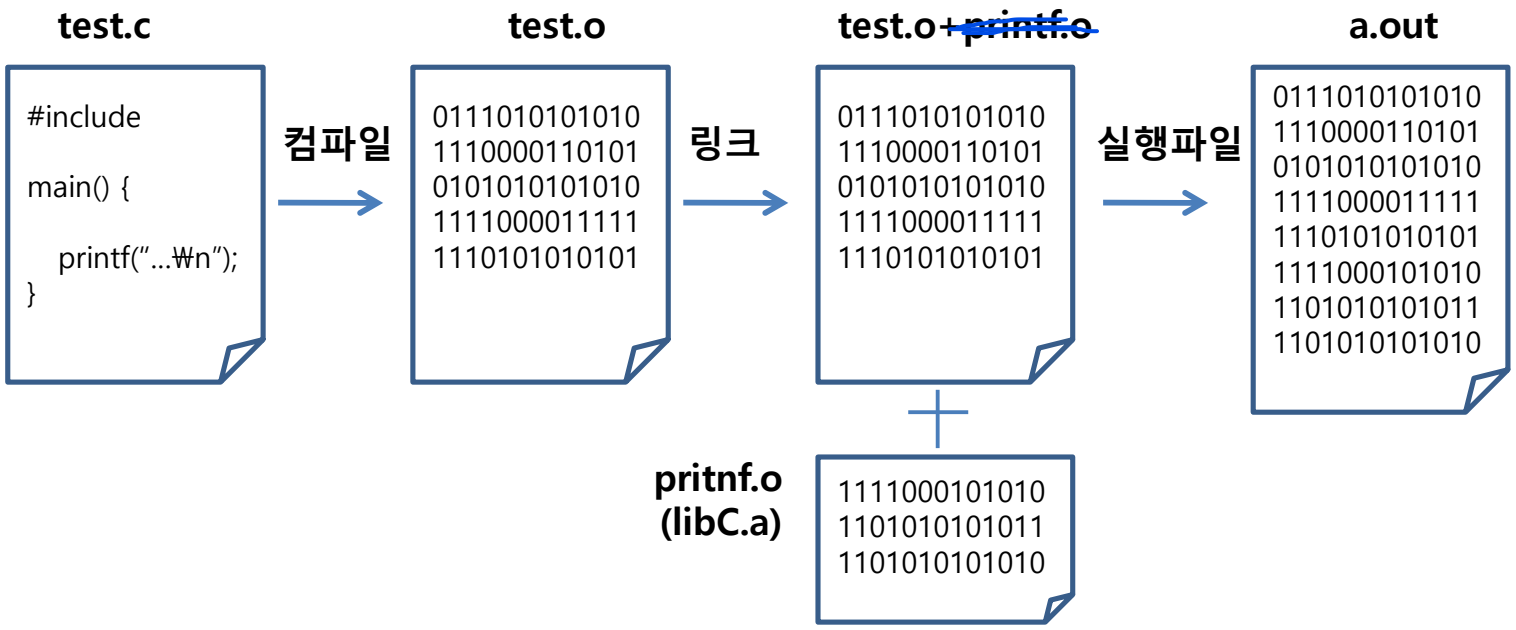
기타 명령

명령	기능	주요 옵션	예제
su	사용자 계정 변경	- : 변경할 사용자의 환경 초기화 파일 실행	su su - su - han02
tar	파일/디렉토리 묶기	-cvf : tar파일생성 -tvf : tar파일내용보기 -xvf : tar파일풀기	tar cvf a.tar * tar tvf a.tar tar xvf a.tar
whereis	파일 위치 검색	–	whereis ls
which		–	which telnet

컴파일 환경[1]

컴파일이란

- 텍스트로 작성한 프로그램을 시스템이 이해할 수 있는 기계어로 변환하는 과정
- 보통 컴파일 과정과 라이브러리 링크 과정을 묶어서 수행하는 것을 의미



컴파일 환경[2]

GNU C 컴파일러 : gcc

- 대부분 GNU C 컴파일러 사용(www.sunfreeware.com)
- /usr/local/bin 디렉토리에 설치됨 -> 경로에 추가해야 함

```
# vi ~/.profile
......
PATH=$PATH:/usr/local/bin
export PATH
```

```
# . ~/.profile
```

바뀐 .profile 적용

- C컴파일러 사용

```
# gcc test.c
# ls
a.out test.c
```

기본 실행파일명은 a.out

```
# gcc -o test test.c
# ls
test   test.c
```

실행파일명 지정은 -o 옵션

컴파일 환경[3]

Makefile과 make

- 소스 파일이 여러 개를 묶어서 실행파일을 생성하는 도구
- make 명령은 Makefile의 내용에 따라 컴파일, /usr/ccs/bin을 경로에 추가해야함

```
# vi ~/.profile
......
PATH=$PATH:/usr/local/bin:/usr/ccs/bin
export PATH
```

[예제 1-3] ex1_3_main.c

```
01  #include <stdio.h>
02  extern int addnum(int a, int b);
03
04  int main(void) {
05      int sum;
06
07      sum = addnum(1, 5);
08      printf("Sum 1~5 = %d\n", sum);
09
10      return 0;
11  }
```

[예제 1-3] ex1_3_addnum.c

```
01  int addnum(int a, int b) {
02      int sum = 0;
03
04      for (; a <= b; a++)
05          sum += a;
06      return sum;
07  }
```

컴파일 환경[3]

[예제 1-3] make 명령 사용하기 Makefile

```
01  # Makefile
02
03  CC=gcc
04  CFLAGS=
05  OBJS=ex1_3_main.o ex1_3_addnum.o
06  LIBS=
07  all:    add
08
09  add:    $(OBJS)
10    $(CC) $(CFLAGS) -o add $(OBJS) $(LIBS)
11
12  ex1_3_main.o:          ex1_3_main.c
  $(CC) $(CFLAGS) -c ex1_3_main.c
14  ex1_3_addnum.o:        ex1_3_addnum.c
15    $(CC) $(CFLAGS) -c ex1_3_addnum.c
16
17  clean:
18    rm -f $(OBJS) add core
```

주석

* 반드시 들여쓰기 할 것 (Tab)

컴파일

라이브러리

add 만들어진 실행파일의 이름

ex1_3_main.c와 ex1_3_addnum.c를 묶어서 add라는 실행파일 생성

실행을 시키고 난 후, 불필요한 코드를 지워라

```
# make
gcc   -c ex1_3_main.c
gcc   -c ex1_3_addnum.c
gcc   -o add ex1_3_main.o
          ex1_3_addnum.o
# ls
Makefile    add*  ex1_3_addnum.c
ex1_3_addnum.o    ex1_3_main.c
ex1_3_main.o
# add
Sum 1~5 = 15
```

떼고 명령 바람

-o : 지정 -c : 컴파일 (링크X)

오류 처리 함수[1]

오류 메시지 출력 : perror(3)

```
#include <stdio.h>
void perror(const char *s);
```

[예제 1-4] perror 함수 사용하기 ex1_4.c

```
01  #include <sys/errno.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      if (access("unix.txt", R_OK) == -1) {
08          perror("unix.txt");
09          exit(1);
10      }
11
12      return 0;
13  }
```

```
# ex1_4.out
unix.txt: No such file or directory
```

번호가 아님

오류 처리 함수[2]

오류 메시지 출력 : strerror(3)

```
#include <string.h>
char *strerror(int errnum);
```

[예제 1-5] strerror 함수 사용하기 ex1_5.c

```
01  #include <sys/errno.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05  #include <string.h>
06
07  extern int errno;
08
09  int main(void) {
10      char *err;
11
12      if (access("unix.txt", R_OK) == -1) {
13          err = strerror(errno);
14          printf("오류:%s(unix.txt)\n", err);
15          exit(1);
16      }
17
18      return 0;
19  }
```

오류에 따라 메시지를 리턴

```
# ex1_5.out
오류: No such file or directory(unix.txt)
```

동적 메모리 할당[1]

메모리할당 : malloc(3)

```
#include <stdlib.h>
void *malloc(size_t size);
```

- 인자로 지정한 크기의 메모리 할당

```
char *ptr
ptr = malloc(sizeof(char) * 100);
```

메모리할당과 초기화 : calloc(3)

```
#include <stdlib.h>
void *calloc(size_t nelem, size_t elsize);
```

- nelem * elsize 만큼의 메모리를 할당하고, 0으로 초기화

```
char *ptr
ptr = calloc(10, 20);
```

동적 메모리 할당[2]

메모리 추가 할당: realloc(3)

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

- 이미 할당받은 메모리(ptr)에 size 크기의 메모리를 추가로 할당

```
char *ptr, *new;
ptr = malloc(sizeof(char) * 100);
new = realloc(ptr, 100);
```

메모리 해제 : free(3)

```
#include <stdlib.h>
void free(void *ptr);
```

- 사용을 마친 메모리 반납

명령행 인자[1]

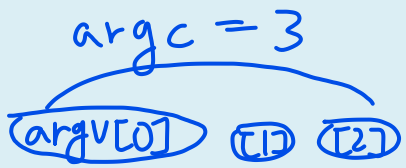
명령행 : 사용자가 명령을 입력하는 행

- 명령행 인자 : 명령을 입력할 때 함께 지정한 인자(옵션, 옵션인자, 명령인자 등)
- 명령행 인자의 전달 : main 함수로 자동 전달

```
int main(int argc, char *argv[])
```

[예제 1-6] 명령행 인자 출력하기 ex1_6.c

```
01  #include <stdio.h>
02
03  int main(int argc, char *argv[]) {
04      int n;
05
06      printf("argc = %d\n", argc);
07      for (n = 0; n < argc; n++)
08          printf("argv[%d] = %s\n", n, argv[n]);
09
10      return 0;
11  }
```




```
# ex1_6.out -h 100
argc = 3
argv[0] = ex1_6.out
argv[1] = -h
argv[2] = 100
```

옵션 -a 와 -b를 동시에 쓰는 법: -ab

명령행 인자[2]

옵션 처리함수: getopt(3)

```
#include <stdio.h>
int getopt(int argc, char * const argv[], const  char  *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

- argc, argv : main 함수에서 받은 것을 그대로 지정
- optstring : 사용할 수 있는 옵션 문자, 옵션에 인자가 있을 경우 문자 뒤에 ':'추가
- optarg : 옵션의 인자 저장
- optind : 다음에 처리할 argv의 주소
- optopt : 오류를 발생시킨 문자
- opterr : 오류 메시지를 출력하지 않으려면 0으로 지정

optind: 옵션하나 처리할 때 마다 1증가

ex) 초기값: 1, -a 처리 후: 2
-b 처리 후: 3
(./a.out -a -b)

유닉스 명령의 옵션 규칙

- 옵션의 이름은 한글자여야 하며, 모든 옵션의 앞에는 하이픈(-)이 있어야 한다.
- 인자가 없는 옵션은 하나의 – 다음에 묶여서 올 수 있다(-xvf)
- 옵션의 첫 번째 인자는 공백이나 탭으로 띄고 입력한다.
- 인자가 있어야 하는 옵션에서 인자를 생략할 수 없다.
- 명령행에서 모든 옵션은 명령의 인자보다 앞에 와야한다.
- 옵션의 끝을 나타내기 위해 --를 사용할 수 있다.

명령행 인자[3]

옵션 처리

abC 에서 c 만 콜론 붙어있다 = c는 argument 필요

[예제 1-7] getopt 함수 사용하기 ex1_7.c

```
01  #include <stdio.h>
02
03  int main(int argc, char *argv[]) {
04      int n;
05      extern char *optarg;
06      extern int optind;
07
08      printf("Current Optind : %d\n", optind);
09      while ((n = getopt(argc, argv, "abc:")) != -1) {
10          switch (n) {
11              case 'a':
12                  printf("Option : a\n");
13                  break;
14              case 'b':
15                  printf("Option : b\n");
16                  break;
17              case 'c':
18                  printf("Option : c, Argument=%s\n",
                                optarg);
19                  break;
20          }
21          printf("Next Optind : %d\n", optind);
22      }
23
24      return 0;
25  }
```

```
# ex1_7.out
Current Optind : 1
# ex1_7.out -a
Current Optind : 1
Option : a
Next Optind : 2
# ex1_7.out -c
Current Optind : 1
ex1_7.out:   option   requires   an
argument -- c
Next Optind : 2
# ex1_7.out -c name
Current Optind : 1
Option : c, Argument=name
Next Optind : 3
# ex1_7.out -x
Current Optind : 1
ex1_7.out: illegal option -- x
Next Optind : 2
```

요약

프로그래밍 표준

- 유닉스 시스템 프로그래밍과 관련 표준으로는 ANSI C, IEEE의 POSIX, X/Open그룹의 XPG3, XPG4, SVID, SUS가 있다.

유닉스 시스템 프로그래밍이란

- 유닉스 시스템이 제공하는 다양한 서비스를 이용하여 프로그램을 구현할 수 있도록 제공되는 프로그래밍 인터페이스를 시스템 호출이라고 하며, 이러한 시스템 호출을 사용하여 프로그램을 작성하는 것을 유닉스 시스템 프로그래밍이라고 한다.

시스템 호출과 라이브러리 함수

- 시스템 호출은 기본적인 형식은 C언어의 함수 형태로 제공된다. 시스템 호출은 직접 커널의 해당 모듈을 호출하여 작업을 하고 결과를 리턴한다. 라이브러리 함수들은 커널 모듈을 직접 호출하지 않는다. 라이브러리 함수가 커널의 서비스를 이용해야 할 경우에는 시스템 호출을 사용한다.
- man 명령을 사용할 때 시스템 호출은 섹션 2에 있고, 라이브러리 함수들은 섹션 3에 배치된다. 따라서 'man -s 2 시스템호출명'과 같이 사용한다.
- 시스템 호출은 오류 발생시 -1을 리턴하고 라이브러리 함수는 NULL 값을 리턴한다.

유닉스 시스템 프로그래밍 도구

- 기본 명령 : pwd, ls, cd, mkdir, cp, mv, rm, ps, kill, tar, vi
- 오류처리함수 : perror, strerror
- 동적메모리 할당 함수 : malloc, calloc, realloc, free
- 명령행인자 처리 함수 : getopt
  - 컴파일 : GNC C 컴파일러(gcc)
  - 컴파일 도구 : make명령과 Makefile