KWANGWOON UNIVERSITY

Network Programming

26. Threads

section

- 26.1 Introduction
- 26.2 Basic Thread Functions: Creation and Termination
- 26.3 str_cli Function Using Threads
- 26.4 TCP Echo Server Using Threads
- 26.5 Thread-Specific Data
- 26.6 Web client and Simultaneous Connections (Continued)
- 26.7 Mutexes : Mutual Exclusion
- 26.8 Condition Variables
- 26.9 Web Client and Simultaneous Connections (Continued)

1. Introduction

- Unix에서 대부분의 네트워크 서버는
  부모 프로세스가 connection을 하고 fork를 통해 자식 프로세스를
  만든다. 그 후 자식 프로세스가 client를 처리하는 형식

- Fork()의 문제점
  1. High cost
  2. Parent process와 child process간의 정보 전달이 느리다.

- Thread는 위의 2가지 문제점을 해결한다.
  1. Thread는 light-weight process
     - thread생성이 process생성보다 10~100배 빠르다.
  2. Process내의 thread는 전역 메모리를 공유
     - 정보를 쉽게 공유 but 단순성 + 동기화 문제

1. Introduction

- Process 내의 thread는 다음과 같은 정보를 공유한다.
  1. Process instructions
  2. Most data
  3. Open files(ex. Descriptors)
  4. Signal handlers and signal dispositions
  5. Current working directory
  6. User and group ID
- 각 스레드는 다음 정보를 자체적으로 가지고 있다.
  1. Thread ID
  2. Stack
  3. Errno
  4. Signal mask
  5. Priority

2. Basic Thread Functions : Creation and Termination

```
#include <pthread.h>

int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(*func) (void *), void *arg);
```

- 새 thread를 성공적으로 생성하면 해당 thread ID가 tid를 통해 반환
- attr 변수는 thread 특성을 지정하기 위해 사용, default 값을 취하려면 NULL을 사용한다.
- 새 thread는 func()을 arg 인자로 실행시키면서 생성된다.
- 생성된 thread는 pthread_exit()을 호출 또는 func()에서 return할 경우 제거된다.

- Thread 생성 성공 시 0을 return
- Thread 생성 실패 시 0이 아닌 에러코드 값을 return

2. Basic Thread Functions : Creation and Termination

```
#include <pthread.h>


int pthread_join (pthread_t tid, void ** status);
```

- pthread_join()을 호출해서 주어진 thread가 종료될 때 까지 기다릴 수 있다.
- **tid – 기다릴 thread의 식별자**
- **status – thread의 return값**
  status가 NULL이 아닌 경우 해당 pointer로 thread의 return 값을 받아올 수 있다.

- **Return값**
  성공 – 0 return
  실패 – error코드 값을 return

2. Basic Thread Functions : Creation and Termination

```
#include <pthread.h>

pthread_t pthread_self (void);
```

- 각 thread는 주어진 process 내에서 이를 식별하는 ID를 가진다.
- Thread ID는 pthread_create()를 통해 return

- Thread는 pthread_self()를 통해 이 값을 자체적으로 가져온다.
  - 현재 thread의 descriptor를 확인할 수 있다.

2. Basic Thread Functions : Creation and Termination

```
#include <pthread.h>

int pthread_detach (pthread_t tid);
```

- Pthread_detach()는 tid를 thread descriptor로 가지는 thread를 main thread에서 분리시킨다.
- Thread가 detach 상태로 되었으면 해당 thread에 대한 pthread_join() 호출은 실패한다.
- 이 때 tid를 가지는 thread가 종료되는 즉시 thread의 정보들을 free한다.

- 성공 시 : 0 return
  실패 시 : 0이 아닌 값 return

2. Basic Thread Functions : Creation and Termination

```
#include <pthread.h>


void pthread_exit (void *status);
```

- 현재 실행중인 thread를 종료시킨다.
- Status는 pthread_join()에서 받아 쓸 수 있다.

3. str_cli Function Using Threads

```
#include     "unpthread.h"
```

unp.h와 pthread.h가 포함

```
void    *copyto (void *);

static int sockfd;                /* global for both threads to access */
static FILE *fp;

void
str_cli(FILE *fp_arg, int sockfd_arg)
{
    char    recvline[MAXLINE];
    pthread_t tid;

    sockfd = sockfd_arg;         /* copy arguments to externals */
    fp = fp_arg;

    Pthread_create(&tid, NULL, copyto, NULL);

    while (Readline(sockfd, recvline, MAXLINE) > 0)
        Fputs(recvline, stdout);
}

void *
copyto(void *arg)
{
    char     sendline[MAXLINE];

    while (Fgets(sendline, MAXLINE, fp) ! = NULL)
        Writen(sockfd, sendline, strlen(sendline));

    Shutdown(sockfd, SHUT_WR); /* EOF on stdin, send FIN */

    return (NULL);
        /* return (i.e., thread terminates) when EOF on stdin */
}
```

새 thread ID가 tid에 저장, copyto함수를 실행하고 인자는 thread에 전달되지 않는다.

4. TCP Echo Server Using Threads

- TCP echo server를 client 당 하나의 thread를 사용한 예

```
#include     "unpthread.h"

static void *doit(void *);      /* each thread executes this function */

int
main(int argc, char **argv)
{
    int     listenfd, connfd;
    pthread_t tid;
    socklen_t addrlen, len;
    struct sockaddr *cliaddr;

    if (argc == 2)
        listenfd = Tcp_listen(NULL, argv[1], &addrlen);
    else if (argc == 3)
        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
    else
        err_quit("usage: tcpserv01 [ <host> ] <service or port>");

    cliaddr = Malloc(addrlen);
    for ( ; ; ) {
        len = addrlen;
        connfd = Accept(listenfd, cliaddr, &len);
        Pthread_create(&tid, NULL, &doit, (void *) connfd);
    }
}

static void *
doit(void *arg)
{
    Pthread_detach(pthread_self());
    str_echo((int) arg);            /* same function as before */
    Close((int) arg);               /* done with connected socket */
    return (NULL);
}
```

Accept가 반환하면 fork 대신 pthread_create를 호출

4. TCP Echo Server Using Threads

- Thread safe : 여러 thread가 동시에 사용해도 안전하다.
- _r을 붙여서 함수를 재정의한다.

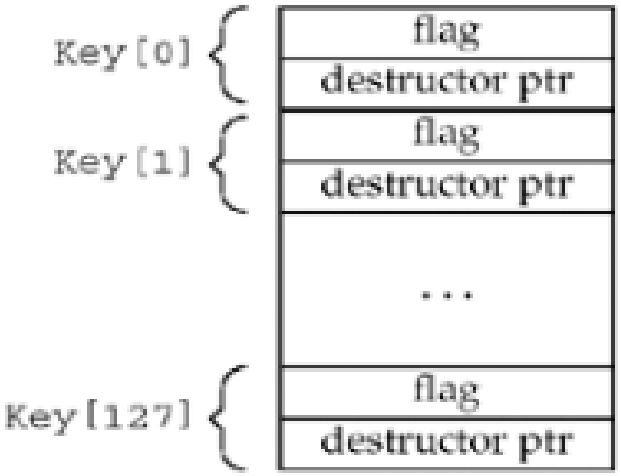
Need not be thread-safe	Must be thread-safe	Comment
asctime	asctime_r	
	ctermid	Thread-safe only if non-null argument
ctime	ctime_r	
getc_unlocked		
getchar_unlocked		
getgrid	getgrid_r	
getgrnam	getgrnam_r	
getlogin	getlogin_r	
getpwnam	getpwnam_r	
getpwuid	getpwuid_r	
gmtime	gmtime_r	
localtime	localtime_r	
putc_unlocked		
putchar_unlocked		
rand	rand_r	
readdir	readdir_r	
strtok	strtok_r	
	tmpnam	Thread-safe only if non-null argument
ttyname	ttyname_r	
gethost*XXX*		
getnet*XXX*		
getproto*XXX*		
getserv*XXX*		
inet_ntoa		

5. Thread-Specific Data

- 멀티 thread를 쓰는 프로그램에서 전역변수를 사용할 때의 문제점
  : 모든 thread가 전역변수를 공유하기 때문에 thread별 전역 변수를 사용할 수 없다.

- 해결책 : Thread-Specific Data를 사용
- 다른 메모리 영역의 주소를 가리키는 각각의 다른 key를 사용함으로써 thread간의 개별적인 전역변수를 access 하게끔 해준다.

5. Thread-Specific Data



- System은 process당 하나의 구조체 배열을 유지 – 키 구조체
- Flag : 배열 요소가 현재 사용중인지 여부 / “not in use”로 초기화
- Thread가 pthread_key_create()를 호출, 새로운 thread-specific data를 작성 -> system은 해당 키 구조배열을 검색, 사용되지 않는 첫 번째 항목을 검색 -> index는 호출 thread로 return된다.

6. Web client and Simultaneous Connections (Continued)

- 16.5의 web client예제를 non-blocking connect 대신 thread를 사용한 예

- Thread를 사용하면 socket을 기본 blocking으로 하고 connection당 하나의 thread를 만들 수 있다.

6. Web client and Simultaneous Connections (Continued)

```
#include     "unpthread.h"
#include     <thread.h>           /* Solaris threads */

#define MAXFILES        20
#define SERV            "80"     /* port number or service name */

struct file {
    char   *f_name;               /* filename */
    char   *f_host;               /* hostname or IP address */
    int     f_fd;                 /* descriptor */
    int     f_flags;              /* F_xxx below */
    pthread_t f_tid;              /* thread ID */
} file[MAXFILES];
#define F_CONNECTING        1     /* connect() in progress */
#define F_READING           2     /* connect() complete; now reading */
#define F_DONE              4     /* all done */

#define GET_CMD       "GET %s HTTP/1.0\r\n\r\n"

int     nconn, nfiles, nlefttoconn, nlefttoread;

void   *do_get_read(void *);
void    home_page(const char *, const char *);
void    write_get_cmd(struct file *);

int
main(int argc, char **argv)
{
    int     i, n, maxnconn;
    pthread_t tid;
    struct file *fptr;

    if (argc < 5)
        err_quit("usage: web <#conns> <IPaddr> <homepage> file1 ...");
    maxnconn = atoi(argv[1]);
    nfiles = min(argc - 4, MAXFILES);
    for (i = 0; i < nfiles; i++) {
        file[i].f_name = argv[i + 4];
        file[i].f_host = argv[2];
        file[i].f_flags = 0;
    }
    printf("nflies = %d\n", nfiles);

    home_page(argv[2], argv[3]);

    nlefttoread = nlefttoconn = nfiles;
    nconn = 0;
```

파일 구조에 thread ID 추가

6. Web client and Simultaneous Connections (Continued)

```
    while (nlefttoread > 0) {
        while (nconn < maxnconn && nlefttoconn > 0) {
                /* find a file to read */
            for (i = 0; i < nfiles; i++)
                if (file[i].f_flags == 0)
                    break;
            if (i == nfiles)
                err_quit("nlefttoconn = %d but nothing found", nlefttoconn);

            file[i].f_flags = F_CONNECTING;
            Pthread_create(&tid, NULL, &do_get_read, &file[i]);
            file[i].f_tid = tid;
            nconn++;
            nlefttoconn--;
        }

        if ( (n = thr_join(0, &tid, (void **) &fptr)) != 0)
            errno = n, err_sys("thr_join error");

        nconn--;
        nlefttoread--;
        printf("thread id %d for %s done\n", tid, fptr->f_name);
    }

    exit(0);
}
```

Thread가 수행하는 함수는 do_get_read
인자는 파일구조에 대한 포인터.

thr_join을 호출하여 thread중 하나가
종료될 때까지 대기한다.

6. Web client and Simultaneous Connections (Continued)

```
void *
do_get_read(void *vptr)
{
    int     fd, n;
    char    line[MAXLINE];
    struct file *fptr;

    fptr = (struct file *) vptr;

    fd = Tcp_connect(fptr->f_host, SERV);
    fptr->f_fd = fd;
    printf("do_get_read for %s, fd %d, thread %d\n",
           fptr->f_name, fd, fptr->f_tid);

    write_get_cmd(fptr);         /* write() the GET command */

        /* Read server's reply  */
    for ( ; ;) {
        if ( (n = Read(fd, line, MAXLINE)) == 0)
            break;               /* server closed connection */

        printf("read %d bytes from %s\n", n, fptr->f_name);
    }
    printf("end-of-file on %s\n", fptr->f_name);
    Close(fd);
    fptr->f_flags = F_DONE;  /* clears F_READING */

    return (fptr);               /* terminate thread */
}
```

Tcp 소켓 생성, connection 설정

7. Mutexes : Mutual Exclusion

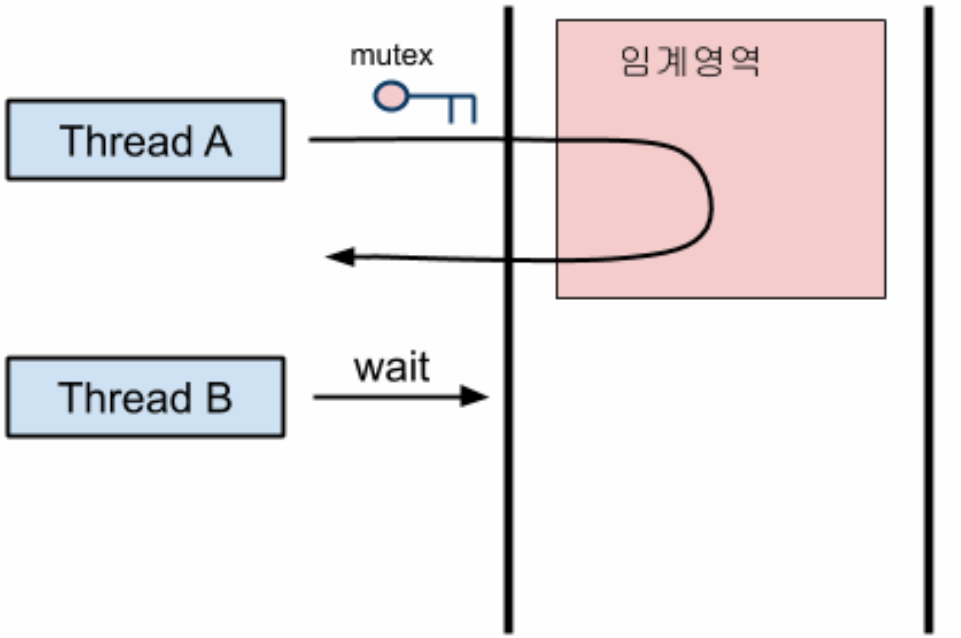
- 여러 thread가 동시에 실행되어 동일한 변수를 access하므로 공유자원영역에 대한 동기화가 필요로 하다.

Time	Thread 1	Thread 2
1	load x into register (5)	
2	add 1 to register (6)	
3	store register in x (6)	
4		load x into register (6)
5		add 1 to register (7)
6		store register in x (7)

Time	Thread 1	Thread 2
1	load x into register (5)	
2	add 1 to register (6)	
3		load x into register (5)
4	store register in x (6)	
5		add 1 to register (6)
6		store register in x (6)

7. Mutexes : Mutual Exclusion

- 보호해야할 공유자원이 있는 공간 : critical area
- Critical area에 들어가기 위한 하나의 키를 가지고 경쟁한다.
- 한 thread가 critical area에 진입하면 다른 thread는 키를 얻을 때 까지 기다려야한다.



7. Mutexes : Mutual Exclusion

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t * mptr);

int pthread_mutex_unlock(pthread_mutex_t * mptr);
```

- 해당 mutex에 대해 lock을 시도한다.
- Thread가 작업을 완료하고 나면 mutex_unlock을 통해 unlock
- 함수 인자에는 해당 mutex가 들어간다.

7. Mutexes : Mutual Exclusion

```
#include    "unpthread.h"

#define NLOOP 5000

int     counter;                /* incremented by threads */
pthread_mutex_t counter_mutex = PTHREAD_MUTEX_INITIALIZER;

void   *doit(void *);

int
main(int argc, char **argv)
{
    pthread_t tidA, tidB;

    Pthread_create(&tidA, NULL, &doit, NULL);
    Pthread_create(&tidB, NULL, &doit, NULL);

        /* wait for both threads to terminate */
    Pthread_join(tidA, NULL);
    Pthread_join(tidB, NULL);

    exit(0);
}
void *
doit(void *vptr)
{
    int     i, val;

    /*
     * Each thread fetches, prints, and increments the counter NLOOP times.
     * The value of the counter should increase monotonically.
     */

    for (i = 0; i < NLOOP; i++) {
        Pthread_mutex_lock(&counter_mutex);

        val = counter;
        printf("%d: %d\n", pthread_self(), val + 1);
        counter = val + 1;

        Pthread_mutex_unlock(&counter_mutex);
    }

    return (NULL);
}
```

8. Condition Variables

- Mutex가 공유자원을 보호해야하는 상호배타를 제공하고 Condition Variables는 신호 메커니즘을 제공(조건 처리)

- Thread가 Condition Variable에 signal이 전달될 때까지 특정 영역에서 대기상태, 다른 threa가 Condition Variable에 signal을 보내면 대기상태가 풀리고 다음 code로 넘어간다.

- Pthreads의 경우 Condition Variables는 pthread_cond_t 유형의 변수 / 다음과 같은 함수와 함께 사용된다.

8. Condition Variables

```
#include <pthread.h>

int pthread_cond_wait(pthread_cond_t *cptr, pthread_mutex_t *mptr);

int pthread_cond_signal(pthread_cond_t *cptr);
```

- **Wait :조건변수 cptr로 signal이 전달되기를 기다린다.**
  - Pthread_cond_wait함수를 호출한 thread는 조건변수에 signal이 전달될 때까지 기다리고 signal을 전달 받아서 thread가 깨어나면 자동적으로 mutex잠금이 된다.

- **Signal : 조건변수 cptr에 signal을 보내서 다른 thread를 깨운다.**

8. Condition Variables

```
#include <pthread.h>

int pthread_cond_broadcast (pthread_cond_t * cptr);

int pthread_cond_timedwait (pthread_cond_t * cptr, pthread_mutex_t *mptr, const struct timespec *abstime);
```

- **Broadcast : 조건변수 cptr에서 기다리는 모든 thread를 깨운다.**
- **Timewait : abstime을 이용해서 제한시간이 지날 때까지 시그널이 전달되지 않을 경우 return 된다는 것을 제외하고는 pthread_cond_wait과 동일하게 작동.**