Linux System Programming #3-3
Lecture Notes

Spring 2020

School of Computer Science and Engineering,

Soongsil University, Seoul, Korea

Jiman Hong

jiman@acm.org

mkdir(2), rmdir(2)

```
#include <sys/stat.h>
#include <sys/types.h>
int mkdir(const char *pathname, mode_t mode);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

```
#include <unistd.h>
int rmdir(const char *pathname);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- **mkdir() : 디렉토리를 생성 시스템호출**
  - "." 디렉토리와 ".." 디렉토리 두 항목을 자동적으로 포함한 새로운 디렉토리가 생성된다.

- **rmdir() : 디렉토리를 삭제 시스템호출**
  - 빈 디렉토리(". " 디렉토리와 ".." 디렉토리만 갖는 디렉토리) 제거
  - rmdir() 호출
    - 디렉토리의 링크 카운트가 0이 되고 다른 프로세스가 이 디렉토리를 오픈하고 있지 않다면, 이 디렉토리가 차지하고 있던 디스크 공간은 새로운 파일이나 디렉토리를 위해 자유 공간으로 리턴
    - 만약 하나 이상의 프로세스가 이 디렉토리를 오픈하고 있다면 이 프로세스들이 모두 디렉토리를 닫은 후에 해당 디렉토리는 제거되며 해당 디렉토리가 제거된 후에 rmdir() 리턴

opendir(3), readdir(3), rewinddir(3), closedir(3), telldir(3), seekdir(3)

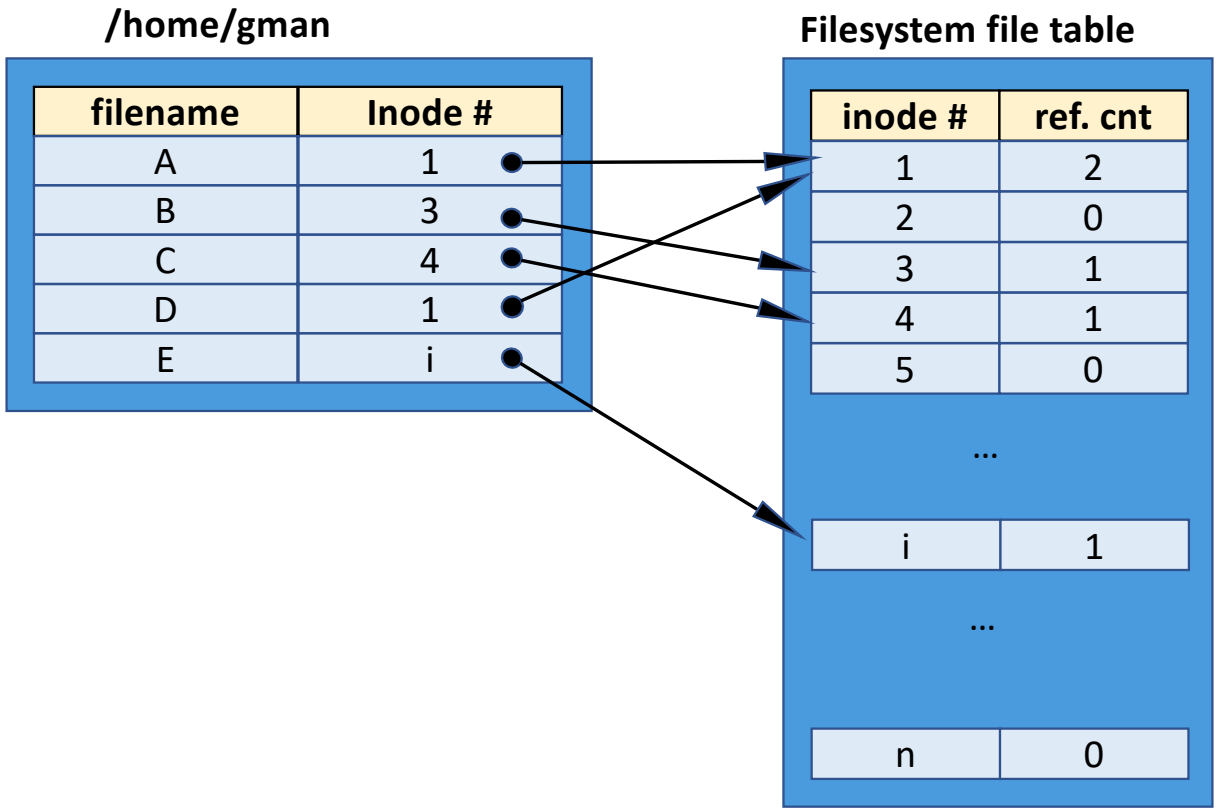
```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name); // 리턴 값: 성공 시 포인터, 에러 시 NULL을 리턴하고 errno가 설정됨

#include <dirent.h>
struct dirent *readdir(DIR *dp); // 리턴 값: 성공 시 포인터, 디렉토리의 끝 또는 에러 시 NULL, errno는 에러 시에만 설정됨

#include <sys/types.h>
#include <dirent.h>
void rewinddir(DIR *dp);

#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dp); // 리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

#include <dirent.h>
long telldir(DIR *dp); // 리턴 값: 성공 시 dp에 해당하는 디렉토리 안의 현재 위치, 에러 시 –1을 리턴하고 errno가 설정됨

#include <dirent.h>
void seekdir(DIR *dp, long loc);
```

- **디렉토리의 내용을 보기 위한 시스템호출**
- **디렉토리는 접근 권한만 있으면 자유롭게 읽을 수 있으나 디렉토리에 대한 쓰기 작업은 커널만 가능**

디렉토리 함수 예제 1, 2

- **교재 참고**
- **<sting.h>**

디렉토리와 파일 테이블


/home/gman
filename
Inode #
A
1
B
3
C
4
D
1
E
i
Filesystem file table
inode #
ref. cnt
1
2
2
0
3
1
4
1
5
0
...
i
1
...
n
0

디렉토리와 inode

inode #2	
. (dot)	2
.. (dot dot)	2
home	123
bin	555
usr	654

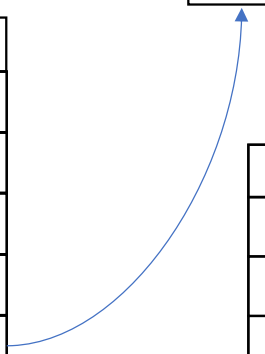
inode 123	
. (dot)	123
.. (dot dot)	2
ian	111
stud0002	755
stud0001	883
stud0003	221

inode 555	
. (dot)	555
.. (dot dot)	2
file1	546
file2	984
file3	333

inode 333
Disk(Data) blocks for file3

inode 335
Disk blocks for login

inode #111	
. (dot)	111
.. (dot dot)	123
.profile	334
.login	335
.logout	433



chdir(2), fchdir(2)

```
#include <unistd.h>
int chdir(const char *pathname);
int fchdir(int filedes);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- **현재 작업 디렉토리를 변경하는 시스템호출**
- **모든 프로세스는 현재 작업 디렉토리에 관한 정보를 가지고 있음**
  - /(루트)로 시작하지 않는 모든 상대적 디렉토리 경로의 시작 위치
  - 사용자가 로그인하면 /etc/passwd 파일에 지정되어 있는 사용자 홈 디렉토리로부터 시작해서 읽는 디렉토리 이름은 프로세스의 한 속성으로 존재
- **chdir()는 지정한 디렉토리에서 많은 파일들을 처리할 필요가 있을 때 유용하게 사용**
  - 절대 경로를 쓰는 것보다 상대 경로를 쓰는 것이 더 효율적
  - fd = open("/usr/oslab/file", O_RDONLY); < chdir("/usr/oslab"); fd = open("file", O_RDONLY);

chdir() 예제

<ssu_chdir.c>

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

int main(void)
{
   if (chdir("/etc") < 0) {
      fprintf(stderr, "chdir error\n");
      exit(1);
   }
   printf("chdir to /etc succeeded.\n");
   exit(0);
}
```

실행 결과

```
oslab@localhost:~/ssu_osdir$ ./ssu_chdir
chdir to /etc succeeded.
```

getcwd(2), get_current_dir_name(2)

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
char *get_current_dir_name(void);
```
리턴 값: 성공 시 현재 작업 디렉토리의 pathname, 에러 시 NULL을 리턴하고 errno가 설정됨

- **현재 작업 디렉토리에 대한 전체 경로 이름을 얻을 수 있는 시스템호출**
  - 시스템은 프로세스의 한 속성으로서 현재 작업 디렉토리에 대한 포인터를 자기고 있음 => 디렉토리 문자열이 아니고, fd 값을 가지고 있음 => 현재 디렉토리 확인 작업 복잡 => fd -> 부모 디렉토리 -> 루트 디렉토리 추적

- **버퍼의 크기는 맨 끝의 NULL을 포함한 임의의 절대경로를 포함할 수 있을 정도의 충분한 크기 확보해야 함**

- **버퍼의 포인터가 NULL -> getcwd() malloc으로 메모리를 할당하고 주소를 리턴**
  - get_current_dir_name() = getcwd(??, ??)

getcwd()예제

<ssu_getcwd.c>

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

#define PATH_MAX    1024

int main(void)
{
   char *pathname;

   if (chdir("/home/oslab") < 0) {
      fprintf(stderr, "chdir error\n");
      exit(1);
   }

   pathname = malloc(PATH_MAX);
   if (getcwd(pathname, PATH_MAX) == NULL) {
      fprintf(stderr, "getcwd error\n");
      exit(1);
   }
   printf("current directory = %s\n", pathname);
   exit(0);
}
```

실행 결과

```
oslab@localhost:~/ssu_osdir$ ./ssu_getcwd
current directory = /home/oslab
```

utime()

```
#include <sys/types.h>
#include <utime.h>
int utime(const char *pathname, const struct utimbuf *times);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- **파일의 최종 접근 시간과 최종 변경 시간을 변경하는 시스템호출**
  - (1) 프로세스의 유효 사용자 ID = 파일의 사용자 ID or (2) 루트권한으로 실행
  - utimbuf 구조체 , <utime.h>
    - 변수형 time_t로 선언된 변수에는 Linux·Unix에서 정의한 캘린더(calander) 시간 저장
    - 캘린더 시간은 1970년 1월 1일 0시부터 시작하여 현재까지의 경과된 시간을 초로 환산한 값

```
struct utimbuf{
   time_t actime;    // 접근 시간
   time_t modtime;  // 수정 시간
}
```

  - utimbuf 구조체에 대한 포인터 == NULL : 파일의 최종 접근 시간과 최종 변경 시간은 현재 시각으로 변경
  - utimbuf 구조체에 대한 포인터!= NULL: 파일의 최종 접근 시간 <= actime, 최종 변경 시간 <= modtime으로 변경

time()

```
#include <time.h>
time_t time(time_t *tloc);  //time_t = long int,
리턴 값: 성공 시 second로 시간 값,  에러 시 –1을 리턴하고 errno가 설정됨
```

- **현재 시간(Unix Time) 리턴하는 시스템호출**
  - time_t : 1970년 1월 1일 0시부터 함수호출까지의 초 카운트 값
    - Calendar time = Unix Time = Epoch Time (2038년 문제, Year 2038 problem, Unix Millennium bug)
    - time_t tloc 저장 영역 문제
      - 1일 = 86, 400 초. (16비트 unsigned short int range, 0~65,535)
      - 초기 16비트 시스템에서 어쩔 수 없이 구조체/배열 사용

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

int main( void)
{
  time_t   current_time;
  time( &current_time);
  printf( "%ldn", current_time);
  printf( ctime( &current_time));
  exit(0);
}
```

asctime(3), ctime(3), gmtime(3), localtime(3), mktime(3)

```
#include <time.h>
char *asctime(const struct tm *tm);
char *asctime_r(const struct tm *tm, char *buf);
char *ctime(const time_t *timep);
char *ctime_r(const time_t *timep, char *buf);
struct tm *gmtime(const time_t *timep);
struct tm *gmtime_r(const time_t *timep, struct tm *result);
struct tm *localtime(const time_t *timep);
struct tm *localtime_r(const time_t *timep, struct tm *result);
time_t mktime(struct tm *tm);
```

```
struct tm {
        int tm_sec; /* seconds */
        int tm_min; /* minutes */
        int tm_hour; /* hours */
        int tm_mday; /* day of the month */
        int tm_mon; /* month */
        int tm_year; /* year */
        int tm_wday; /* day of the week */
        int tm_yday; /* day in the year */
        int tm_isdst; /* daylight saving time */
};
//broken –down time, Coordinated Universal Time (UTC)
```

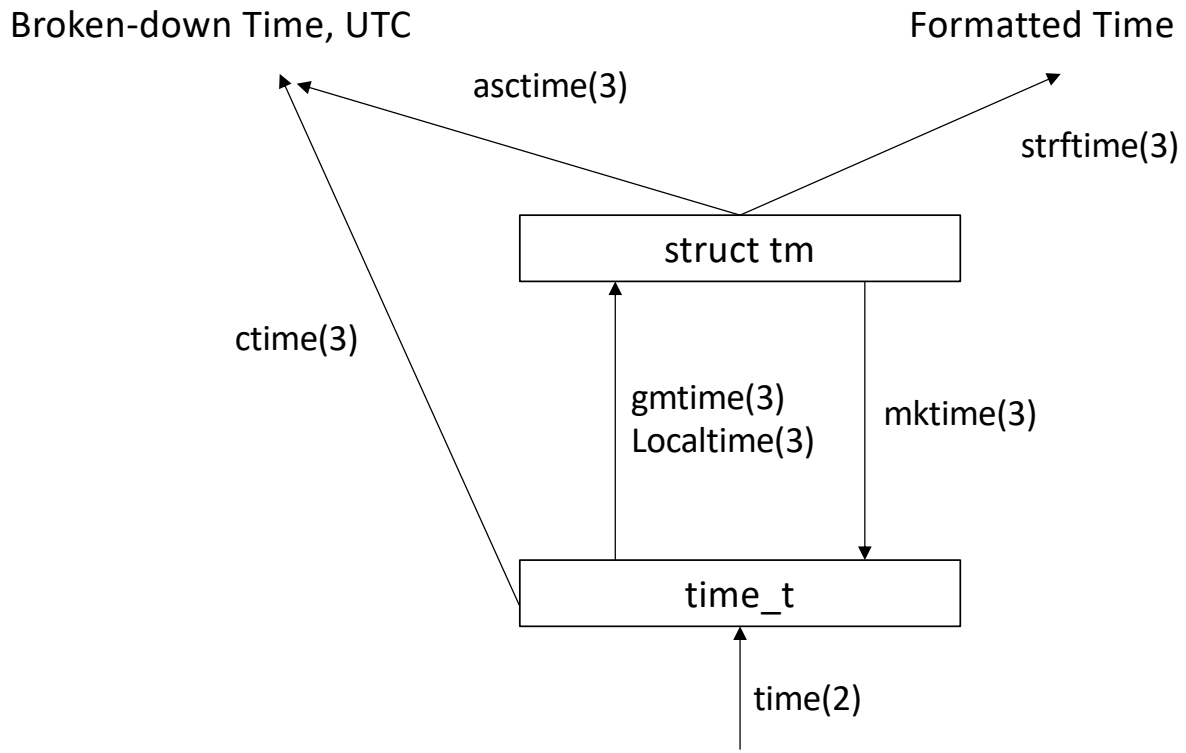
- **asctime(), mktime() : struct tm -> year, month, day, ….**
- **ctime() : (= asctime(localtime(t))) time_t => null-terminated string. ”Wed Jun 30 21:49:08 2019\n”,**
- **gmtime(), localtime() : time_t => struct tm**
- **strftime() : tm => format 에 따라 크기가 max 인 s 문자배열에**

strftime()

```
#include <time.h>

ssize_t strftime(char *s, size_t max, const char *format, const struct tm * tm);
```

%a The abbreviated name of the day of the week according to the current locale. (Calculated from tm_wday.)요일 이름 약자
%A The full name of the day of the week according to the current locale. (Calculated from tm_wday.) 요일 이름
%b The abbreviated month name according to the current locale. (Calculated from tm_mon.)월 이름 약자
%B The full month name according to the current locale. (Calculated from tm_mon.) 월 이름
%c 지역 날짜 시간 %d 날짜 %H 시간(00~23) %I 시간(0-12)
%m 월 (1-12) %M 분(0-59) %p AM/PM %y 년도(00-99) %Y 년도 (2020) %w 요일 ….

From time_t to struct tm



clock(3)

```
#include <time.h>
      clock_t clock(void);
```
리턴값 : 프로그램이 사용한 CPU 시간, 에러시 (clock_t)-1

- **CLOCKS_PER_SEC : 초당 Clock 수, 일반적으로 1,000,000 (32bit, POSIX)**
- **GNU clock_t = long**

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

int main( void)
{
   clock_t start = clock();
   ……
   clock_t end=clock();
   double cpu_time = ((double) (end-start))/CLOCKS_PER_SEC;
  …….
}
```

gettimeofday(2), settimeofday(2)

```
#include <sys/time.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
int settimeofday(const struct timeval *tv, const struct timezone *tz);
리턴값 : 성공시 0, 에러시 -1, EFAULT
```

```
struct timeval
{
   long tv_sec;
   long tv_usec;
}

struct timezone {
   int tz_minuteswest: // 그리니치 서측분차
   int tz_dsttime // DST 보정타입(일광 절약시간) }
}// 거의 사용하지 않고 있음 =>NULL 사용
```

```
#include <stdio.h>
#include <sys/time.h>
int main()
{
   struct timeval start_time, end_time;
   //struct timeval new_time;
   double proc_time;
   gettimeofday(&start_time, NULL);
    //…..settimeofday(&newtime, NULL);
    gettimeofday(&end_time, NULL);
   proc_time = ( end_time.tv_sec – start_time.tv_sec ) + (( end_time.tv_usec – start_time.tv_usec ) / 1000000);
   printf("%f s\n", proc_time);
   exit(0);
}
```