리눅스시스템 및 응용

Week 14

Environment Programming

학습목표

- Shell Parameter Transfer
- 시간 관련 함수
- 시스템 정보 함수
- 사용자 정보 검색 함수
- 임시파일 만들기
- Error Logging

Shell Parameter Transfer

- C언어로 작성된 프로그램은 main()부터 시작
- 외부의 인수 전달은 main()함수의 전달 인수로 입력
  - **`main(int argc, char *argv[])`**
- ex: **`% mypro left right “and center”`**
  - **`argc=4`**
  - **`argv={“mypro”, “left”, “right”, “and center”}`**
- Parameter
  - option(-)
  - argument

Ex

```
#include <stdio.h>

int main(int argc, char *argv[]){
        int arg;
        for(arg = 0; arg < argc; arg++) {
           if(argv[arg][0] == '-’)
           printf("option: %s\n", argv[arg]+1);
        else
           printf("argument %d: %s\n", arg, argv[arg]);
        }
        return(0);
}
```

getopt()

- main()의 파라메터 parsing을 하는 함수

```
#include <unistd.h>
extern char *optarg;
extern int optind, opterr, optopt;
```

  - int getopt(int argc,char const *argv[],const char *optstring);
  - getopt(): 더 이상 처리할 파라메터가 없을 때 –1 return
  - optstring: option을 표현하는 문자의 목록. ":"으로 option에 따르는 value가 있음을 표시
  - argument: option에 따르는 value, 외부변수 optarg의 의해 전달
  - 특별한 option "--"는 따르는 value가 없음
  - optopt: 지정된 option이외 option형식의 입력이 있을 때 "?"를 반환하고 optopt에 그 value를 전달
  - option이 따르는 value를 요구함에도 value가 없을 때 getopt()는 ":"를 전달
  - 외부변수 optind는 다음 처리할 argv의 index

```
#include <stdio.h>
#include <unistd.h>
main(int argc, char *argv[]){
          int opt;
          while((opt = getopt(argc, argv, "if:lr")) != -1) {
            switch(opt) {
               case 'i': /* option */
               case 'l': /* option */
               case 'r’:
                    printf("option: %c\n", opt); break;
               case 'f': /* option with argument */
                    printf("filename: %s\n", optarg); break;
               case ':': /* can’t get argument */
                    printf("option needs a value\n"); break;
               case '?’:
                    printf("unknown option: %c\n", optopt); break;
            }
          }
          for(; optind < argc; optind++)
             printf("argument: %s\n", argv[optind]);
}
```

- % gcc –o argopt argopt.c
- % ./argopt –i –lr ‘hi ther’ –f fred.c –q
  - option: i
  - option: l
  - option: r
  - filename: fred.c
  - argopt: illegal invalid option –q
  - unknown option: q
  - argument: hi there

- *optstring if:lr에서 f: 는 –f 다음에 argument value가 있다는 것을 표시

시간 관련 함수

- Unix/Linux 시스템은 1970년1월1일 GMT 0시 0분 0초를 기준 시간으로지정(=Epoch time)
  - MS-DOS는 1980년의 기준, 나머지는 동일

- 모든 시간은 기준 시간에서 초단위로 계산
  - 32bits 시스템에서는 2038년에 원점으로 환원
  - 이를 Y2K38 problem, 해결은 그 이전에 32bits 보다 크게 확장
  - Linux에서는 long time_t 변수 사용
  - 관련 Header file: /usr/include/time.h

- time()
  - 단위 시간정보 가져 오기

```
#include <time.h>                                    [함수 원형]

time_t time(time_t *tloc);
```

  - tloc : 검색한 시간 정보를 저장할 주소

  - time() 함수의 특징
    - time() 함수는 1970년 1월 1일 0시 0분 0초(UTC)부터 현재까지 경과된 시간을 초 단위로 알림
    - tloc가 널이 아니면 tloc가 가리키는 주소에 시간 정보를 저장하고, NULL이면 시간 정보를 리턴
    - time() 함수는 실패하면 -1을 리턴

```c
#include <time.h>
#include <stdio.h>
#include <unistd.h>

int main(){
        int i;
        time_t the_time;
        for(i = 1; i <= 10; i++) {
           the_time = time((time_t *)0);
           printf("The time is %ld\n", the_time);
           sleep(2);
        }
        for(i = 1; i <= 10; i++) {
           time(&the_time);
           printf("The time is %ld\n", the_time);
           sleep(2);
        }
}
```

- gtime()
  - 저수준 시간값을 일반적인 구조체로 변환 기능

```
#include <time.h>
struct tm *gmtime(const time_t timeval);
struct tm {
        int tm_sec /*0-59까지의 초*/
        int tm_min /*0-59까지의 분*/
        int tm_hour /*0-23까지의 시*/
        int tm_mday /*1-31까지의 일*/
        int tm_mon /*0-11까지의 월,1월은 0*/
        int tm_year /*1900 이후의 년*/
        int tm_wday /*0-6까지의 요일,월요일은 0*/
        int tm_yday /*0-365까지의 날짜*/
        int tm_isdst /*일광절약 daylight saving time 적용여부*/
}
```

```
01 #include <time.h>
02 #include <stdio.h>
03
04 int main() {
05     struct tm *tm;
06     time_t timep;
07
08     time(&timep);
09     printf("Time(sec) : %d₩n", (int)timep);
10
11     tm = gmtime(&timep);
12     printf("GMTIME=Y:%d ", tm->tm_year);
13     printf("M:%d ", tm->tm_mon);
14     printf("D:%d ", tm->tm_mday);
15     printf("H:%d ", tm->tm_hour);
16     printf("M:%d ", tm->tm_min);
17     printf("S:%d₩n", tm->tm_sec);
18
19     tm = localtime(&timep);
20     printf("LOCALTIME=Y:%d ", tm->tm_year);
21     printf("M:%d ", tm->tm_mon);
22     printf("D:%d ", tm->tm_mday);
23     printf("H:%d ", tm->tm_hour);
24     printf("M:%d ", tm->tm_min);
25     printf("S:%d₩n", tm->tm_sec);
26 }
```

- **08행** time() 함수로 초 단위 시간을 구함
- **11~17행** 초 단위 시간을 gmtime() 함수로 분해해 출력
- **19~25행** 초 단위 시간을 localtime() 함수로 분해해 출력
- **실행 결과** 연도(Y)가 121이므로 1900+121=2021년임을 알 수 있음
  월(M)이 2이면 3월을 의미
  시간(H)을 보면 gmtime() 함수의 결과는 13,
  localtime() 함수의 결과는 22로 9시간
  차이가 나는 것을 알 수 있음

```
#include <time.h>
struct tm *localtime(const time_t *timeval);
char *asctime(const struct tm *timeptr);
char *ctime(const time_t *timeval);
```

- localtime()은 time_t형의 값인 GMT시간을 시스템에 설정된 지역시간 (한국은 KST)과 일광절약 시간제로 변환
- asctime()함수는 struct tm의 값을 ASCII string으로 값을 변환하여 전달. 길이는 26으로 고정
- ctime()은 local time으로 변환한 뒤 이를 asctime()을 동작시킨 것과 동일 즉

  `ctime(timeval)=asctime(localtime(timeval))`

```c
#include <time.h>
#include <stdio.h>
int main(){
        time_t timeval;
        (void)time(&timeval);
        printf("The date is: %s", ctime(&timeval));
        printf("The date is: %s", asctime(localtime(&timeval)));
}
```

시스템 정보 함수

- 시스템 정보 검색
  - 기본 환경과 관련된 구조체나 상수를 사용해 정보를 검색
  - 하드웨어와 운영체제의 종류 정보, 메모리 페이지의 크기나 최대 패스워드 길이와 같은 시스템 환경 설정 정보 등

- 운영체제 기본 정보 검색
  - uname 명령
    - 시스템에 설치된 운영체제의 이름과 버전, 호스트명, 하드웨어 종류 등을 검색하려면 uname 명령을 사용
    - uname 명령에 -a 옵션을 지정하면 현재 시스템에 설치되어 있는 운영체제 정보가 출력
    - Ex: **`uname, uname –a`**

- uname()

```
#include <sys/utsname.h>                                    [함수 원형]

int uname(struct utsname *buf);
```

- buf : utsname 구조체 주소

- uname() 함수의 특징
  - 운영체제 정보를 검색해 utsname 구조체에 저장
  - utsname 구조체는 sys/utsname.h 파일에 정의되어 있고 man –s 2 uname으로 확인할 수 있음

```
struct utsname {

    char sysname[];
    char nodename[];
    char release[];
    char version[];
    char machine[];
};
```

항목	설명
sysname	현재 운영체제의 이름을 저장한다.
nodename	네트워크를 통해 통신할 때 사용하는 시스템의 이름을 저장한다.
release	운영체제의 릴리즈 번호를 저장한다.
version	운영체제의 버전 번호를 저장한다.
machine	운영체제가 동작하는 하드웨어의 표준 이름(아키텍처)을 저장한다.

```
01 #include <sys/utsname.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main() {
06     struct utsname uts;
07
08     if (uname(&uts) == -1) {
09         perror("uname");
10         exit(1);
11     }
12
13     printf("OSname : %s₩n", uts.sysname);
14     printf("Nodename : %s₩n", uts.nodename);
15     printf("Release : %s₩n", uts.release);
16     printf("Version : %s₩n", uts.version);
17     printf("Machine : %s₩n", uts.machine);
18 }
```

- **06행** utsname 구조체 변수를 선언 .
- **08행** utsname 구조체 주소를 인자로 지정, uname() 함수가 수행에 성공하면 utsname구조체 변수인 uts에 시스템 정보가 저장
- **13~17행** 구조체에 저장된 운영체제 정보를 문자열로 출력한다. 실행 결과에서 출력 내용을 확인할 수 있음

- sysconf()
  - 시스템 자원 정보 검색

```
#include <unistd.h>                                    [함수 원형]

long sysconf(int name);
```

  - name : 검색할 정보를 나타내는 상수
  - sysconf() 함수의 특징
    - 검색하려는 시스템 정보를 나타내는 상수를 인자로 받고 현재 설정되어 있는 시스템 자원값 또는 옵션값을 리턴
    - 오류가 발생하면 –1을 리턴
    - sysconf() 함수의 인자로 지정할 수 있는 상수는 sys/unistd.h 파일에 정의

상수	설명
_SC_ARG_MAX	exec() 계열 함수에 사용하는 인자의 최대 크기
_SC_CHILD_MAX	한 UID에 허용되는 최대 프로세스 개수
_SC_HOST_NAME_MAX	호스트명의 최대 길이
_SC_LOGIN_NAME_MAX	로그인명의 최대 길이
_SC_CLK_TCK	초당 클록 틱 수
_SC_OPEN_MAX	프로세스당 열 수 있는 최대 파일 수
_SC_PAGESIZE	시스템 메모리의 페이지 크기
_SC_VERSION	시스템이 지원하는 POSIX.1의 버전

```c
01 #include <unistd.h>
02 #include <stdio.h>
03
04 int main() {
05     printf("Arg Max : %ld₩n", sysconf(_SC_ARG_MAX));
06     printf("Clock Tick : %ld₩n", sysconf(_SC_CLK_TCK));
07     printf("Max Open File : %ld₩n", sysconf(_SC_OPEN_MAX));
08     printf("Max Login Name Length : %ld₩n", sysconf(_SC_LOGIN_NAME_MAX));
09 }
```

실행

```
$ ch5_2.out
Arg Max : 2097152
Clock Tick : 100
Max Open File : 1024
Max Login Name Length : 256
```

- **05~08행** 검색하고 싶은 명령을 sysconf() 함수의 인자로 지정
- **실행 결과** 인자의 최대 길이는 2097152바이트, 클릭은 초당 100번, 열 수 있는 최대 파일 개수는 1024개, 로그인명은 최대 256바이트 임을 알 수 있음

사용자 정보 검색 함수

- getlogin()
  - 로그인 이름으로 검색

```
#include <unistd.h>                                    [함수 원형]

char *getlogin(void);
```

- getuid()/geteuid()
  - UID로 검색

```
#include <unistd.h>                                    [함수 원형]
#include <sys/types.h>

uid_t getuid(void);
uid_t geteuid(void);
```

  - 현재 프로세스의 실제 사용자 ID를, geteuid() 함수는 유효 사용자 ID를 리턴, 두 함수 모두 인자를 받지 않음
  - **실제 사용자 ID(RUID)** : 로그인할 때 사용한 로그인명에 대응하는 UID로, 프로그램을 실행하는 사용자
  - **유효 사용자 ID(EUID)** : 프로세스에 대한 접근 권한을 부여할 때 사용,처음 로그인할 때는 실제 사용자 ID와 유효 사용자

ID가 같지만, setuid가 설정된 프로그램을 실행하거나 다른 사용자 ID로 변경할 경우 유효 사용자 ID는 달라짐

```c
01 #include <sys/types.h>
02 #include <unistd.h>
03 #include <stdio.h>
04
05 int main() {
06     uid_t uid, euid;
07     char *name;
08
09     uid = getuid();
10     euid = geteuid();
11     name = getlogin();
12
13     printf("Login Name=%s, UID=%d, EUID=%d₩n", name, (int)uid, (int)euid);
14 }
```

실행

```
$ ch5_4.out
Login Name=jw, UID=1000, EUID=1000
```

- getpwuid()
  - uid로 사용자 정보를 /etc/passwd 로 부터 조회

```
#include <sys/types.h>                                    [함수 원형]
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
```

- getpwnam()
  - 로긴 이름으로 사용자 정보를 /etc/passwd 로 부터 조회

```
#include <sys/types.h>                                    [함수 원형]
#include <pwd.h>

struct passwd *getpwnam(const char *name);
```

```
struct passwd {
        char *pw_name /* user login name */
        uid_t pw_uid /* UID numnber */
        gid_t pw_gid /* GID number */
        char *pw_dir /* user home directory $HOME */
        char *pw_shell /* user's shell */
}
```

```
01 #include <sys/types.h>
02 #include <pwd.h>
03 #include <stdio.h>
04
05 int main() {
06     struct passwd *pw;
07
08     pw = getpwuid(getuid());
10     printf("UID : %d₩n", (int)pw->pw_uid);
11     printf("Login Name : %s₩n", pw->pw_name);
12
13     pw = getpwnam("root");
14     printf("UID : %d₩n", (int)pw->pw_uid);
15     printf("Home Directory : %s₩n", pw->pw_dir);
16 }
```

임시파일 만들기

- mkstemp()
  - 임시파일 이름 지정

```
#include <stdlib.h>                                    [함수 원형]
char *template;
int mkstemp(char *template);
/* Returns open file descriptor or -1 on error (may not set errno) */
```

- template
  - 임시파일 이름의 템플릿
  - Ex: /tmp/tempFile_XXXXXXX
  - 정상적으로 만들어 지면 예를 들면 /tmp/tempFile_iQ2XoGK 처럼 임시파일의 path정보가 됨
- return 값
  - file descriptor : 정상적인 수행시 template 만들고, 임시파일 생성해서 파일 디스크립터 반환
  - -1 : 실패

```
#include <stdlib.h>
#include <stdio.h>

int main(){
        char temppath[]="/tmp/tempFile_XXXXXX";
        int fd;

        fd=mkstemp(temppath);
        if( fd>0 ){
                printf("Temporary File Name: %s \n:, temppath);
                printf("File Desc: %d\n", fd);
        } else
                 printf("mkstemp Error\n");
}
```

```c
#include <stlib.h>
#include <stdio.h>

int main(){
        char tmpname[L_tmpnam];
        char *filename;
        FILE *tmpfp;
        filename = tmpnam(tmpname);
        printf("Temporary file name is: %s\n", filename);
        tmpfp = tmpfile();
        if(tmpfp)
                printf("Opened a temporary file OK\n");
        else
                perror("tmpfile");
        exit(0);
}
```

Error Logging

- Logging : 에러메시지, 동작과정의 기록
  - /var/log : system-wide logging
  - /var/log/dmesg
- /etc/syslog.conf
  - system-wide log에 대한 설정
- log보기 **:%dmesg , %last , %lastcom**
- User Error log on user's file
  - 사용자 프로그램에서 별도 log를 관리하는 것은 file을 생성하여 별도 관리
  - fopen(), fprintf() 사용
- system error logging 중요함수
  - syslog()
  - openlog()
  - closelog()
  - setlogmask()

```
#include <syslog.h>
void syslog(int priority,const char *message, argument . . . );
```

- syslog(): 로그 기능에 로그 메시지 전달
  - return value : 0(success), -1(fail)
- priority:log level, 메시지의 중요도
- LOG_USER: 사용자에 의한 생성
- system생성 LOG일경우 아래의 log level
  - LOG_EMERG : 비상사태, 모든 사용자에게 전달
  - LOG_ALERT : 높은 정도의 상태, 운영자에게 전달
  - *이하 로그파일에 기록
  - LOG_CRIT : hardware failure등의 치명적인 error,
  - LOG_ERR : 일반적인 error
  - LOG_WARNING : 경고
  - LOG_NOTICE : 주의
  - LOG_INFO : 단순 정보
  - LOG_DEBUG : 시스템 debug message

```
#include <syslog.h>
#include <stdio.h>
int main(){
        FILE *f;
        f = fopen("not_here","r"); /*존재하지 않는 파일-열기 당연 error*/
        if(!f)
                syslog(LOG_ERR|LOG_USER,"oops - %m\n");
}
```

```
#include <syslog.h>
void openlog(const char *ident, int logopt, int facility);
void closelog(void);
int selogmask(int maskpri);<syslog.h>
void syslog(int priority,const char *message, argument . . . );
```

- ident: log message에 연결되는 string
- facility: loggin의 기본 동작 값(default는 LOG_USER)
- logopt: loggin의 동작 환경
  - LOG_PID: 프로세스 번호를 logging
  - LOG_CONS:logging을 할 수 없을 경우 console로 출력
  - LOG_ODELAY:syslog() 처음 호출부터 logging
  - LOG_NDELAY:syslog()에 즉시 logging
- logmask(): log message의 우선순위를 위한 log mask 설정
  - LOG_MASK(priority) 혹은 LOG_UPTO(priority) macro 이용

```
#include <syslog.h>
#include <stdio.h>
#include <unistd.h>

int main(){
        int logmask;
        openlog("logmask", LOG_PID|LOG_CONS, LOG_USER);
        syslog(LOG_INFO,"informative message, pid = %d", getpid());
        syslog(LOG_DEBUG,"debug message, should appear");
        logmask = setlogmask(LOG_UPTO(LOG_NOTICE));
        syslog(LOG_DEBUG,"debug message, should not appear");
}
```

Question?