POSIX thread 를
pthread 라고 하며 쓰려면 반드시 #include <pthread.h> 헤더파일을 선언해주어야 한다

쓰레드의 일반적인 개념을 정리했다.

기술적으로 thread 는 운영체제에서 스케쥴링이 가능한 독립된 명령(instructions)흐름의 단위이다. 여기에서 독립된 명령 단위를 보통 문맥이라고 표기한다. 스케쥴링 가능한 문맥 혹은 코드조각 으로 이해하면 될 것 같다.

소프트웨어 개발자에게 있어서 thread 는 main 프로그램으로 부터 독립적으로 실행될 수 있는 procedure 로 보는게 이해가 더 쉬울 수 있을 것 같다.

main 프로그램은 여러개의 독립적으로 실행 가능한 프로시져 즉 쓰레드를 포함할 수 있는데, 이들을 동시에 운용할 경우 이를 multi thread 프로그램이라고 한다.

아래는 쓰레드간에 서로 공유하는 자원들

작업디렉토리

파일지시자들

대부분의 전역변수와 데이타들

UID 와 GID

signal

다음은 각각의 쓰레드가 고유하게 가지는 자원들을 나열한 것이다.

에러번호(errno)

쓰레드 우선순위

스택

쓰레드 ID

레지스터 및 스택지시자

shared memory 모델

모든 쓰레드가 전역 메모리공간에 공유하는 방식으로 프로그래머는 전역 메모리 공간에 대한 Access 동기화에 신경을 써야 한다.

pthread_create : 쓰레드 생성을 위해서 사용
pthread_create(&snd_thread, NULL, send_msg, (void *)&sock); //송수신 분리위해 스레드
pthread_create(&rcv_thread, NULL, recv_msg, (void *)&sock);

pthread_join : pthread 가 종료될떄까지 기다리다가 특정 pthread 가 종료시 자원 해제시킴
pthread_join(snd_thread, &thread_ret); //스레드 조인하는순간 구동
pthread_join(rcv_thread, &thread_ret);

pthread_detach : th_id 식별자를 가지는 pthread 가 부모 pthread 부터 독립할수 있게 한다
pthread_detach(t_id); // 떼어내는거 thread id 값

pthread_exit : 현재 실행중인 pthread 를 종료 시킬때 사용

pthread_cleanup_push : pthread_exit()가 호출될 때 호출된 handler 를 정하는 함수.
보통 자원 해제용이나 mutex lock 해제를 위한 용도로 사용된다.

pthread_cleanup_pop : 설정된 cleanup handler 를 제거하기 위해서 사용되는 함수.
exec 값을 0 일 경우 바로 cleanup handler 제거하고
그외의 값을 가질 경우 cleanup handler 를 한번 실행한 후 제거한다

이외에

쓰레드 동기화 함수

pthread_mutex_t mtx;

pthread_mutex_init

pthread_mutex_destroy

pthread_mutex_lock // pthread_mutex_lock(&mtx);

실제코드에서 락을 걸었다가 풀었다가 함 (크리티컬 섹션 보호)

뒤죽박죽 꼬이지않게 하기위해서

void *clnt_handler(void *arg) //

{

```
int clnt_sock = *((int *)arg);
int str_len = 0, i;
char msg[BUF_SIZE] = {0};
char pattern[BUF_SIZE] = "Input Number: \n";

signal(SIGALRM, sig_handler);  // 시그 알람 등록( 3 초내에
pthread_mutex_lock(&mtx); //다시 락을 걸어줌
thread_pid[idx++] = getpid();          //pid 값에 get pid 값을 넣음
i = idx - 1;
printf("i = %d\n", i);
write(clnt_socks[i], pattern, strlen(pattern));  // cnlt_socks 첫번째 클라이언트에 패턴을 써주겟다
pthread_mutex_unlock(&mtx);  //뒤죽박죽 꼬이지않게 락을 걸어줌 (크리티컬섹션 보호 )

alarm(3);  //3 초 대기

while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0)  //이제 리드하기 시작  패턴은 숫자입력하시오 리드는
{
        alarm(0);
        proc_msg(msg, str_len, i);              //수신한건 msg 에 들어감
        alarm(3); //위의 프록엠에스지가 끝나서 알람 3
}

pthread_mutex_lock(&mtx);

for(i = 0; i < clnt_cnt; i++)
{
        if(clnt_sock == clnt_socks[i])
        {
                while(i++ < clnt_cnt - 1)
```

```
                clnt_socks[i] = clnt_socks[i + 1];
            break;
        }
    }
```

pthread_mutex_unlock pthread_mutex_unlock(&mtx);

pthread_cond_init

pthread_cond_signal

pthread_cond_boradcast

pthread_cond_wait

pthread_cond_timewait

pthread_cond_destroy

쓰레드 Attribute 함수

pthread_attr_init

pthread_attr_distroy

pthread_attr_getscope

pthread_attr_setscope

pthread_attr_getdetachstate

pthread_attr_setdetachstate

쓰레드 시그널 관련 함수

pthread_sigmask

pthread_kill

sigwait

쓰레드 취소 함수

pthread_cancel

pthread_setcancelstate

pthread_setcancelstate

pthread_setcanceltype

pthread_testcance

gserv.c 메인함수에서 실제 동작을 하는 부분임

```
for(;;)                 //여기서 부터가 실제
{
    addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);

    thread_pid[idx++] = getpid();

    pthread_mutex_lock(&mtx);  //락은 거는 이유는 ?( 이순간부터 락이 걸림 다른애는 접근하지마 )
    data[clnt_cnt] = rand() % 3333 + 1;
    clnt_socks[clnt_cnt++] = clnt_sock;
    pthread_mutex_unlock(&mtx);

    pthread_create(&t_id, NULL, clnt_handler, (void *)&clnt_sock);          //피스레드를 만듬
                clnt_handler 은
    pthread_detach(t_id); // 떼어내는거 thread id 값
    printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
}

close(serv_sock);
```

gclnt.c 에서 메인함수는 다른 클라이언트와 마찬가지고 거의 패턴형식으로 진행됨

```
int main(int argc, char **argv)
{
    int sock;
    si serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)
        err_handler("socket() error");
```

```
	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, (sp)&serv_addr, sizeof(serv_addr)) == -1)	//거의 패턴임
		err_handler("connect() error");

	pthread_create(&snd_thread, NULL, send_msg, (void *)&sock);		//송수신 분리위해 스레
드
	pthread_create(&rcv_thread, NULL, recv_msg, (void *)&sock);	// snd , rc0v 송 수신  (create
순간 thread 만듬)
	pthread_join(snd_thread, &thread_ret);		//스레드 조인하는순간 구동
	pthread_join(rcv_thread, &thread_ret);

	close(sock);
```

이 함수는 함수의 이름처럼 서버로부터 들어오는 정보를 수신

```
void *recv_msg(void *arg)
{
	int sock = *((int *)arg);
	char msg[BUF_SIZE];
	int str_len;

	for(;;)		//실제알고리즘 서버로 부터 들어오는 정보를 수신
	{
		str_len = read(sock, msg, BUF_SIZE - 1);		//read 를 write 로 해도됨

		msg[str_len] = 0;
		fputs(msg, stdout);
	}

	return NULL;
```

리비스 함수와 샌드의 함수 두개를 만듬으로 송수신을 분리함

```
void *send_msg(void *arg)
{
	int sock = *((int *)arg);		//인자를 많이 받을려면  type 이 다르면 구조체로
	char msg[BUF_SIZE];		//
```

```
    for(;;)                 //무한루프
    {
        fgets(msg, BUF_SIZE, stdin);          //stdin 입력을받겟따  msg 에 저장
        write(sock, msg, strlen(msg));        //sock 에다가  write 를 해주고 있음 (서버로 전송이됨)
    }

    return NULL;
}
```

2. 네트워 플밍 기본기

네트워크 프로그래밍 은 두 디바이스 (컴퓨터 or 기기등등)간에 데이터를 서로 주고 받을수 있도록 프로그래밍 하는것이다

그러기 위해서 socket(소켓)이라는 매개체가 필요하다 두 디바이스 간에 데이터를 주고 받으려면 물리적 연결이 필요하며 이러한 연결을 도와주고
소켓을 통해 연결된 두 디바이스는 서로 데이터를 주고 받을 수 있다
모든 데이터 송수신 작업이 끝나고 소켓을 닫으면 두 디바이스간의 연결은 사라진다.

소켓의 연결과정에서
서버와 클라이언트가 언제나 짝을 이루어 만들어져야 한다

소켓을 생성하고

소켓에 주소정보를 할당한다 (bind)

연결수신 대기상태 (클라이언트가 연결을 시도하면 받을수 있도록 소켓을 listen socket 로 설정)
클라이언트 부분에서는 connect 함수를 사용 하여 서버의 소켓이 가지고 있는 IP, PORT 를 입력한다

연결 수락
클라이언트가 연결을 요청하면 서버에서는 수락을 할수 있다 여기서 수락을 하는 함수명은 accept 이며 수락과 동시에 또하나의 socket 가 생성된다. listen 으로 설정된 socket 은 다른 client 의 연결을 받기위해 계속 대기해야 하기 때문

이렇게 되면 연결이 완료 되어 송수신 할수있음

파일을 송수신 하는
서버측 코드이다

```
int main(int argc, char **argv)
{
	int serv_sock, clnt_sock, fd;
	char buf[BUF_SIZE] = {0};
	int read_cnt;

	si serv_addr, clnt_addr;
	socklen_t clnt_addr_size;

	if(argc !=2)
	{
		printf("use: %s <port>\n", argv[0]);
		exit(1);
	}

	fd = open("file server.c", O_RDONLY);    //이걸 생성함   이걸 바꾸면 다른거
	serv_sock = socket(PF_INET, SOCK_STREAM, 0);

	if(serv_sock == -1)
		err_handler("sockat() error");

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

	if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
	   err_handler("bind() error");

      if(listen(serv_sock, 5) == -1)
	   err_handler("listen() error");
	clnt_addr_size = sizeof(clnt_addr);

	clnt_sock =accept(serv_sock, (sap)&clnt_addr, &clnt_addr_size);

	for(;;)
	{
		read_cnt = read(fd, buf, BUF_SIZE);

		if(read_cnt < BUF_SIZE);
		{
```

```
                write(clnt_sock, buf, read_cnt);
                break;
            }

            write(clnt_sock, buf, BUF_SIZE);
        }

        shutdown(clnt_sock, SHUT_WR);
        read(clnt_sock, buf, BUF_SIZE);
        printf("msh from clin: %s\n", buf);

        close(fd);
        close(clnt_sock);
        close(serv_sock);

        return 0;
```

아래는 클라이언트 부분으로 거의 클라이언트부분은 비슷한 패턴을 보인다
서버부분에서 모든 작업이 이루어 지게 해야된다는걸 알수있다

```
void err_handler(char *msg)
{
        fputs(msg, stderr);
        fputs('\n', stderr);
        exit(1);
}

int main(int argc, char **argv)
{
        char buf[BUF_SIZE] = {0};
        int fd, sock, read_cnt;
        si serv_addr;


        if(argc !=3)
        {
            printf("use: %s <IP><port>\n", argv[0]);
            exif(1);
        }

        fd =open("receve.txt", O_CREAT | O_WRONLY);
        sock = socket(PF_INET, SOCK_STREAM, 0);

        if(sock == -1)
            err_handler("socket() error");

        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, (sap)&serv_addr, sizeof(serv_addr)) == -1)      //거의 패턴임
    err_handler("connect() error");
else
          puts("Connected . .....");
while((read_cnt = read(sock, buf, BUF_SIZE)) !=0)
          write(fd, buf, read_cnt);

puts("RECEIVER FILE Date");
write(sock, "thank you", 10);
close(fd);
close(sock);

return 0;
```