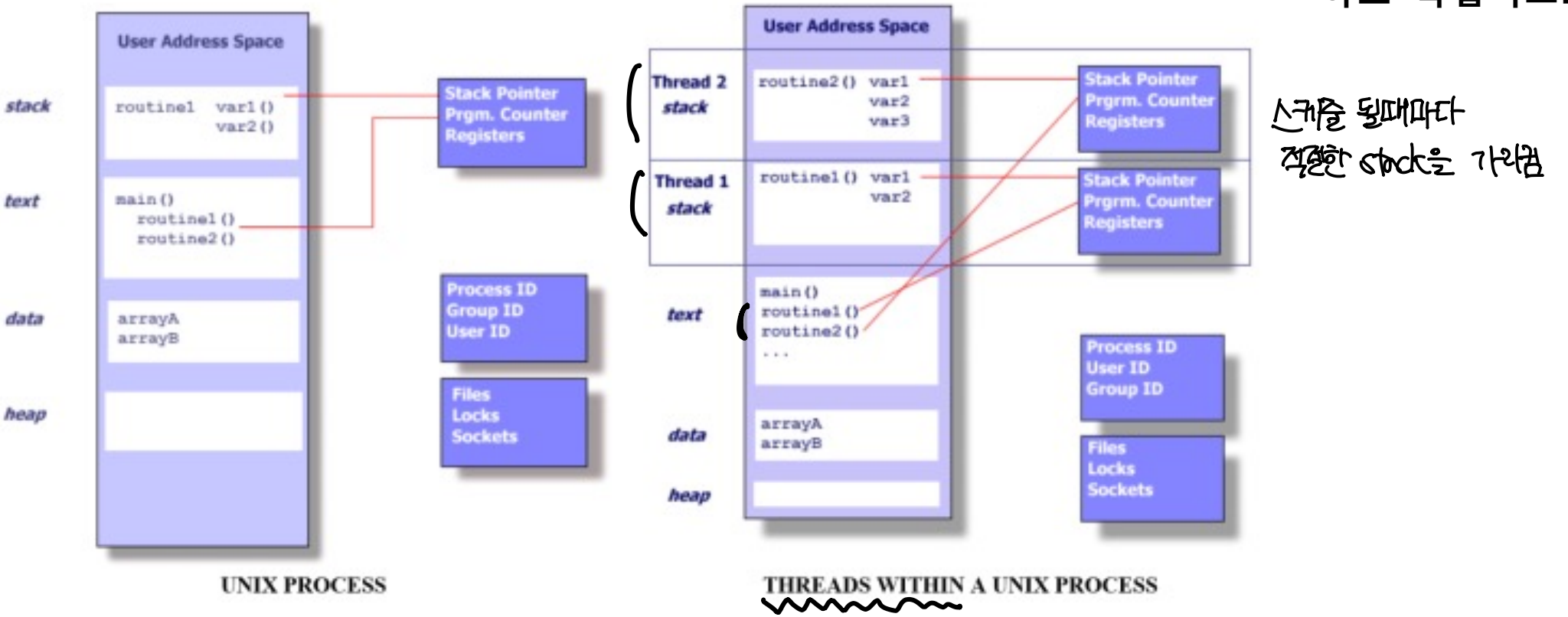
pthread Lib.

쓰레드의 개념

- 하나의 프로세스에 여러 개의 제어 흐름을 두어 프로세스가 동시에 여러 개의 일을 하도록 하는 것
- 쓰레드는 운영체제 커널에 의해 스케쥴 가능한 독립적인 명령어로 정의
- 소프트웨어 개발자에게는 쓰레드는 main 함수와 독립적으로 프로시저(procedure)의 개념



스케줄 될때마다 적절한 stack을 가리킴

Pthread Lib.

- 전통적으로 하드웨어 회사들의 다양한 쓰레드를 별도로 개발했음
- 프로그래머들이 동일한 인터페이스로 프로그래밍할 수 있게 표준화한 것이 **Pthread** 라이브러리임
- 커널 쓰레들의 장점을 충분히 활용하기 위해 프로그래머에게 표준화된 인터페이스를 제공함
- **Unix** 시스템에서는 **IEEE POSIX 1003.1c** 표준**(1995)**으로 **Pthread**가 개발됨
- **Pthreads**는 **C** 프로그래밍 언어의 형태와 프로시져 형태를 그대로 사용
- 참고 사이트
  - standards.ieee.org/findstds/standard/1003.1-2008.html
  - www.opengroup.org/austin/papers/posix_faq.html
  - www.unix.org/version3/ieee_std.html

Pthread Lib.를 이용

- 쓰레드 접근 방식의 장점
- 비동기적인 사건들을 처리하는 코드를 단순화할 수 있음
- 다중 프로세스 접근 방식에서는 프로세스들이 메모리나 파일 디스크립터를 공유하려면 운영체제가 제공하는 복잡한 기법들을 사용해야 하지만, 쓰레드들은 특별한 처리 없이도 동일한 메모리 주소 공간과 파일 디스크립터들에 접근할 수 있음
- 프로그램이 해결해야 할 문제를 작은 조각들로 분할함으로써 프로그램의 전반적인 산출량을 높일 수 있음
- 대화식 프로그램의 경우에는 사용자의 입·출력을 처리하는 부분과 그 외의 부분을 여러 쓰레드들로 나눔으로써 프로그램의 체감 반응 속도를 높일 수 있음
- 모든 쓰레드는 각자 고유한 쓰레드 **ID**를 가짐
- 프로세스 **ID**가 시스템 전반에서 고유한 것과 달리, 쓰레드 **ID**는 쓰레드가 속한 프로세스의 문맥 안에서만 의미를 가짐
- 컴파일 시 **POSIX Thread** 라이브러리를 링크해야 사용 가능
  - gcc –o hello hello.c -lpthread

1. 쓰레드 생성

Thread ID가 저장되는 변수

```
#include <pthread.h>
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr,
                   void *(*start_rtn)(void *), void *restrict arg);
                                                반환값: 성공 시 0, 실패 시 오류 번호
```

쓰레드 생성 함수

어떤 작업을 수행할건지?

Attr – 쓰레드 특성. NULL 지정

pthread_attr_t 의 값은 pthread_attr_init() 이용해 지정 (동적 지정)

사용자 모드 쓰레드를 사용하기 위해서는 PTHREAD_SCOPE_SYSTEM을 쓰고 커널 모드 쓰레드를 사용하기 위해서는 PTHREAD_SCOPE_PROCESS 값을 사용.

리눅스는 /usr/include/bits/pthreadtypes.h

다중 인자 – 구조체로 //. 모든 인자는 주소로 전달(반드시 (void *)형 변환 필요)

쓰레드 생성 예

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


pthread_t tidnum;


void print_ids(const char *s) {
    pid_t       pid;
    pthread_t   tid;


    pid = getpid();
    tid = pthread_self();
    printf("%s pid %u tid %u (0x%x)\n", s, (unsigned int)pid,
      (unsigned int)tid, (unsigned int)tid);
}


void * thread_func(void *arg) {
    print_ids("new thread: ");
    return((void *)0);
}
```

→ thread ID 획득

```c
int main(void) {
    int     errnum;


    errnum = pthread_create(&tidnum, NULL, thread_func, NULL);
    if (errnum != 0) {
        printf("Can't create thread : %s\n", strerror(errnum));
        exit(1);
    }
    print_ids("Main Thread:");
    sleep(1);
    exit(0);
}
```

```
[ssuos@localhost os]$ gcc 11-3-1.c -lpthread
[ssuos@localhost os]$ ./a.out
Main Thread: pid 17530 tid 3078445312 (0xb77d5900)
new thread:  pid 17530 tid 3078441792 (0xb77d4b40)
```

↓

프로세스를 공유하므로

PID는 동일

쓰레드로 인자 전달

- **pthread_create()는 프로그래머에게 쓰레드 시작 루틴에 하나의 인자를 전달할 수 있게 함**
- **다중 인자를 전달하려면 구조체 형태로 전달**
- **모든 인자는 주소로 전달(반드시 (void *)형 변환 필요)**

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 8
char *messages[NUM_THREADS];
struct thread_data
{
  int thread_id;
  int  sum;
  char *message;
};

struct thread_data thread_data_array[NUM_THREADS];

void *PrintHello(void *threadarg)
{
  int taskid, sum;
  char *hello_msg;
  struct thread_data *my_data;


  sleep(1);
  my_data = (struct thread_data *) threadarg;
  taskid = my_data->thread_id;
  sum = my_data->sum;
  hello_msg = my_data->message;
  printf("Thread %d: %s  Sum=%d\n", taskid, hello_msg, sum);
  pthread_exit(NULL);
}
```

```
int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int *taskids[NUM_THREADS];
int rc, t, sum;

sum=0;
messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "Klingon: Nuq neH!";
messages[4] = "German: Guten Tag, Welt!";
messages[5] = "Russian: Zdravstvytye, mir!";
messages[6] = "Japan: Sekai e konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";
for(t=0;t<NUM_THREADS;t++) {
 sum = sum + t;
 thread_data_array[t].thread_id = t;
 thread_data_array[t].sum = sum;
 thread_data_array[t].message = messages[t];
 printf("Creating thread %d\n", t);
 rc = pthread_create(&threads[t], NULL, PrintHello, (void *)  &thread_data_array[t]);
 if (rc) {
   printf("ERROR; return code from pthread_create() is %d\n", rc);
   exit(-1);
   }
 }
pthread_exit(NULL);
}
```

쓰레드 종료

```
#include <pthread.h>
void pthread_exit(void *rval_ptr);
```

- **쓰레드를 종료시키는 방법은 여러 가지가 있음**
  - 쓰레드를 시작 루틴에서 아무 일 없이 일을 수행하면 종료 후 리턴
  - 쓰레드가 일을 완료하던지 안하던지 간에 pthread_exit() 호출하면 쓰레드는 종료
  - 전체 프로세스가 exec()이나 exit()을 호출하면 쓰레드는 종료
  - pthread_exit()을 명시적으로 호출하지 않고 main()가 종료되면 쓰레드는 종료
- **pthread_exit() 파일을 close x... open()** (파일을 자동으로 닫지 않는다)
- **Pthread_join().. 자원을 회수**

pthread_exit()을 호출해도

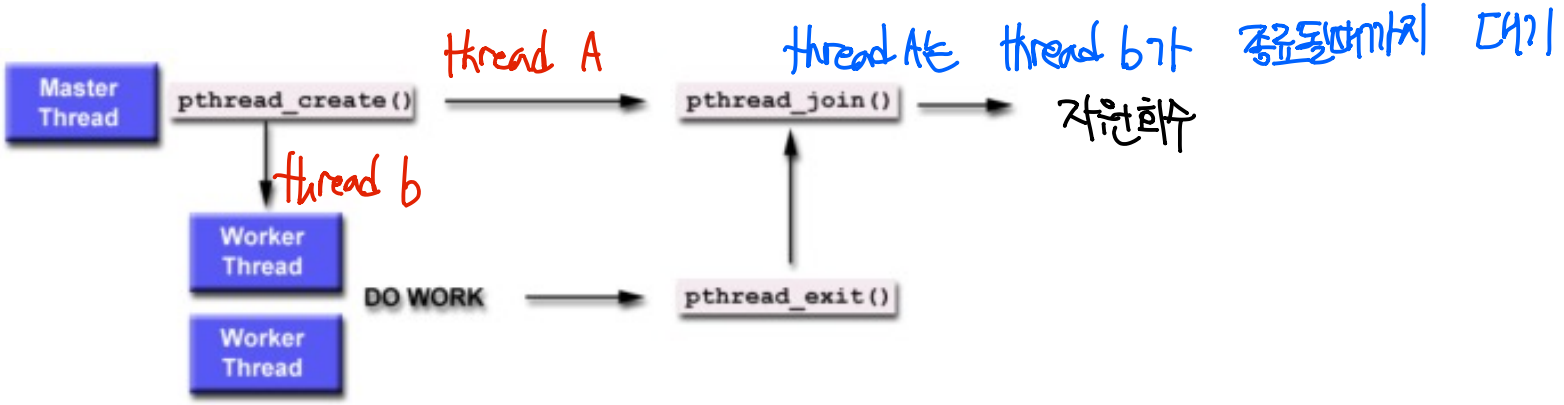
쓰레드 종료 대기

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **rval_ptr);
                                          반환값: 성공 시 0, 실패 시 오류 번호
```

프로세스가 종료를 하지 않고.. 쓰레드가 종료되는 상황

1. pthread_exit()
2. 같은 프로세스에서 다른 어떤 쓰레드가 취소요청에 의해 쓰레드가 취소됨.
3. 쓰레드 종료해야루틴이 정상적으로 return ...-> 쓰레드의 종료 코드...

pthread_join 함수를 호출한 쓰레드는 thread로 지정된 쓰레드가 pthread_exit 함수를 호출하거나 취소될 때까지 차단됨
쓰레드가 취소되면 rval_ptr가 가리키는 메모리 장소 PTHREAD_CANCELED가 설정됨



쓰레드 분리

```
#include <pthread.h>
int pthread_detach(pthread_t tid);
```

반환값: 성공 시 0, 실패 시 오류 번호

tid에 해당하는 thread를 main thread에서 완전히 분리

•pthread_detach()함수를 호출하는 외에도 pthread_create()시 pthread_attr_t에 detachstate를 지정해 줌으로써 detach상태로 생성 가능

•pthread_attr_t는 pthread_attr_init(3) 변경 가능

기타

- **다른 쓰레드의 취소를 요청하는 함수**

```
#include <pthread.h>
int pthread_cancel(pthread_t tid);
                                        반환값: 성공 시 0, 실패 시 오류 번호
```

tid에 해당하는 thread 취소

- **쓰레드 종료 시 호출되는 함수를 처리하는 함수**

```
#include <pthread.h>
void pthread_cleanup_push(void(*rtn)(void *), void *arg);
void pthread_cleanup_pop(int execute);
```

종료루틴등록

↳0이 아닌값을 넣어서 호출

- **두 쓰레드 ID를 비교하는 함수 (pthread_self() = getpid)**

```
#include <pthread.h>
int pthread_equal(pthread_t tid1, pthread_t tid2);
                                        반환값: 둘이 같으면 0이 아닌 값, 같지 않으면 0
```

- **최대 한 번만 수행됨을 보장하는 함수**

하나의 프로세스 내에서 하나의 쓰레드만 해당 루틴을 실행 가능

```
#include <pthread.h>
int pthread_once(pthread_once_t *once_control, void (*init_routine)(void));
반환값: 성공 시 0, 실패 시 오류 번호
```

Mutex Lock

- 쓰레드의 동기화와 다중 쓰기 작업이 있을 때 공유되는 데이터의 보호
- **mutex** 변수는 공유 데이터를 접근하는 것을 보호하는 **Lock**처럼 사용
- 주어진 시간에 하나의 쓰레드만이 **mutex** 변수를 **lock** 걸 수 있음 : 동시에 여러개의 쓰레드가 **mutex** 변수를 **lock**하려고 해도 단 하나의 쓰레드만 **lock**을 걸 수 있음
- 공유변수를 번갈아 가며 사용 - **race condition** 방지
- 은행 트랙잭션의 예

Thread 1	Thread 2	Balance
Read balance: $1000		$1000
	Read balance: $1000	$1000
	Deposit $200	$1000
Deposit $200		$1000
Update balance $1000+$200		$1200
	Update balance $1000+$200	$1200

Mutex Lock 필요 예

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


int cnt=0;


void *count(void *arg){
   int i;
   char* name = (char*)arg;

   //======== critical section ============
   cnt=0;
   for (i = 0; i <10; i++)
   {
      printf("%s cnt: %d\n", name,cnt);
      cnt++;
      usleep(1);
   }
   //========= critical section ============
}
```

임계영역
즉, Lock을 걸어주어야 한다..!

```
int main()
{
   pthread_t thread1,thread2;


   pthread_create(&thread1, NULL, count, (void *)"thread1");
   pthread_create(&thread2, NULL, count, (void *)"thread2");


   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);
}
```

2개의 thread

2개의 프로세스가 끝날때까지 대기

```
Thread 1, cnt 0
Thread 2, cnt 1
Thread 1, cnt 32
Thread 2, cnt 3....
```

Lock이 걸려있지 않아서 올바르지 않은 값이 나오게됨

```
Thread 1, cnt0
Therad1, cnt 1.....
Thread 2, cnt 0, cnt1, cnt2....
```

이상적인 결과

Mutex 관련 함수와 조건 변수

```
#include <pthread.h>
pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexatt_t *attr);
pthread_mutex_destroy (pthread_mutex_t *mutex);
반환값: 성공 시 0, 실패 시 오류 번호
```

Pthread_mutex_t … 형태의 변수 선언…
pthread_mutex_t mutex;

정적 방법 : 초
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cont_t mutex = PTHREAD_COND_INITIALIZER;

동적방법
pthread_mutex_init(); // ulock 상태로 초기화 된다.
Pthread_cond_init() ;

Mutex 예

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

pthread_mutex_t mutex;
int cnt=0;

void *count(void *arg){
   int i;
   char* name = (char*)arg;

   pthread_mutex_lock(&mutex);

   //======== critical section ============
   cnt=0;
   for (i = 0; i <10; i++)
   {
      printf("%s cnt: %d\n", name,cnt);
      cnt++;
      usleep(1);
   }
   //========= critical section ============
   pthread_mutex_unlock(&mutex);
}
```

임계영역

```
int main()
{
   pthread_t thread1,thread2;

   pthread_mutex_init(&mutex,NULL);

   pthread_create(&thread1, NULL, count, (void *)"thread1");
   pthread_create(&thread2, NULL, count, (void *)"thread2");

   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);

   pthread_mutex_destroy(&mutex);
}
```

⇒ 동적 초기화

Mutex ==== 조건 변수

- **조건변수는 쓰레드간 동기화를 위해서 사용하는 장치**
- **공유되는 데이터의 안정성 보장하기 위한 용도**
- **하나의 쓰레드는 조건변수에 시그널이 전달될 때까지 특정영역에서 대기 상태로**

Thread 1	Thread 2
i ++;	i 출력

- **조건 변수 사용**
  - 조건 변수 사용하지 않으면 프로그래머가 지속적으로 쓰레드를 지속적으로 polling하면서 조건이 만족하는지 검사해야 함
  - 조건 변수는 반드시 mutex lock과 함께 사용
- **조건변수 초기화 및 해제 함수**

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *restrict cond, pthread_condattr_t *restrict attr);
int pthread_cond_destroy(pthread_cond_t *cond);
                                        반환값(둘 다): 성공 시 0, 실패 시 오류 번호
```

조건 변수

- **쓰레드를 잠그는 함수**

```
#include <pthread.h>
int pthread_cond_wait(pthread_cond_t *restrict cond, pthread_mutex_t *Restrict mutex);
int pthread_cond_timedwait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex,
                        const struct timespec *restrict timeout);
                                        반환값(둘 다): 성공 시 0, 실패 시 오류 번호
```

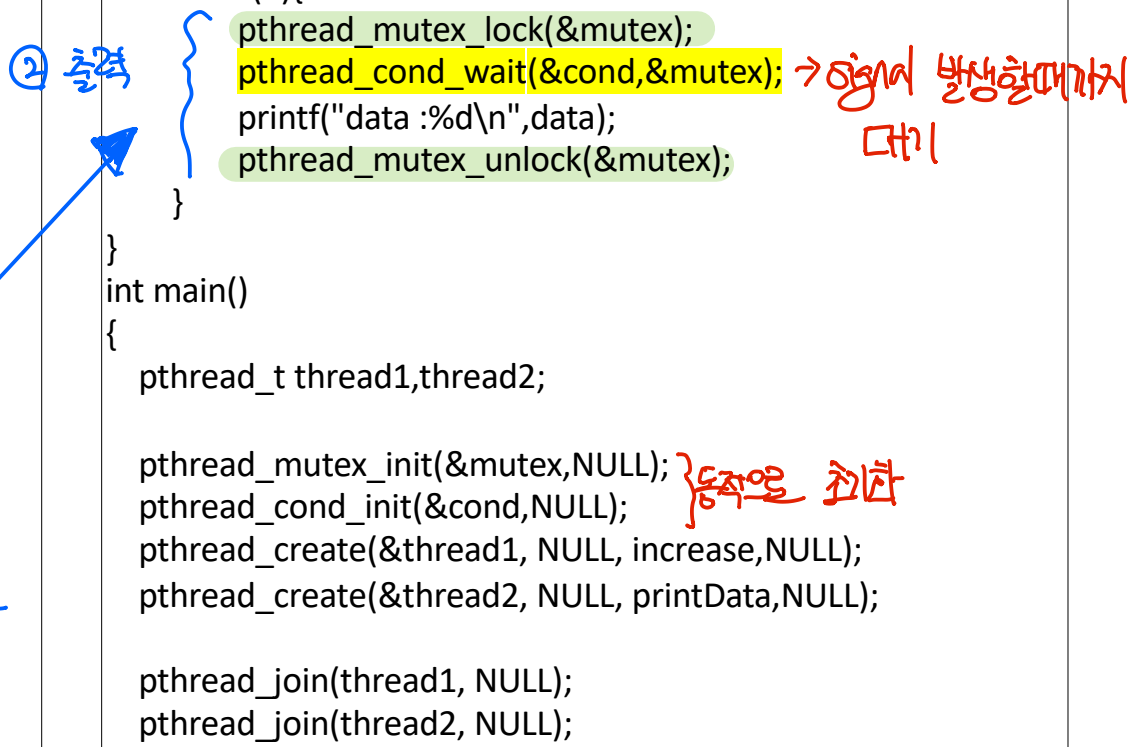
- **쓰레드를 깨우는 함수**

```
#include <pthread.h>
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
                                        반환값(둘 다): 성공 시 0, 실패 시 오류 번호
```

조건변수 사용 예

```c
#include <pthread.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>

pthread_mutex_t mutex;
pthread_cond_t cond;

int data=0;

void *increase(void *arg){
    while(1){
        pthread_mutex_lock(&mutex);
        pthread_cond_signal(&cond);
        data++;
        pthread_mutex_unlock(&mutex);
        sleep(1);
    }
}
```

```c
void *printData(void *arg){
    while(1){
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond,&mutex);
        printf("data :%d\n",data);
        pthread_mutex_unlock(&mutex);
    }
}
int main()
{
    pthread_t thread1,thread2;

    pthread_mutex_init(&mutex,NULL);
    pthread_cond_init(&cond,NULL);
    pthread_create(&thread1, NULL, increase,NULL);
    pthread_create(&thread2, NULL, printData,NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```



조건변수 사용 예

```
#include <pthread.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>

pthread_mutex_t mutex;
pthread_cond_t cond;

int data=0;

void *increase(void *arg){
      while(1){
            pthread_mutex_lock(&mutex);
            pthread_cond_signal(&cond);
            data++;
            pthread_mutex_unlock(&mutex);
            sleep(1);
      }
}
```

```
void *printData(void *arg){
      while(1){
            pthread_mutex_lock(&mutex);
            pthread_cond_wait(&cond,&mutex);
            printf("data :%d\n",data);
            pthread_mutex_unlock(&mutex);
      }
}
int main()
{
   pthread_t thread1,thread2;

   pthread_mutex_init(&mutex,NULL);
   pthread_cond_init(&cond,NULL);
   pthread_create(&thread1, NULL, increase,NULL);
   pthread_create(&thread2, NULL, printData,NULL);

   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);

   return 0;
}
```