# Thread

- Process - fork() 복사: code, data, heap, stack
- Thread - pthread_create 복사: stack 공유: code, data, heap, stack

## pthread_create

```
// attr: thread 속성값을 저장하고 있는 구조체
// star_routine: thread와 함께 호출되는 함수 포인터
// arg: star_routine에 전해지는 매개변수
pthread_create(threadID, attr, star_routine, arg)
```

컴파일시 -lpthread 사용해야함

▶ 실습 01: thread_1.c

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

void *init_thread(void *parm)
{
    int i;
    for(i = 0; i < 10; i++)
    {
        printf("Counter: %d \n", i);
        sleep(1);
    }
    printf("Thread is now terminated. \n");
}

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

    //thread id, attr, 함수, 아규먼트
    if(pthread_create(&thread_id, NULL, init_thread, NULL) != 0)
    {
        fprintf(stderr,"PThread Creation Error \n");
        exit(0);
    }

    sleep(5);
    printf("Main function is terminated.\n");
    return 0;
}
```



main thread가 종료되어 count 끝까지 못함

## pthread_join

```
//thread ID, thread가 종료하면서 반환하는 값에 접근할 수 있는 더블포인터
phtread_join(thread, retval)
```

다른 thread 실행 끝날 때까지 main 종료하지 않음

실습 thread_01에 join 추가하면 정상작동 후 종료

### pthread_detach

해당 스레드 종료 시점에서 자동으로 운영체제에게 자원 반납

▶ 실습 05: multithreadtcpsrv.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/wait.h>
#include <pthread.h>

void * client_module(void * data)
{
    char rBuff[BUFSIZ];
    int readLen;
    int connectSd;
    connectSd = *((int *) data);
    while(1)
    {
        //읽기
        readLen = read(connectSd, rBuff, sizeof(rBuff)-1);
        if(readLen <= 0) break;
        rBuff[readLen] = '\0';
        printf("Client(%d): %s\n",connectSd,rBuff);
        //쓰기
        write(connectSd, rBuff, strlen(rBuff));
    }
    fprintf(stderr,"The client is disconnected.\n");
    close(connectSd);
}

int main(int argc, char** argv)
{
    int listenSd, connectSd;
    struct sockaddr_in srvAddr, clntAddr;
    int clntAddrLen, strLen;

    struct sigaction act;

    //스레드 생성
    pthread_t thread;

    if(argc != 2)
```

```
42      {
43          printf("Usage: %s [Port Number]\n", argv[0]);
44          return -1;
45      }
46
47      printf("Server start...\n");
48      listenSd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
49
50      memset(&srvAddr, 0, sizeof(srvAddr));
51      srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
52      srvAddr.sin_family = AF_INET;
53      srvAddr.sin_port = htons(atoi(argv[1]));
54
55      //bind -> 포트 할당
56      bind(listenSd, (struct sockaddr *) &srvAddr, sizeof(srvAddr));
57      //client 기다리기
58      listen(listenSd, 5);
59
60      clntAddrLen = sizeof(clntAddr);
61      while(1)
62      {
63          //클라이언트 연결함
64          connectSd = accept(listenSd,
65                     (struct sockaddr *) &clntAddr, &clntAddrLen);
66          if(connectSd == -1)
67          {
68              continue;
69          }
70          else
71          {
72              printf("A client is connected...\n");
73          }
74
75          //스레드 생성해서 client 처리
76          pthread_create(&thread, NULL, client_module, (void *) &connectSd);
77          pthread_detach(thread);
78      }
79      close(listenSd);
80      return 0;
81  }
82
                                                    Colored by Color Scripter
```

## Semaphore

파일 형태로 공동변수 관리 공유자원 read는 상관없지만 T1이 write하고 있을 때 T2가 read해버리면 동기화안됨

### sem_open

세마포어 시작

```
sem_open(name, oflag, mode(권한 정보 설정), value(세마포어 초기값))
```

**sem_wait**

세마포어 값을 확인하고 해당 값이 0인 경우 그값이 양수가 될때까지 보호되는 영역 (critical section)에 대한 접근을 대기상태로 전환함

```
sem_wait(sem(critical section의 세마포어를 가리킴))
```

**sem_post**

작업 끝내고 다른 thread로 작업할 수 있도록 함 --> 세마포어 값을 양수로 만들어 다른 스레드도 공유자원 사용할 수 있도록 함

```
sem_post(sem);
```

**sem_unlink**

name의 세마포어 제거함

```
sem_unlink(name)
```

▶ 실습 03: thread_03.c

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <fcntl.h>

#define SEM_NAME "/Test"

void *threadA_main(void *arg);
void *threadB_main(void *arg);

static sem_t * sem;
static int counter = 0;

int main(int argc, char **argv)
{
    //thread 생성
    pthread_t thread_id_1, thread_id_2;
    int res;

    //세마포어 시작
    sem = sem_open(SEM_NAME, O_RDWR | O_CREAT, 0777, 1);

    if(sem == SEM_FAILED)
    {
        fprintf(stderr, "Sem_Open Error \n");
```

```
27          exit(1);
28      }
29
30      //스레드 A 시작
31      if(pthread_create(&thread_id_1, NULL, threadA_main, NULL) != 0)
32      {
33          fprintf(stderr,"PThread 1 Creation Error \n");
34          exit(0);
35      }
36
37      //스레드 B 시작
38      if(pthread_create(&thread_id_2, NULL, threadB_main, NULL) != 0)
39      {
40          fprintf(stderr,"PThread 2 Creation Error \n");
41          exit(0);
42      }
43
44      //스레드 A 끝날 때까지 기다리기
45      if(pthread_join(thread_id_1, (void **) &res) != 0)
46      {
47          fprintf(stderr,"PThread 1 Join Error \n");
48          exit(0);
49      }
50
51      //스레드 B 끝날 때까지 기다리기
52      if(pthread_join(thread_id_2, (void **) &res) != 0)
53      {
54          fprintf(stderr,"PThread 2 Join Error \n");
55          exit(0);
56      }
57
58      //세마포어 닫기
59      sem_unlink(SEM_NAME);
60      return 0;
61  }
62
63  void *threadA_main(void *arg)
64  {
65      int i;
66      for(i=0; i < 10; i++)
67      {
68          sem_wait(sem);
69          counter += 2;
70          printf("Thread A increases the counter by 2: Counter - %d \n", counter);
71          sem_post(sem);
72      }
73      return NULL;
74  }
75
76  void *threadB_main(void *arg)
77  {
78      int i;
79      for(i=0; i < 10; i++)
80      {
81          sem_wait(sem);
82          counter += 3;
83          printf("Thread B increases the counter by 3: Counter - %d \n", counter);
```

```
        sem_post(sem);
    }
    return NULL;
}
```



## Mutex

- pthread_mutex_init 뮤텍스(공유자원) 초기화
- pthread_mutex_lock 뮤텍스 잠금 (상호배타)
- pthread_mutex_unlock 뮤텍스 풀기 (다른사람이 lock가능)
- pthread_mutex_destroy 뮤텍스 파괴

▶ 실습 04: thread_4.c

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

void *threadA_main(void *arg);
void *threadB_main(void *arg);

static int counter = 0;

//뮤텍스 객체
static pthread_mutex_t mutex;

int main(int argc, char **argv)
{
    pthread_t thread_id_1, thread_id_2;
    int res;

    //뮤텍스 초기화
    pthread_mutex_init(&mutex, NULL);

    //스레드 만들기
    pthread_create(&thread_id_1, NULL, threadA_main, NULL);
    pthread_create(&thread_id_2, NULL, threadB_main, NULL);

    //스레드 끝날때까지 기다리기
    pthread_join(thread_id_1, (void **) &res);
    pthread_join(thread_id_2, (void **) &res);

    //뮤텍스 파괴
    pthread_mutex_destroy(&mutex);
    return 0;
}

void *threadA_main(void *arg)
```

```
{
    int i;
    for(i=0; i < 20; i++)
    {
        //뮤텍스 lock
        pthread_mutex_lock(&mutex);

        //실행
        counter += 2;
        printf("Thread A increases the counter by 2: Counter - %d \n", counter);

        //뮤텍스 unlock
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

void *threadB_main(void *arg)
{
    int i;
    for(i=0; i < 20; i++)
    {
        pthread_mutex_lock(&mutex);
        counter += 3;
        printf("Thread B increases the counter by 3: Counter - %d \n", counter);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```



### Semaphore & Mutex

- Semaphore 순차적인 다중통신 지원 접근을 대기하는 것
- Mutex 상호배제 공유자원에 대한 lock, unlock

## Multiplexing

하나의 프로세스가 여러명의 클라이언트에게 서비스를 제공함

### select

- 수신한 데이터를 지니고 있는 socket이 존재하는가?
- 데이터의 전송이 가능한 socket은 무엇인가?
- 예외상황이 발생한 socket은 무엇인가?

1. fd 설정 (어떤 소켓 볼건지) 검사 범위 지정 타임아웃 설정
2. select 호출
3. event 발생시 호출결과 확인

**select: 연결된 socket에 읽을 데이터가 있는지 없는지 확인하는 작업**

```
//파일 디스크립터중 제일 높은 값, fd 집합(read/write/except)
//관찰 결과 발생할때까지 프로그램 블로킹상태(timeout)

select (nfd, readfd, writefd, exceptfd, timeout)
```

- FD_ZERO //fdset의 모든 비트를 0으로
- FD_SET //fd를 1로 설정
- FD_CLR //fd를 0으로
- FD_ISSET //fd 1이면 양수 fd 배열중 관심 있는 걸 fdset

| select(multiplexing) | Multi Process/thread |
|---|---|
| socket | socket |
| bind | bind |
| listen | listen |
| 관심 대상 (socket) 설정 | accept<br>→ fork/thread create |
| listen socket에서 읽을 데이터가 있는 경우, accept | |
| accept 호출로 생성된 socket을 관찰 대상으로 등록 | |
| 새롭게 등록된 socket에 데이터가 있을 경우 입출력 처리 | |

## Socket programming with select()

▶ 실습 06: selecttcpsrv.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <errno.h>
#include <arpa/inet.h>


int main(int argc, char** argv)
{
    int listenSd, connectSd;
    struct sockaddr_in srvAddr, clntAddr;
    int clntAddrLen, readLen, strLen;
    char rBuff[BUFSIZ];
```

```
19      int maxFd = 0;
20      //fd setting
21      fd_set defaultFds, rFds;
22      int res, i;
23
24      if(argc != 2)
25      {
26          printf("Usage: %s [Port Number]\n", argv[0]);
27          return -1;
28      }
29
30      printf("Server start...\n");
31      listenSd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
32      if(listenSd == -1 ) printf("socket error");
33
34      memset(&srvAddr, 0, sizeof(srvAddr));
35      srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
36      srvAddr.sin_family = AF_INET;
37      srvAddr.sin_port = htons(atoi(argv[1]));
38
39      if(bind(listenSd, (struct sockaddr *) &srvAddr,
40                  sizeof(srvAddr)) == -1)
41          printf("bind error\n");
42      if(listen(listenSd, 5) < 0) printf("listen error\n");
43
44      FD_ZERO(&defaultFds);
45      FD_SET(listenSd, &defaultFds);
46      maxFd = listenSd;
47      //maxfd = 3, listense = 3
48
49      clntAddrLen = sizeof(clntAddr);
50      while(1)
51      {
52          //호출 전 복사 (원본 변경 X)
53          rFds = defaultFds;
54          printf("Monitoring ... \n");
55          //client 요청 받기 (새로운 클라이언트가 접속하거나 / 기존의 클라이언트의 요청)
56          if((res = select(maxFd + 1, &rFds, 0, 0, NULL)) == -1) break;
57          //현재 연결된 fd for문 돌리기
58          for(i=0; i<maxFd+1; i++)
59          {
60              //해당 fd가 관심있는 fd인가? 1이면 true
61              //요청이 온 fd를 처리하기
62              if(FD_ISSET(i, &rFds))
63              {
64                  //관심있는 소켓이 리슨 소켓이면
65                  //
66                  if(i == listenSd)
67                  {
68                      //accept new client
69                      connectSd = accept(listenSd,
70                           (struct sockaddr *) &clntAddr,
71                           &clntAddrLen);
72                      //return client's fd
73                      if(connectSd == -1)
74                      {
75                          printf("Accept Error");
```

```
                        continue;
                    }
                    printf("A client is connected...\n");

                    //해당 fd를 1로 설정(관심 있음)
                    FD_SET(connectSd, &defaultFds);
                    //max면 변환
                    if(maxFd < connectSd){
                        maxFd = connectSd;
                    }
                }
                else // client request의 경우
                {
                    //요청 읽기
                    readLen = read(i, rBuff, sizeof(rBuff)-1);
                    if(readLen <= 0)
                    {
                        printf("A client is disconnected...\n");
                        FD_CLR(i, &defaultFds);
                        close(i);
                        continue;
                    }
                    rBuff[readLen] = '\0';
                    printf("Client(%d): %s\n",i-3,rBuff);
                    //그대로 보내기
                    write(i,rBuff, strlen(rBuff));
                }
            }
        }
    }
    close(listenSd);
    return 0;
}


                                                    Colored by Color Scripter
```

select를 이용해 multi thread/process 이용하지 않아도 다중 통신이 구현 가능하다.

select를 이용해 client 요청이 올 때, 해당 client가 기존 client인지 새로운 client인지 max로 구분한다. 새로운 클라이언트라면 fd집합에 포함시키고 관심소켓으로 설정. 기존의 클라이언트면 메세지를 보냈음으로 read/write로 처리함