ISL

Network Programming #8

ISL (IoT Standard Lab)

Index

System call vs Standard IO

- System calls
  - How a user process contacts the Operating System
  - For advanced services that may require special privilege
- Standard I/O library
  - Generic I/O support for C programs
  - A smart wrapper around I/O-related system calls
  - Stream concept, line-by-line input, formatted output, ...

```c
int main(int argc, char *argv[])
{
    int fd1, fd2, len;
    char buf[BUF_SIZE];
    long long start, end;

    fd1=open("news.txt", O_RDONLY);
    fd2=open("cpy.txt", O_WRONLY|O_CREAT|O_TRUNC);


    start = current_timestamp();
    while((len=read(fd1, buf, sizeof(buf)))>0)
        write(fd2, buf, len);
    end = current_timestamp();

    printf("%lld msec\n", end - start);
    close(fd1);
    close(fd2);
    return 0;
}
```

System call

```c
int main(int argc, char *argv[])
{
    FILE * fp1;
    FILE * fp2;
    char buf[BUF_SIZE];
    long long start, end;

    fp1=fopen("news.txt", "r");
    fp2=fopen("cpy.txt", "w");

    start = current_timestamp();
    while(fgets(buf, BUF_SIZE, fp1)!=NULL)
        fputs(buf, fp2);
    end = current_timestamp();

    printf("%lld msec\n", end - start);
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

Standard IO

Glibc – fopen.c

```c
FILE *
rpl_fopen (const char *filename, const char *mode)
{
#if defined _WIN32 && ! defined __CYGWIN__
  if (strcmp (filename, "/dev/null") == 0)
    filename = "NUL";
#endif
```

```c
    size_t len = strlen (filename);
    if (len > 0 && filename[len - 1] == '/')
      {
        int fd;
        struct stat statbuf;
        FILE *fp;

        if (mode[0] == 'w' || mode[0] == 'a')
          {
            errno = EISDIR;
            return NULL;
          }

        fd = open (filename, O_RDONLY);
        if (fd < 0)
          return NULL;

        if (fstat (fd, &statbuf) >= 0 && !S_ISDIR (statbuf.st_mode))
          {
            close (fd);
            errno = ENOTDIR;
            return NULL;
          }

        fp = fdopen (fd, mode);
        if (fp == NULL)
          {
            int saved_errno = errno;
            close (fd);
            errno = saved_errno;
          }
        return fp;
      }
```

System call vs Standard IO

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./syscpy
3 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./syscpy
3 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./syscpy
3 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./syscpy
4 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./syscpy
3 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./syscpy
3 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./syscpy
3 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./stdcpy
0 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./stdcpy
0 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./stdcpy
1 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./stdcpy
0 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./stdcpy
0 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./stdcpy
0 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./stdcpy
0 msec
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

fdopen() & fileno()

```
#include <stdio.h>

FILE * fdopen(int fildes, const char * mode);
    ➡ 성공 시 변환된 FILE 구조체 포인터, 실패 시 NULL 반환
```

- fildes : 변환할 파일 디스크립터를 인자로 전달.
- mode : 생성할 FILE 구조체 포인터의 모드(mode)정보 전달.

```
#include <stdio.h>

int fileno(FILE * stream);
    ➡ 성공 시 변환된 파일 디스크립터, 실패 시 -1 반환
```

```c
int main(void)
{
    FILE *fp;
    int fd=open("data.dat", O_WRONLY|O_CREAT|O_TRUNC);
    if(fd==-1)
    {
        fputs("file open error", stdout);
        return -1;
    }

    fp=fdopen(fd, "w");
    fputs("Network C programming \n", fp);
    fclose(fp);
    return 0;
}
```

open()으로 반환받은 fd로 fp를 반환

data.dat

socket8 > stdio > data.dat

```
1  Network C programming
2
```

```c
int main(void)
{
    FILE *fp;
    int fd=open("data.dat", O_WRONLY|O_CREAT|O_TRUNC);
    if(fd==-1)
    {
        fputs("file open error", stdout);
        return -1;
    }

    printf("First file descriptor: %d \n", fd);
    fp=fdopen(fd, "w");
    fputs("TCP/IP SOCKET PROGRAMMING \n", fp);
    printf("Second file descriptor: %d \n", fileno(fp));
    fclose(fp);
    return 0;
}
```

fdopen()으로 반환한 fp를 기반으로 fd 반환

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./fileno
First file descriptor: 3
Second file descriptor: 3
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

Standard IO Server & Client

```c
for(i=0; i<5; i++)
{
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
    if(clnt_sock==-1)
        error_handling("accept() error");
    else
        printf("Connected client %d \n", i+1);

    readfp=fdopen(clnt_sock, "r");
    writefp=fdopen(clnt_sock, "w");

    while(!feof(readfp))
    {
        fgets(message, BUF_SIZE, readfp);
        fputs(message, writefp);
        fflush(writefp);
    }
    fclose(readfp);
    fclose(writefp);
}
close(serv_sock);
return 0;
```

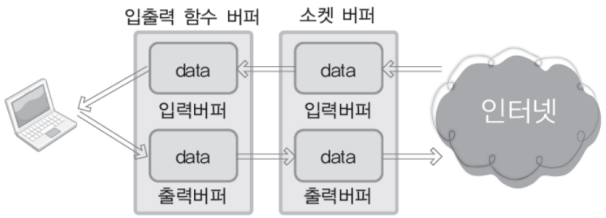
입력용, 출력용 FILE 구조체 포인터를 각각 생성해야 한다.

표준 C 입출력 함수를 사용할 경우 소켓의 버퍼 이외에 버퍼링이 되기 때문에 필요하다면, fflush 함수를 직접 호출해야 한다.

Server

```c
if(connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
    error_handling("connect() error!");
else
    puts("Connected...........");

readfp=fdopen(sock, "r");
writefp=fdopen(sock, "w");

while(1)
{
    fputs("Input message(Q to quit): ", stdout);
    fgets(message, BUF_SIZE, stdin);
    if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
        break;

    fputs(message, writefp);
    fflush(writefp);
    fgets(message, BUF_SIZE, readfp);
    printf("Message from server: %s", message);
}
fclose(writefp);
fclose(readfp);
return 0;
```

Client



Standard IO Server & Client

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./server 8080
Connected client 1
```

Server

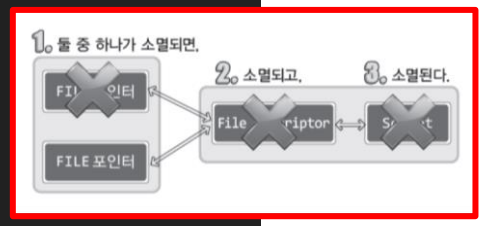
```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./client 127.0.0.1 8080
Connected...........
Input message(Q to quit): Hi
Message from server: Hi
Input message(Q to quit): Hello
Message from server: Hello
Input message(Q to quit): Bye
Message from server: Bye
Input message(Q to quit): Q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

Client

File Descriptor Copy & Half Close - fclose()

Server - fclose()

```
clnt_adr_sz=sizeof(clnt_adr);
clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr,&clnt_adr_sz);

readfp=fdopen(clnt_sock, "r");
writefp=fdopen(clnt_sock, "w");

fputs("FROM SERVER: Hi~ client? \n", writefp);
fputs("I love all of the world \n", writefp);
fputs("You are awesome! \n", writefp);
fflush(writefp);

fclose(writefp);
fgets(buf, sizeof(buf), readfp); fputs(buf, stdout);
fclose(readfp);
return 0;
```



클라이언트 소켓 파일 디스크립터가 close() 되면서 바로 커넥션 종료

Client

```
connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
readfp=fdopen(sock, "r");
writefp=fdopen(sock, "w");

while(1)
{
    if(fgets(buf, sizeof(buf), readfp)==NULL)
        break;
    fputs(buf, stdout);
    fflush(stdout);
}

fputs("FROM CLIENT: Thank you! \n", writefp);
fflush(writefp);
fclose(writefp); fclose(readfp);
return 0;
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./sep_serv 8080
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./sep_cli 127.0.0.1 8080
FROM SERVER: Hi~ client?
I love all of the world
You are awesome!
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

File Descriptor Copy & Half close – dup()

```c
int main(int argc, char *argv[])
{
    int cfd1, cfd2;
    char str1[]="Hi~ \n";
    char str2[]="It's nice day~ \n";

    cfd1=dup(1);
    cfd2=dup2(cfd1, 7);

    printf("fd1=%d, fd2=%d \n", cfd1, cfd2);
    write(cfd1, str1, sizeof(str1));
    write(cfd2, str2, sizeof(str2));

    close(cfd1);
    close(cfd2);
    write(1, str1, sizeof(str1));
    close(1);
    write(1, str2, sizeof(str2));
    return 0;
}
```

stdout(1) 복사

stdout(1)를 7로 복사

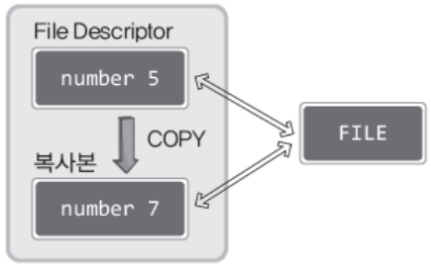
stdout(1)을 close하였으므로 파일 디스크립터가 close 되어 출력 X

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./dup
fd1=3, fd2=7
Hi~
It's nice day~
Hi~
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

File Descriptor Copy & Half close – dup()

```c
#define BUF_SIZE 3
int main(int argc, char *argv[])
{
    int fd, len, dfd;
    char buf[BUF_SIZE];

    fd=open("test.txt", O_RDONLY);
    dfd = dup(fd);

    len=read(fd, buf, sizeof(buf));
    printf("fd: %s\n", buf);
    close(fd);

    len = read(dfd, buf, sizeof(buf));
    printf("dfd: %s\n", buf);
    close(dfd);

    return 0;
}
```

원본을 close




```
smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week10$ ./fd_test
fd: 123
dfd: 456
smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week10$
```

File Descriptor Copy & Half close – dup() & fileno()

Server - fileno()

```c
clnt_adr_sz=sizeof(clnt_adr);
clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr,&clnt_adr_sz);

readfp=fdopen(clnt_sock, "r");
writefp=fdopen(dup(clnt_sock), "w");

fputs("FROM SERVER: Hi~ client? \n", writefp);
fputs("I love all of the world \n", writefp);
fputs("You are awesome! \n", writefp);
fflush(writefp);

shutdown(fileno(writefp), SHUT_WR);
fclose(writefp);

fgets(buf, sizeof(buf), readfp); fputs(buf, stdout);
fclose(readfp);
return 0;
```

상대방에게 EOF 전달하고, 원본이 남아있으므로 커넥션을 유지

Client

```c
connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
readfp=fdopen(sock, "r");
writefp=fdopen(sock, "w");

while(1)
{
    if(fgets(buf, sizeof(buf), readfp)==NULL)
        break;
    fputs(buf, stdout);
    fflush(stdout);
}

fputs("FROM CLIENT: Thank you! \n", writefp);
fflush(writefp);
fclose(writefp); fclose(readfp);
return 0;
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./sep_serv2 8080
FROM CLIENT: Thank you!
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$ ./sep_cli 127.0.0.1 8080
FROM SERVER: Hi~ client?
I love all of the world
You are awesome!
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/stdio$
```

Thread Create 1

```c
int main(int argc, char *argv[])
{
    pthread_t t_id;
    int thread_param=5;

    if(pthread_create(&t_id, NULL, thread_main, (void*)&thread_param)!=0)
    {
        puts("pthread_create() error");
        return -1;
    };
    sleep(10);  puts("end of main");
    return 0;
}

void* thread_main(void *arg)
{
    int i;
    int cnt=*((int*)arg);
    for(i=0; i<cnt; i++)
    {
        sleep(1);  puts("running thread");
    }
    return NULL;
}
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ gcc thread1.c -o thread1 -lpthread
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./thread1
running thread
running thread
running thread
running thread
running thread
end of main
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$
```

빌드 시 –lpthread 옵션을 추가
(Link pthread)

running thread가 5초에 걸쳐 출력된 후 5초 뒤에 main 종료

```c
#include <pthread.h>

int pthread_create (
    pthread_t *restrict thread, const pthread_attr_t *restrict attr,
    void *(*start_routine)(void*), void *restrict arg
);
```

➜ 성공 시 0, 실패 시 0 이외의 값 반환

thread	생성할 쓰레드의 ID 저장을 위한 변수의 주소 값 전달, 참고로 쓰레드는 프로세스와 마찬가지로 쓰레드의 구분을 위한 ID가 부여된다.
attr	쓰레드에 부여할 특성 정보의 전달을 위한 매개변수, NULL 전달 시 기본적인 특성의 쓰레드가 생성된다.
start_routine	쓰레드의 main 함수 역할을 하는, 별도 실행흐름의 시작이 되는 함수의 주소 값(함수 포인터) 전달.
arg	세 번째 인자를 통해 등록된 함수가 호출될 때 전달할 인자의 정보를 담고 있는 변수의 주소 값 전달.

Thread Create 2

```c
int main(int argc, char *argv[])
{
    pthread_t t_id;
    int thread_param=5;
    void * thr_ret;

    if(pthread_create(&t_id, NULL, thread_main, (void*)&thread_param)!=0)
    {
        puts("pthread_create() error");
        return -1;
    };

    if(pthread_join(t_id, &thr_ret)!=0)
    {
        puts("pthread_join() error");
        return -1;
    };

    printf("Thread return message: %s \n", (char*)thr_ret);
    free(thr_ret);
    return 0;
}
```

```c
void* thread_main(void *arg)
{
    int i;
    int cnt=*((int*)arg);
    char * msg=(char *)malloc(sizeof(char)*50);
    strcpy(msg, "Hello, I'am thread~ \n");

    for(i=0; i<cnt; i++)
    {
        sleep(1);  puts("running thread");
    }
    return (void*)msg;
}
```

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **status);
    ➜ 성공 시 0, 실패 시 0 이외의 값 반환

  thread   이 매개변수에 전달되는 ID의 쓰레드가 종료될 때까지 함수는 반환하지 않는다.
  status   쓰레드의 main 함수가 반환하는 값이 저장될 포인터 변수의 주소 값을 전달한다.
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./thread2
running thread
running thread
running thread
running thread
running thread
Thread return message: Hello, I'am thread~

smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$
```

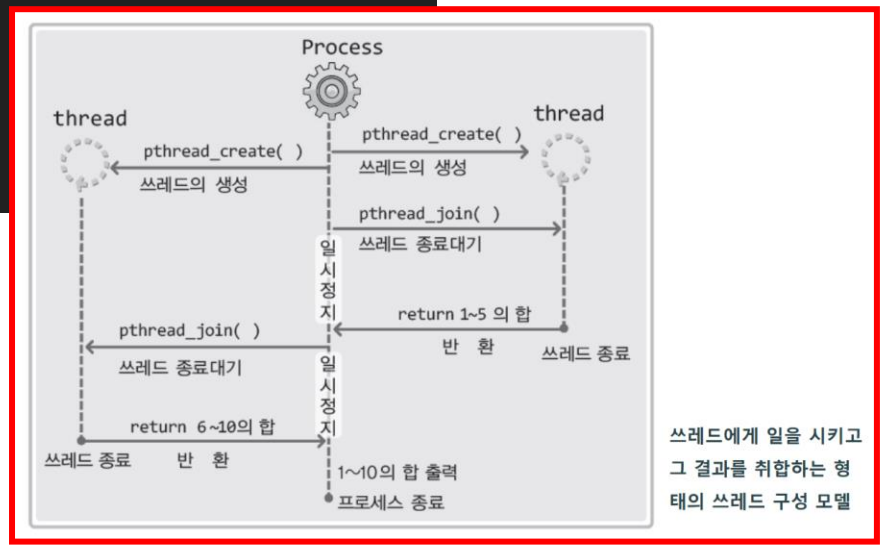
running thread가 5초에 걸쳐 출력된 후 바로 main 종료

Thread Create 3

```c
int sum=0;

int main(int argc, char *argv[])
{
    pthread_t id_t1, id_t2;
    int range1[]={1, 5};
    int range2[]={6, 10};

    pthread_create(&id_t1, NULL, thread_summation, (void *)range1);
    pthread_create(&id_t2, NULL, thread_summation, (void *)range2);

    pthread_join(id_t1, NULL);
    pthread_join(id_t2, NULL);
    printf("result: %d \n", sum);
    return 0;
}
```

```c
void * thread_summation(void * arg)
{
    int start=((int*)arg)[0];
    int end=((int*)arg)[1];

    while(start<=end)
    {
        sum+=start;
        start++;
    }
    return NULL;
}
```

```
smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week10$ ./thread3
result: 55
smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week10$ ./thread3
result: 15
smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week10$ ./thread3
result: 55
```



쓰레드에게 일을 시키고 그 결과를 취합하는 형태의 쓰레드 구성 모델

Critical Section

```
long long num=0;
```

전역 변수 num을 2개의 스레드가 동시 접근

```
void * thread_inc(void * arg)
{
    int i;
    for(i=0; i<50000; i++)
        num+=1;
    return NULL;
}
void * thread_des(void * arg)
{
    int i;
    for(i=0; i<50000; i++)
    {
        num-=1;
    }
    return NULL;
}
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ gcc mutex.c -o critical -lpthread
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: -17914
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: 6552
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: -27889
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: -38565
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: 6178
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: -18437
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: 19859
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: -9113
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: -2529
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./critical
result: -7183
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$
```

Critical Section (Mutex)

```
long long num=0;
```

전역 변수 num을 2개의 스레드가 동시 접근

```
long long num=0;
pthread_mutex_t mutex;

int main(int argc, char *argv[])
{
    pthread_t thread_id[NUM_THREAD];
    int i;

    pthread_mutex_init(&mutex, NULL);

    for(i=0; i<NUM_THREAD; i++)
    {
        if(i%2)
            pthread_create(&(thread_id[i]), NULL, thread_inc, NULL);
        else
            pthread_create(&(thread_id[i]), NULL, thread_des, NULL);
    }

    for(i=0; i<NUM_THREAD; i++)
        pthread_join(thread_id[i], NULL);

    printf("result: %lld \n", num);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Mutex 초기화

Mutex 소멸

```
void * thread_inc(void * arg)
{
    int i;
    pthread_mutex_lock(&mutex);
    for(i=0; i<50000; i++)
        num+=1;
    pthread_mutex_unlock(&mutex);
    return NULL;
}
void * thread_des(void * arg)
{
    int i;
    for(i=0; i<50000; i++)
    {
        pthread_mutex_lock(&mutex);
        num-=1;
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Mutex Lock

Critical Section

Mutex Unlock

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ gcc mutex.c -D_REENTRANT -o mutex -lpthread
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./mutex
result: 0
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multi
```

헤더파일 선언 이전에 매크로 _REENTRANT를 정의하면, 쓰레드에 불안전한 함수의 호출문을 쓰레드에 안전한 함수의 호출문으로 자동 변경 컴파일 된다.

```
root@my_linux:/tcpip# gcc -D_REENTRANT mythread.c -o mthread -lpthread
```

Critical Section (Semaphore)

```c
static sem_t sem_one;
static sem_t sem_two;
static int num;

int main(int argc, char *argv[])
{
    pthread_t id_t1, id_t2;
    sem_init(&sem_one, 0, 0);
    sem_init(&sem_two, 0, 1);

    pthread_create(&id_t1, NULL, read, NULL);
    pthread_create(&id_t2, NULL, accu, NULL);

    pthread_join(id_t1, NULL);
    pthread_join(id_t2, NULL);

    sem_destroy(&sem_one);
    sem_destroy(&sem_two);
    return 0;
}
```

세마포어 카운트가 0이면 진입불가, 0보다 크면 진입가능

```c
void * read(void * arg)
{
    int i;
    for(i=0; i<5; i++)
    {
        fputs("Input num: ", stdout);

        sem_wait(&sem_two);
        scanf("%d", &num);
        sem_post(&sem_one);
    }
    return NULL;
}
void * accu(void * arg)
{
    int sum=0, i;
    for(i=0; i<5; i++)
    {
        sem_wait(&sem_one);
        sum+=num;
        sem_post(&sem_two);
    }
    printf("Result: %d \n", sum);
    return NULL;
}
```

시작 시 0보다 크므로 sem_two를 -1하고 진입 (결과: sem_two = 0)

키보드 입력받고 sem_one을 +1(결과: sem_one = 1)

시작 시 00이므로 대기하고, read()하는 스레드가 +1하면 -1하고 진입
(결과: sem_one = 0)

Sum에 키보드로 입력받은 num값 누적시키고 sem_two를 +1(결과: sum_two = 1)

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);
    ➜ 성공 시 0, 실패 시 0 이외의 값 반환
```

- sem — 세마포어 생성시에는 세마포어의 참조 값 저장을 위한 변수의 주소 값 전달, 그리고 세마포어 소멸 시에는 소멸하고자 하는 세마포어의 참조 값을 저장하고 있는 변수의 주소 값 전달.
- pshared — 0 이외의 값 전달 시, 둘 이상의 프로세스에 의해 접근 가능한 세마포어 생성, 0 전달 시 하나의 프로세스 내에서만 접근 가능한 세마포어 생성, 우리는 하나의 프로세스 내에 존재하는 쓰레드의 동기화가 목적이므로 0을 전달한다.
- value — 생성되는 세마포어의 초기 값 지정.

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ gcc semaphore.c -o sema -lpthread
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./sema
Input num: 3
Input num: 4
Input num: 5
Input num: 6
Input num: 7
Result: 25
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$
```

Chat Server based on Multithread

```c
int main(int argc, char *argv[])
{
    int serv_sock, clnt_sock;
    struct sockaddr_in serv_adr, clnt_adr;
    int clnt_adr_sz;
    pthread_t t_id;
    if(argc!=2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    pthread_mutex_init(&mutx, NULL);
    serv_sock=socket(PF_INET, SOCK_STREAM, 0);

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_adr.sin_port=htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*) &serv_adr, sizeof(serv_adr))==-1)
        error_handling("bind() error");
    if(listen(serv_sock, 5)==-1)
        error_handling("listen() error");

    while(1)
    {
        clnt_adr_sz=sizeof(clnt_adr);
        clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr,&clnt_adr_sz);

        pthread_mutex_lock(&mutx);
        clnt_socks[clnt_cnt++]=clnt_sock;
        pthread_mutex_unlock(&mutx);

        pthread_create(&t_id, NULL, handle_clnt, (void*)&clnt_sock);
        pthread_detach(t_id);
        printf("Connected client IP: %s \n", inet_ntoa(clnt_adr.sin_addr));
    }
    close(serv_sock);
    return 0;
}
```

```c
int clnt_cnt=0;
int clnt_socks[MAX_CLNT];
pthread_mutex_t mutx;
```

Critical Section

클라이언트 소켓 fd를 안전하게 **추가**하기 위한 Mutex

```c
#include <pthread.h>

int pthread_detach(pthread_t thread);
    ➔ 성공 시 0, 실패 시 0 이외의 값 반환
```

pthread_join 함수의 호출은 블로킹 상태에 놓이게 되니 pthread_detach 함수를 호출해서 쓰레드의 소멸을 도와야 한다.

```c
void * handle_clnt(void * arg)
{
    int clnt_sock=*((int*)arg);
    int str_len=0, i;
    char msg[BUF_SIZE];

    while((str_len=read(clnt_sock, msg, sizeof(msg)))!=0)
        send_msg(msg, str_len);

    pthread_mutex_lock(&mutx);
    for(i=0; i<clnt_cnt; i++)   // remove disconnected client
    {
        if(clnt_sock==clnt_socks[i])
        {
            while(i++<clnt_cnt-1)
                clnt_socks[i]=clnt_socks[i+1];
            break;
        }
    }
    clnt_cnt--;
    pthread_mutex_unlock(&mutx);
    close(clnt_sock);
    return NULL;
}
```

클라이언트 소켓 fd를 안전하게 **삭제**하기 위한 Mutex

```c
void send_msg(char * msg, int len)   // send to all
{
    int i;
    pthread_mutex_lock(&mutx);
    for(i=0; i<clnt_cnt; i++)
        write(clnt_socks[i], msg, len);
    pthread_mutex_unlock(&mutx);
}
```

클라이언트 소켓 fd로 안전하게 **write**하기 위한 Mutex

Chat Client based on Multithread

```c
int main(int argc, char *argv[])
{
    int sock;
    struct sockaddr_in serv_addr;
    pthread_t snd_thread, rcv_thread;
    void * thread_return;
    if(argc!=4) {
        printf("Usage : %s <IP> <port> <name>\n", argv[0]);
        exit(1);
     }

    sprintf(name, "[%s]", argv[3]);
    sock=socket(PF_INET, SOCK_STREAM, 0);

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_addr.sin_port=htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
        error_handling("connect() error");

    pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
    pthread_create(&rcv_thread, NULL, recv_msg, (void*)&sock);
    pthread_join(snd_thread, &thread_return);
    pthread_join(rcv_thread, &thread_return);
    close(sock);
    return 0;
}
```

```c
void * send_msg(void * arg)   // send thread main
{
    int sock=*((int*)arg);
    char name_msg[NAME_SIZE+BUF_SIZE];
    while(1)
    {
        fgets(msg, BUF_SIZE, stdin);
        if(!strcmp(msg,"q\n")||!strcmp(msg,"Q\n"))
        {
            close(sock);
            exit(0);
        }
        sprintf(name_msg,"%s %s", name, msg);
        write(sock, name_msg, strlen(name_msg));
    }
    return NULL;
}
```

"[이름] 메시지" 형태로 문자열을 만들고 서버에게 전송

```c
void * recv_msg(void * arg)   // read thread main
{
    int sock=*((int*)arg);
    char name_msg[NAME_SIZE+BUF_SIZE];
    int str_len;
    while(1)
    {
        str_len=read(sock, name_msg, NAME_SIZE+BUF_SIZE-1);
        if(str_len==-1)
            return (void*)-1;
        name_msg[str_len]=0;
        fputs(name_msg, stdout);
    }
    return NULL;
}
```

서버가 "[이름] 메시지" 형태로 문자열을 받아 표준 출력

Chat Program based on Multithread

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./chat_serv 8080
Connected client IP: 127.0.0.1
Connected client IP: 127.0.0.1
```

Server

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./chat_clnt 127.0.0.1 8080 a
hi
[a] hi
[b] hi
hello
[a] hello
[b] hello
q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$
```

Client a

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$ ./chat_clnt 127.0.0.1 8080 b
[a] hi
hi
[b] hi
[a] hello
hello
[b] hello
q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/multithread$
```

Client b

03 Network assignment #8

Multithread Calculator Server & Client

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$ ./server 8080
Connected client Port: 21138
Connected client Port: 21650
Connected client Port: 22162
Closed client
Closed client
Closed client
```

클라이언트가 이용한 포트번호를 출력

Server

띄어쓰기 기준으로 Operand Count, Operand, Operator를 입력

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$ ./client 8080 127.0.0.1
 aaaa
3 1 2 3 + +
[aaaa] 1+2+3=6
[bbbb] 3*6=18
[cccc] 10+4-5+23=32
0
Overflow Number(0) - Closed client
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$
```

Client [aaaa]

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$ ./client 8080 127.0.0.1
 bbbb
[aaaa] 1+2+3=6
2 3 6 *
[bbbb] 3*6=18
[cccc] 10+4-5+23=32
128
Overflow Number(-128) - Closed client
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$
```

Client [bbbb]

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$ ./client 8080 127.0.0.1
cccc
[aaaa] 1+2+3=6
[bbbb] 3*6=18
4 10 4 5 23 + - +
[cccc] 10+4-5+23=32
256
Overflow Number(0) - Closed client
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$
```

Client [cccc]

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$ ./client 8080 127.0.0.1 a
ID have to be 4
smalldragon@SD-DESKTOP:~/Workspace/socket/socket8/assignment$
```

Client [a]

03 Network assignment #8

Multithread Calculator Server & Client

- **server.c**

1. Main 함수 인자로부터 포트번호를 받고 이를 활용하여 서버 소켓을 생성

   Ex) ./server 8080

2. 멀티스레드 형식으로 서버를 구현하며 메인 스레드의 역할은 다음과 같음
   1. 클라이언트의 요청이 오면 Mutex를 이용하여 클라이언트 소켓 fd(File Descriptor)를 관리하는 배열에 스레드가 접근하지 못하도록 하고, 전체 소켓 fd를 관리하는 배열의 count 수를 늘려서 저장함
   2. 스레드를 생성하고 생성된 클라이언트 소켓 fd를 인자로 넘겨줌
   3. 스레드를 detach하고 "Connected client Port: 클라이언트포트번호"를 표준 출력한 뒤, 다음 연결 요청을 대기함
3. 클라이언트의 요청을 처리하는 워커 스레드의 역할은 다음과 같음
   1. 메인 스레드로부터 받은 클라이언트 소켓 fd를 활용하여 해당 클라이언트의 계산 요청을 readv()로 대기함
      1. 클라이언트는 writev()를 통해 첫 번째 배열에는 4글짜의 ID 값을 주고, 두 번째 배열에는 Network Assignment #2 과제와 동일한 계산 데이터를 줌
   2. readv()에서 계산 요청을 받으면 계산을 해서 결과 값과 함께 아래와 같은 문자열 형식을 만듦
      1. [id] 계산식: 계산결과 `[aaaa] 1+2+3=6`
   3. 이 문자열을 현재 연결되어 있는 모든 클라이언트에게 write()하고 다음 요청을 대기함
   4. 이 과정에서 Mutex를 활용하여 공유 자원을 참조할 때 의도치 않은 상황이 일어나는 것을 배제함
4. 클라이언트가 연결을 종료한다면 Closed client와 함께 연결을 종료함 – detach를 통해 스레드 자원은 반납되도록 설정

03 Network assignment #8

Multithread Calculator Server & Client

- **client.c**

1. Main 함수 인자로부터 포트번호, IP, ID(4글짜) 순으로 데이터를 받음

   Ex) ./client 8080 127.0.0.1 aaaa

2. ID가 4글짜가 아니라면 “ID have to be 4” 메시지를 표준 출력하고 종료
3. 소켓을 열어 서버와 연결하고 멀티스레드 형식으로 Send 및 Receive를 하는 스레드들을 생성하여 서버와의 통신을 진행
4. Send 스레드의 경우 다음과 같이 동작함
   1. 키보드를 통해 띄어쓰기를 기준으로 아래와 같이 Network assignment #2와 같은 계산 요청 데이터를 구성함

      Ex) 3 1 2 3 + +

   2. 이때 바이트 수 역시 Network assignment #2와 동일하게 구성해야 됨
   3. iovec구조체를 활용하여, 첫 번째 배열에는 main 함수 인자로 얻은 ID를 입력 받고, 두 번째 배열에 계산 요청 데이터를 입력 받아서 writev()를 통해 데이터를 전송함
   4. 전송 후에는 다시 키보드를 통해 계산 데이터를 받는 것을 대기함
   5. 만약 oprand count 값으로 1바이트 기준 오버플로우되는 값을 키보드로 받았다면, “Overflow Number(오버플로우된 값) – Closed client”를 표준 출력하고 종료
5. Receive 스레드의 경우 다음과 같이 동작함
   1. 서버로부터의 데이터를 수신하기 위해 read()를 하여 대기함
   2. 데이터를 수신하면 이를 화면 그대로 출력해주고 다시 대기함

03 Network assignment #8

Multi Process Calculator Server

- 참고사항

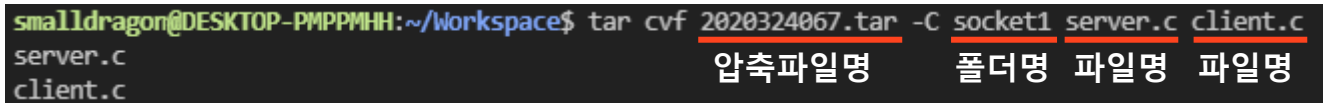
1. 과제에서 의도한 대로 데이터를 주고받고 이를 출력하는 방식이 아닌, 겉으로 출력 결과만 똑같이 보인다면 점수 없음
2. 빌드 시 Warning이 발생할 경우 점수 없음

- 제출관련

1. 서버 프로그램은 server.c, 클라이언트 프로그램은 client.c로 명명하여 과제 진행
2. 빌드 시(gcc) Warning이 발생해서는 안됨
3. 제출 시 파일들을 “자신의 학번.tar” 파일로 제출

   Ex) 2020324067.tar

   ~/Workspace/socket1/(server.c, client.c)



4. 과제는 10점 만점
5. 제출 기한: 2023.05.19(금) PM 11:59
6. 지각 제출 허용: 2023.05.23(화) PM 11:59 / 하루 늦을 때 마다 2점 씩 감점

   지각제출 시 보낼 이메일: eunjia24@gmail.com
7. 기한 안에 아예 제출을 하지 않았을 시 점수 없음