리눅스 멀티 스레드 소켓 프로그래밍

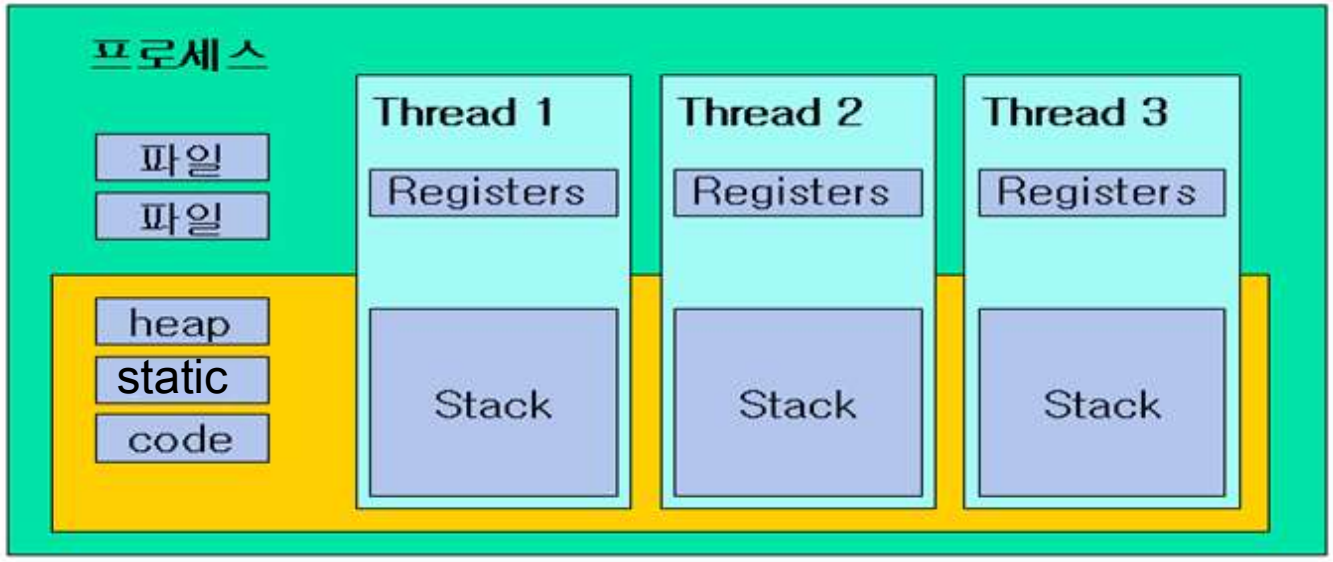
뇌를 자극하는 TCP/IP 소켓 프로그래밍

멀티 스레드 ?

- 최소 실행단위는 프로세스 ?
- 프로세스를 새로 생성하는 대신, 스레드(코드 흐름 단위, 문맥)를 만들고 이들 사이를 스위칭 한다.
- 스레드는 **Light Weight Process** 라고 부른다.

멀티 스레드 방식의 장점(over Multi-Processing)

- 빠르게 실행한다.
  - 프로세스 생성, 문맥교환에 드는 비용보다 스레드를 생성, 문맥교환하는 비용이 적다.
- 스레드간 데이터 교환이 효율적이다(IPC 에 비하여)
  - 별다른 **IPC**를 사용하지 않고도 스레드간에 **Code**, **Files(Sockets), memory data(Heap, Static)** 공유한다.
  - 각 스레드는 자기만의 **stack**과 **register**를 가지고 있다
- CPU를 잘 활용할 수 있다.

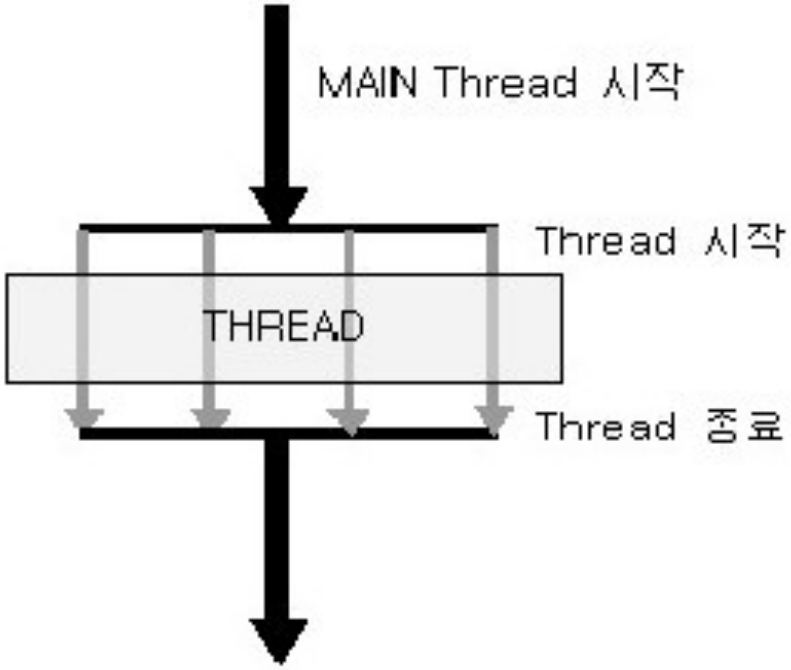


Main thread vs. Worker threads(Secondary threads)

멀티 스레드 방식의 단점

- 프로그래밍 난이도가 올라간다.
  - 직관적이지 않다.
  - 문맥의 흐름을 예상하기가 쉽지 않다.
- 불안하다.
  - 스레드에서 발생한 문제가 프로세스의 다른 스레드에 영향을 미친다.
- 디버깅이 어렵다.
- 제대로 만들기가 어렵다.
  - 병렬 프로그래밍, 공유 자원 관리는 높은 기술 숙련도를 요구한다.

스레드의 생성과 종료



Main thread

Worker threads
(Secondary threads)

POSIX Thread : pthread

- POSIX 표준을 따르는 스레드
  - POSIX Thread, pthread
  - Multi thread 프로그래밍을 위한 API를 제공

- 뛰어난 호환성
  - 윈도우를 제외한 모든 Unix계열 운영체제를 지원
  - CPU 각 벤더에서도 독자적인 thread API를 제공한다. pthread에 비해서 뛰어난 성능을 보여주지만, 이식성과 유지/보수 상의 문제로 대부분 pthread를 이용.

스레드 생성

- pthread_create

```
#include <pthread.h>

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

- 새로운 스레드를 생성하며, 새로 생성된 스레드는 다른 스레드와는 독립적으로 병렬로 실행된다

1. thread : 생성되는 스레드 식별 번호 반환을 위한 매개 변수
2. attr : 생성되는 스레드의 특성을 정하기 위해서 사용
3. **start_routine** : 스레드 함수의 포인터
4. **arg** : 스레드 함수로 넘길 매개 변수

스레드 대기 및 분리

- **pthread_join**(pthread_t th, void **thread_return);
  - 메인 스레드는 **worker** 스레드를 기다림
  - 스레드의 종료값을 받아오기 위함
  - 종료된 스레드 자원을 정리

- **pthread_detach**(pthread_t th);
  - 해당 worker 스레드를 메인 스레드로 부터 분리함
  - 분리된 스레드는 종료시 **pthread_join**을 기다리지 않고 모든 자원을 즉시 회수

- pthread_t **pthread_self** (void);
  - 해당 스레드의 식별자를 돌려준다

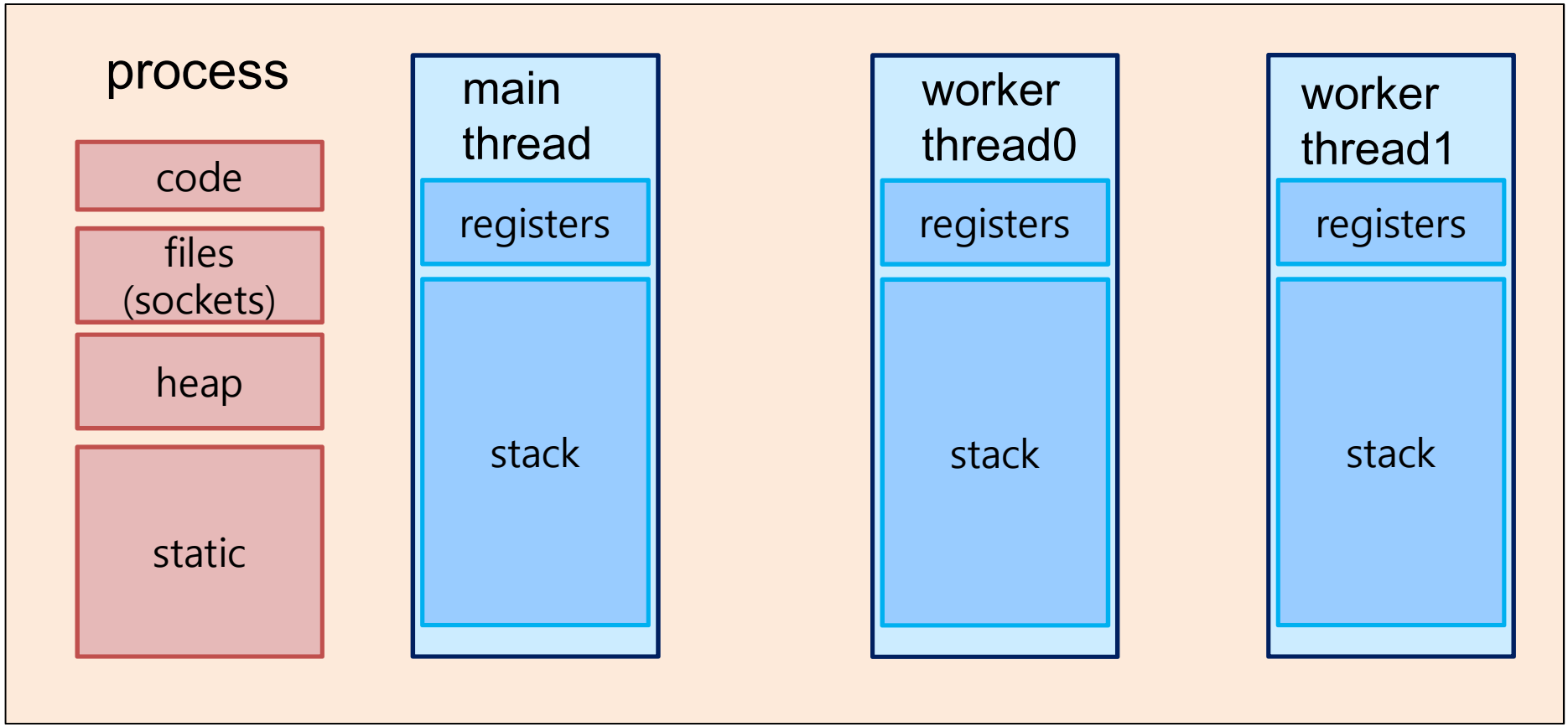
예제 프로그램 : count_thread.c

```
#define         MAX_THREAD      2

void *t_func(void *data)
{
    int *count = (int *)data;          // auto variable stored in stack
    int tmp;
    pthread_t thread_id = pthread_self();

    while(1)
    {
        printf("%lu %d\n", thread_id, *count);
        *count = *count+1;
        sleep(1);
    }
}
```

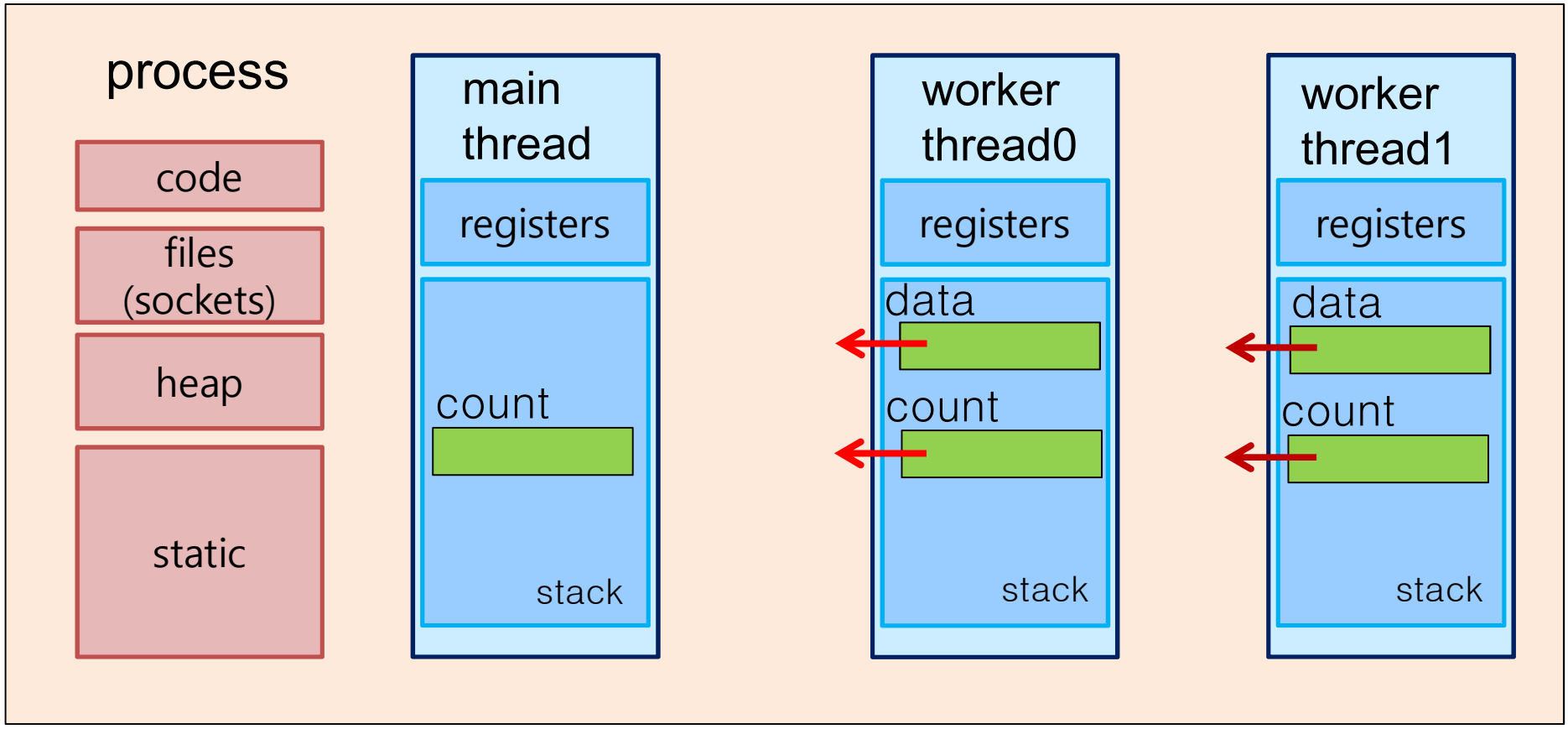
예제 프로그램 : count_thread.c

```
int main(int argc, char **argv)
{
	pthread_t thread_id[MAX_THREAD];
	int i = 0;	int count = 0;		// auto variables stored in stack

	for(i = 0; i < MAX_THREAD; i++) {
		pthread_create(&thread_id[i], NULL, t_func, (void *)&count);
		usleep(5000);
	}
	while(1) {
		printf("Main Thread : %d\n", count);
		sleep(2);
	}
	for(i = 0; i < MAX_THREAD; i++) {
		pthread_join(thread_id[i], NULL);
	}
	return 0;
}
```

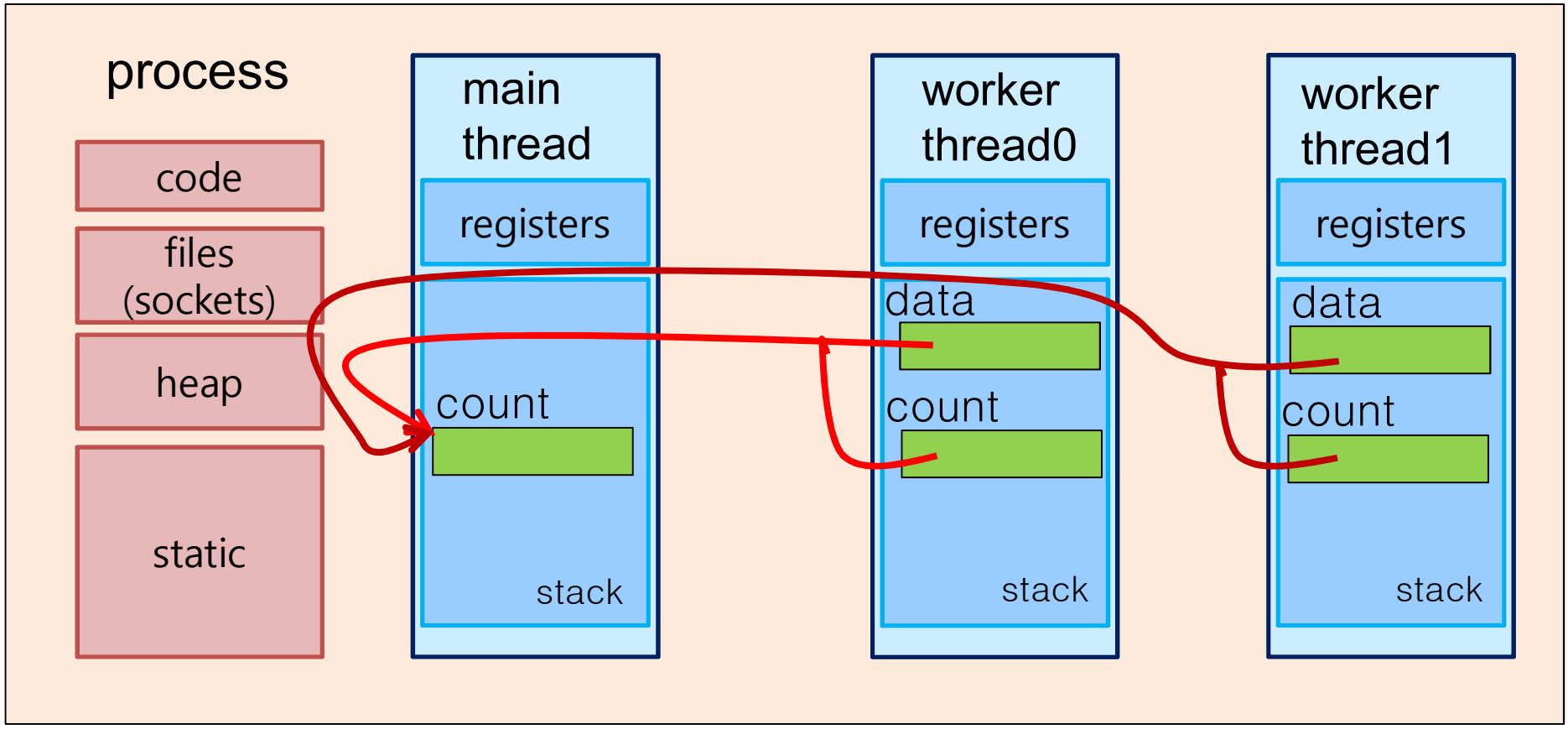
예제 프로그램 : count_thread.c



예제 프로그램 : count_thread.c



예제 프로그램 : count_thread.c



예제 프로그램 : count_thread.c

Lab

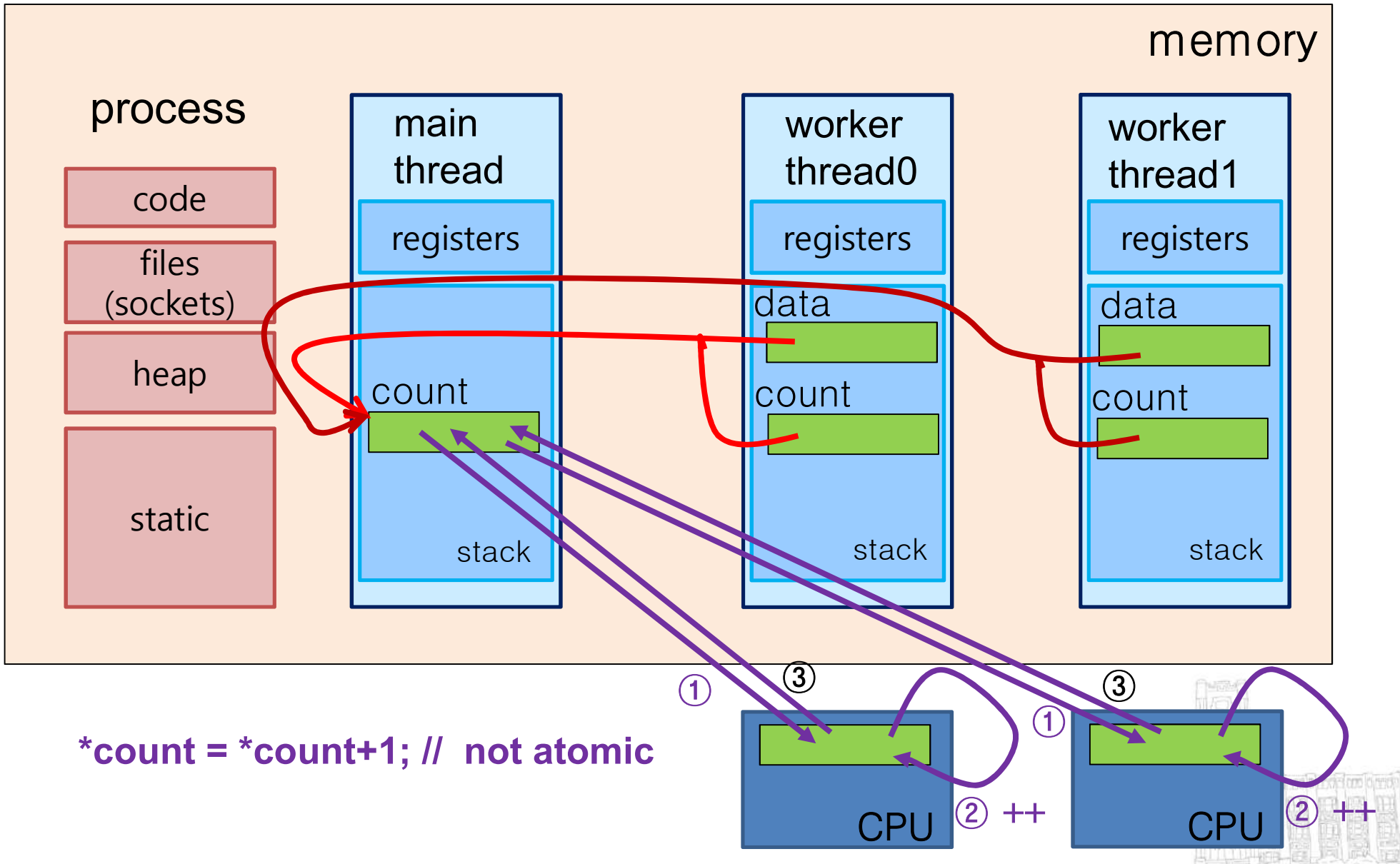
osnw00000000@osnw00000000-osnw: ~/lab08

```
osnw00000000@osnw00000000-osnw:~/lab08$ ./count_thread
140135318943296 0
140135310550592 1
Main Thread : 2
140135318943296 2
140135310550592 3
140135318943296 4
Main Thread : 5
140135310550592 5
140135318943296 6
140135310550592 7
140135318943296 8
Main Thread : 9
140135310550592 9
140135318943296 10
140135310550592 11
Main Thread : 12
140135318943296 12
140135310550592 13
140135318943296 14
140135310550592 15
Main Thread : 16
```

osnw00000000@osnw00000000-osnw: ~

```
osnw00000000@osnw00000000-osnw:~$ ps -eLf | grep count_thread
osnw000+  117863  117513  117863  0    3 02:53 pts/0    00:00:00 ./count_thread
osnw000+  117863  117513  117864  0    3 02:53 pts/0    00:00:00 ./count_thread
osnw000+  117863  117513  117865  0    3 02:53 pts/0    00:00:00 ./count_thread
osnw000+  117871  117571  117871  0    1 02:54 pts/1    00:00:00 grep --color=auto count_thread
```

예제 프로그램 : count_thread.c



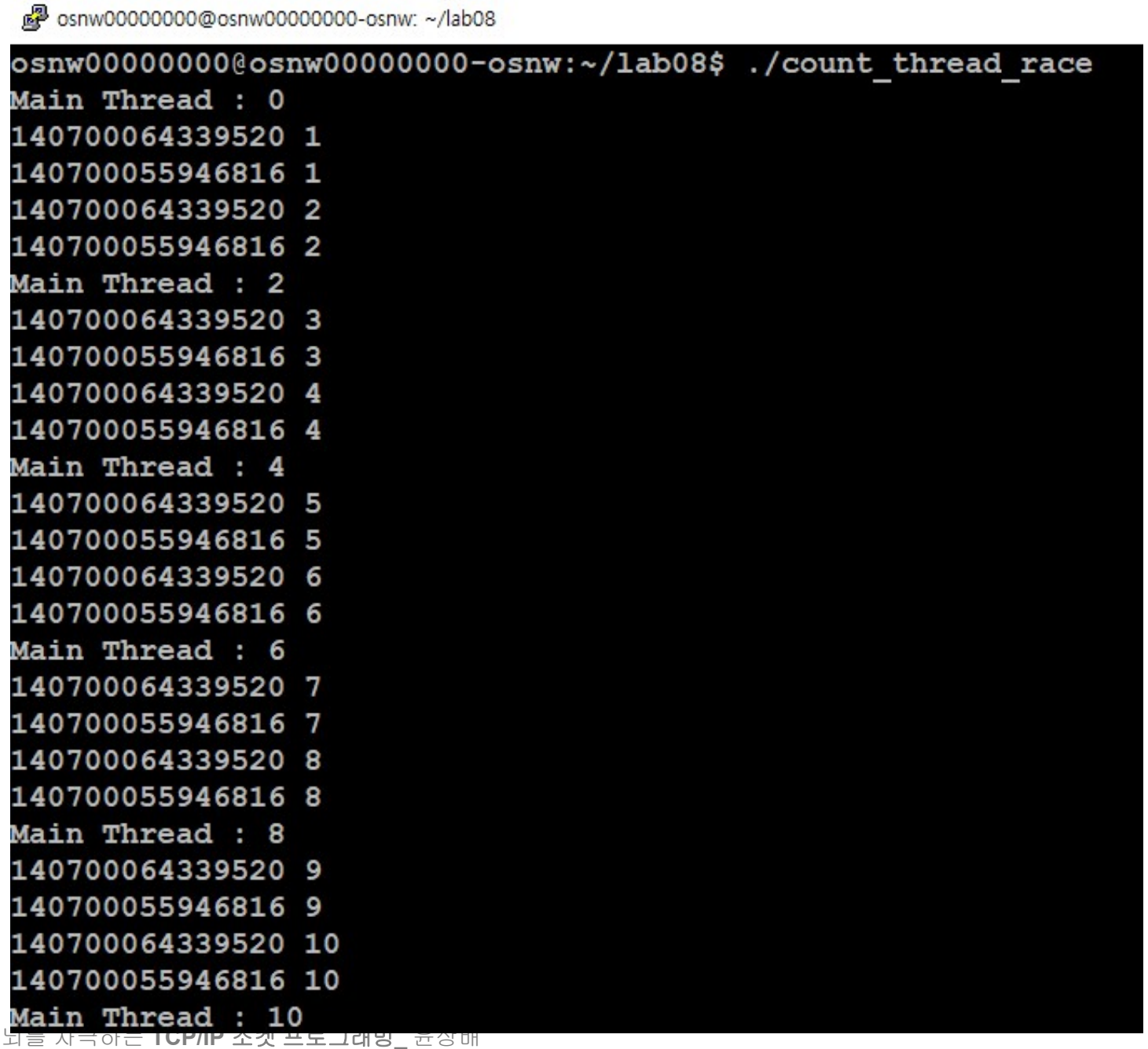
예제 프로그램 : count_thread_race.c

Race Condition 발생을 위한 프로그램 수정 **t_func()**

```
while(1)
{
        printf("%lu %d\n", thread_id, *count);
        *count = *count+1;              //  not atomic
        sleep(1);
}
// ================================➔
while(1)
{
        tmp = *count;
        tmp++;
        sleep(1);
        *count = tmp;
        printf("%lu %d\n", thread_id, *count);
}
```

예제 프로그램 : count_thread_race.c

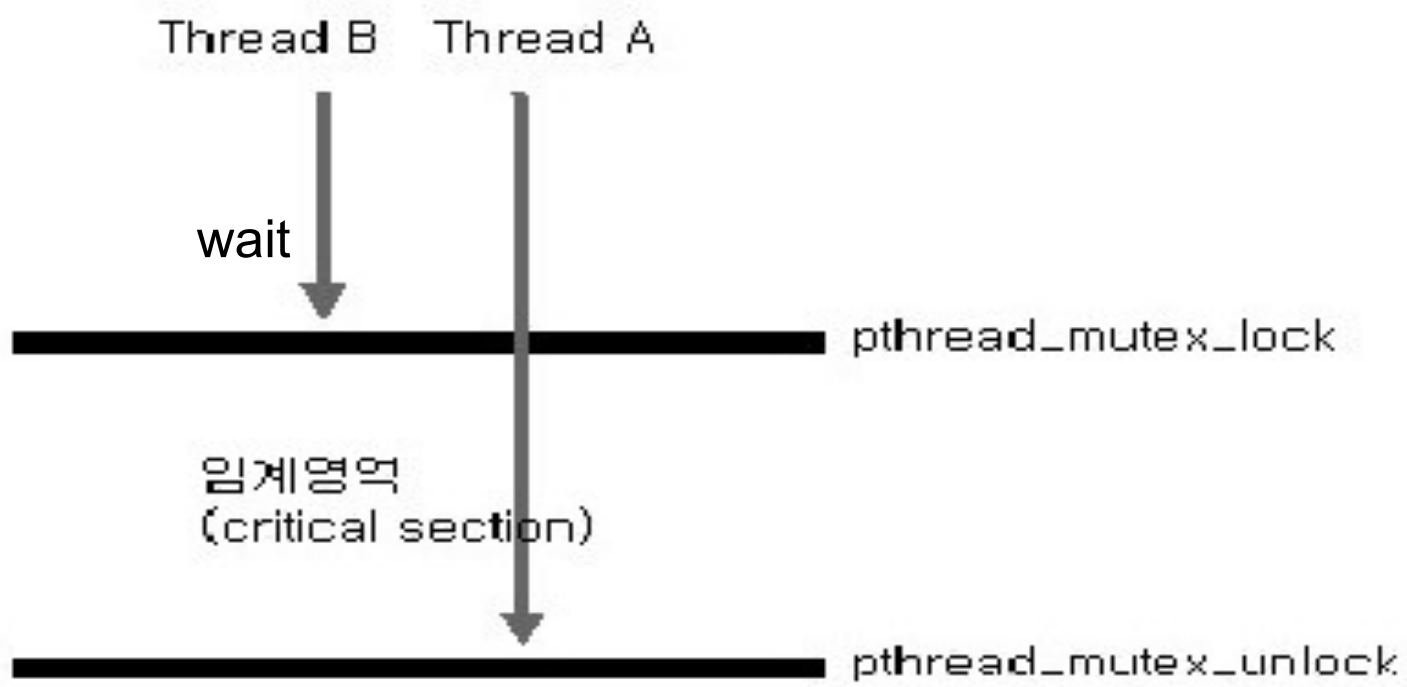
Lab



mutex를 이용한 공유자원 보호

- 하나의 자원을 여러 스레드가 접근할 경우
  - 한번에 하나의 스레드만 자원에 접근할 수 있도록 한다.

- 세마포어 : 커널자원으로 다른 프로세스간 사용 가능
- **mutex** : 프로세스 자원으로 단일 프로세스에서만 사용 가능
  - 임계영역 설정
  - mutex를 얻은 스레드만 임계영역에 진입 가능



mutex 사용하기

- **mutex 초기화**

```
int pthread_mutex_init(pthread_mutex_t * mutex,
        const pthread_mutex_attr *attr);
```

- **mutex 얻기(잠금)**

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

- **mutex 돌려주기(잠금해제)**

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- **mutex 파기**

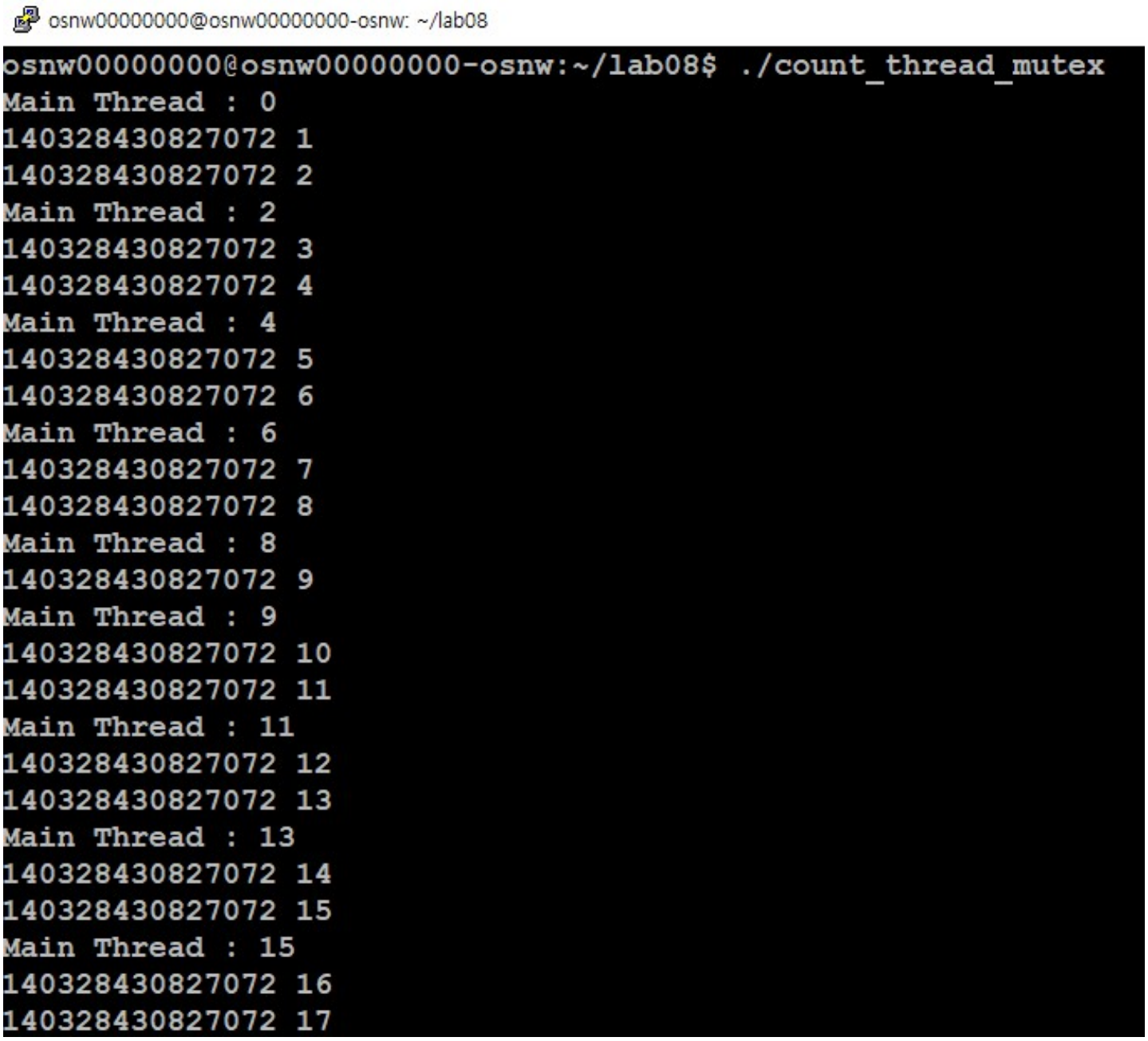
```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

공유자원 보호 예제: count_thread_mutex.c

```
pthread_mutex_t m_lock;                    // global variable

void *t_func(void *data)
{
      int *count = (int *)data;
      while(1)
      {
            pthread_mutex_lock(&m_lock);
                  tmp = *count;
                  tmp++;
                  sleep(1);
                  *count = tmp;
            pthread_mutex_unlock(&m_lock);
      }
}
int main(int argc, char **argv)
{     int count = 0;
      pthread_mutex_init(&m_lock, NULL);
      pthread_create(,,t_func,(void *)&count);
      pthread_join(...);
}
```

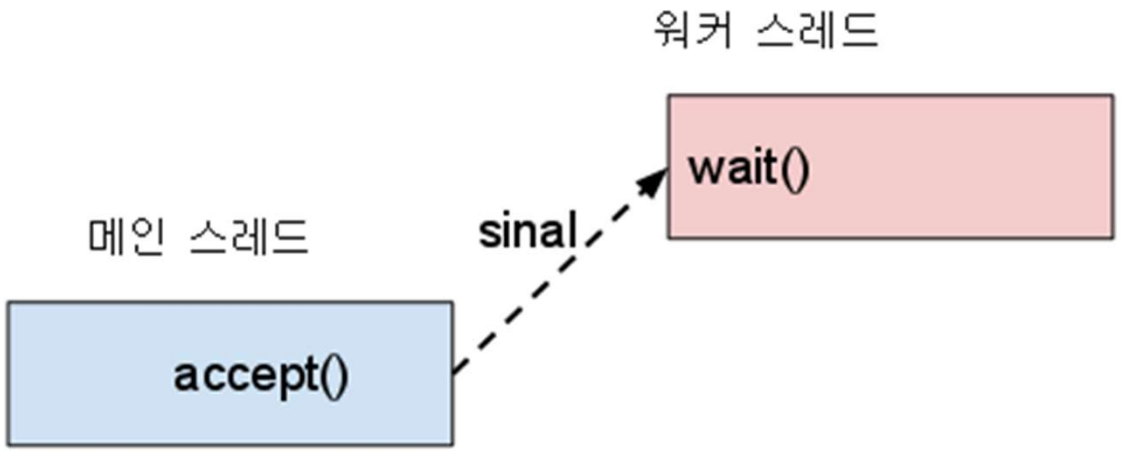
공유자원 보호 예제: count_thread_mutex.c

Lab



스레드 동기화

- 스레드를 서로 동기화 하기 위한 메커니즘
- **signal / wait** 으로 동기화 구현
  - 기다리는 스레드가 signal을 wait 하고
  - 이벤트를 전달하는 스레드가 signal을 전송, 스레드를 깨운다.



조건 변수를 이용한 스레드 동기화

- 조건 변수 : Pthread에서 제공하는 signal / wait 기반의 동기화 매커니즘

- 조건 변수 초기화 (생성)

int **pthread_cond_init**(pthread_cond_t *cond,
const pthread_cond_attr *attr);

- initialize the condition variable referenced by *cond* with attributes referenced by *attr*

- 조건 변수 기다리기

int **pthread_cond_wait**(pthread_cond_t *cond, pthread_mutex_t *mutex);

- block on a condition variable.
- shall be called with mutex locked by the calling thread or undefined behavior results

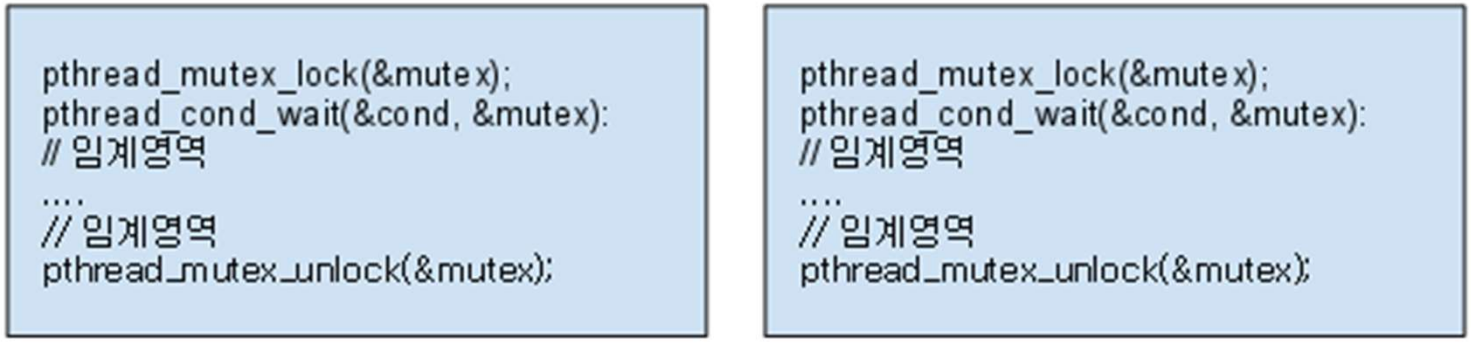
조건 변수를 이용한 스레드 동기화

- 조건 변수로 신호 보내기

int **pthread_cond_signal**(pthread_cond_t *cond);
int **pthread_cond_broadcast**(pthread_cond_t *cond);

- unblock threads blocked on a condition variable.
- The pthread_cond_broadcast() shall unblock all threads currently blocked on the specified condition variable cond.
- The pthread_cond_signal() shall unblock at least one of the threads that are blocked on the specified condition variable cond (if any threads are blocked on cond).

조건 변수를 이용한 스레드 동기화



- 조건 변수를 사용하는 영역은 **mutex**로 보호한다.
- pthread_cond_wait : signal을 기다린다.
  - signal이 전달되면, 기다리는 스레드들 중 하나가 mutex를 얻고 임계영역에 진입
  - 어떤 스레드가 깨어날지 알수 없음 : 어떤 스레드가 mutex를 얻을지 알수 없음으로.

공유자원 보호 예제: calc_multi.c

```
#define ARRAY_SIZE 100
#define THREAD_NUM 4

pthread_mutex_t t_lock;
pthread_cond_t t_cond;

int *data_array;
int sum_array[THREAD_NUM];

struct data_info
{
        int *d_point;
        int idx;
};
```

```
void *t_func(void *data)
{
        struct data_info d_info;
        int i = 0;
        int sum=0;
        d_info = *((struct data_info *)data);

        pthread_mutex_lock(&t_lock);
        pthread_cond_wait(&t_cond, &t_lock);
        printf("Start %d Thread\n", d_info.idx);
        pthread_mutex_unlock(&t_lock);

        for(i = 0; i < 25; i++)
                sum += d_info.d_point[(d_info.idx*25)+i];

        printf("(%d) %d\n", d_info.idx, sum);
        sum_array[d_info.idx] = sum;
        return NULL;
}
```

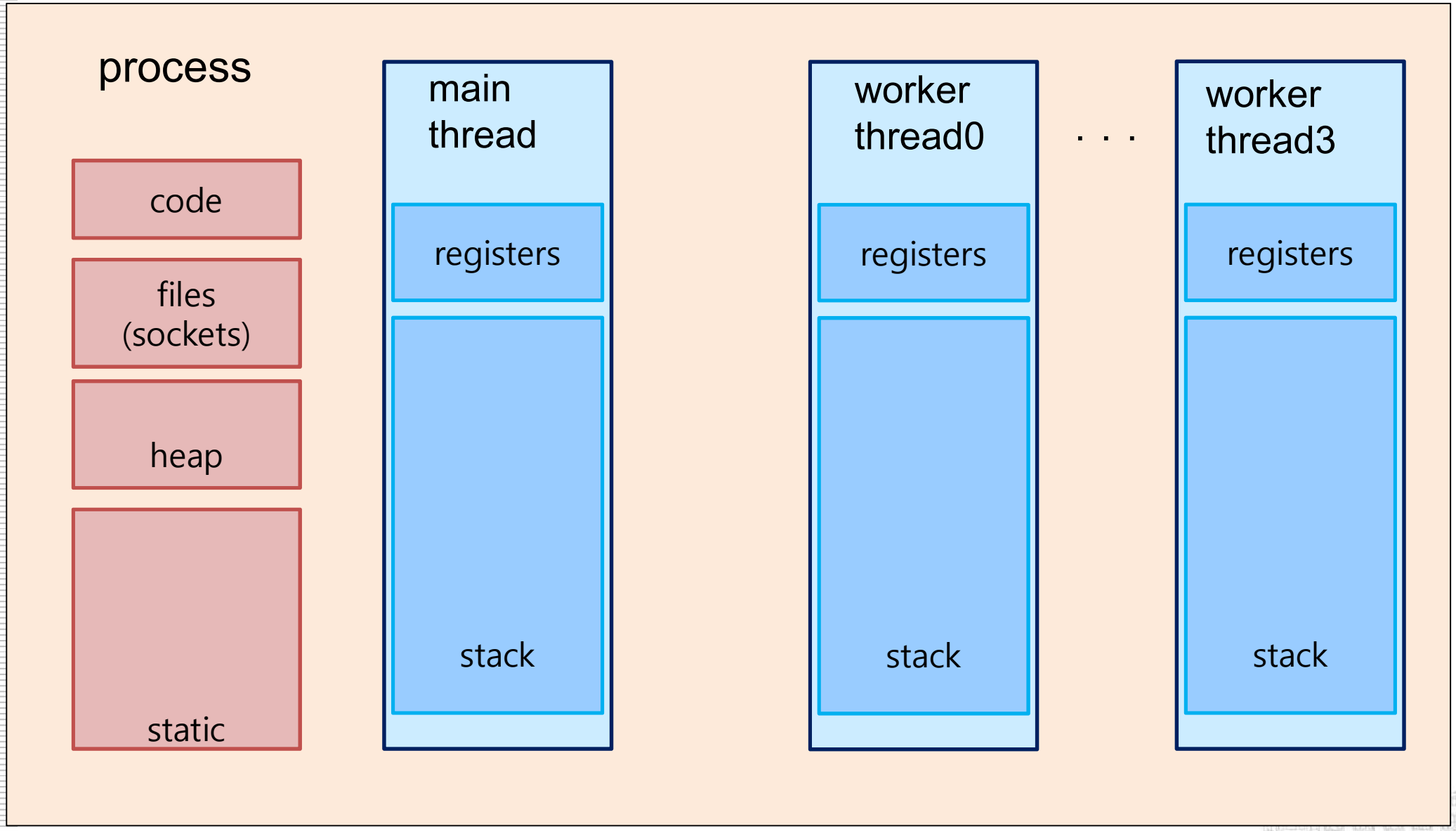
공유자원 보호 예제: calc_multi.c

```
int main(int argc, char **argv)
{
        int i=0;
        int sum=0;
        struct data_info d_info;
        pthread_t thread_id[THREAD_NUM];

        data_array = malloc(sizeof(int)*ARRAY_SIZE));

        pthread_mutex_init(&t_lock,NULL);
        pthread_cond_init(&t_cond, NULL);

        for(i = 0; i < THREAD_NUM; i++)
        {
                d_info.d_point = data_array;
                d_info.idx = i;

                pthread_create(&thread_id[i], NULL, t_func, (void *)&d_info);
                usleep(100);
        }
```

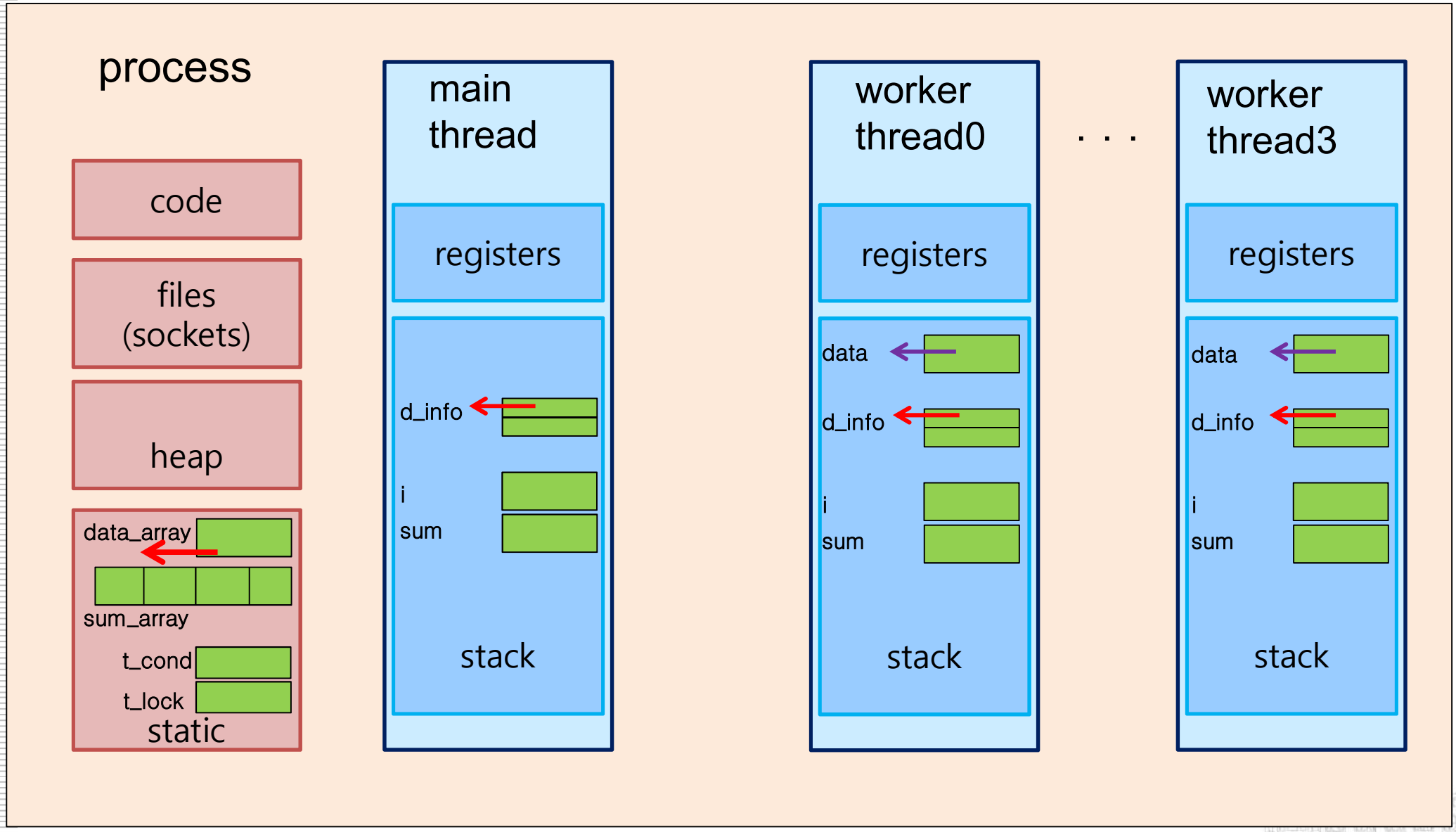
공유자원 보호 예제: calc_multi.c

```
        for (i = 0; i < ARRAY_SIZE; i++)
        {
                *data_array = i;
                *data_array++;
        }

        pthread_cond_broadcast(&t_cond);

        for(i = 0; i < THREAD_NUM; i++)
        {
                pthread_join(thread_id[i],NULL);
                sum += sum_array[i];
        }
        printf("SUM (0-99) : %d\n",sum);
        return 0;
}
```

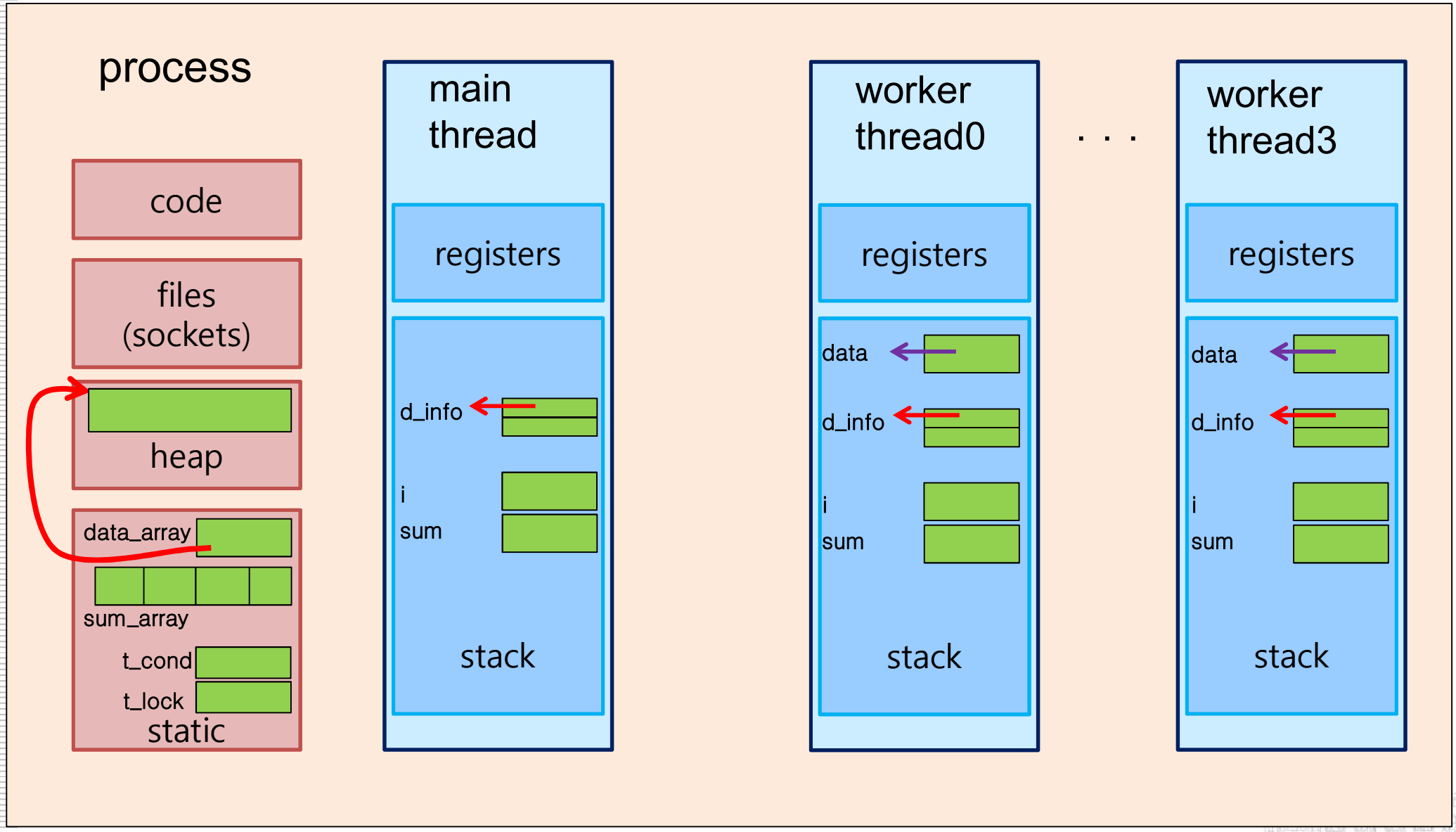
공유자원 보호 예제: calc_multi.c



공유자원 보호 예제: calc_multi.c

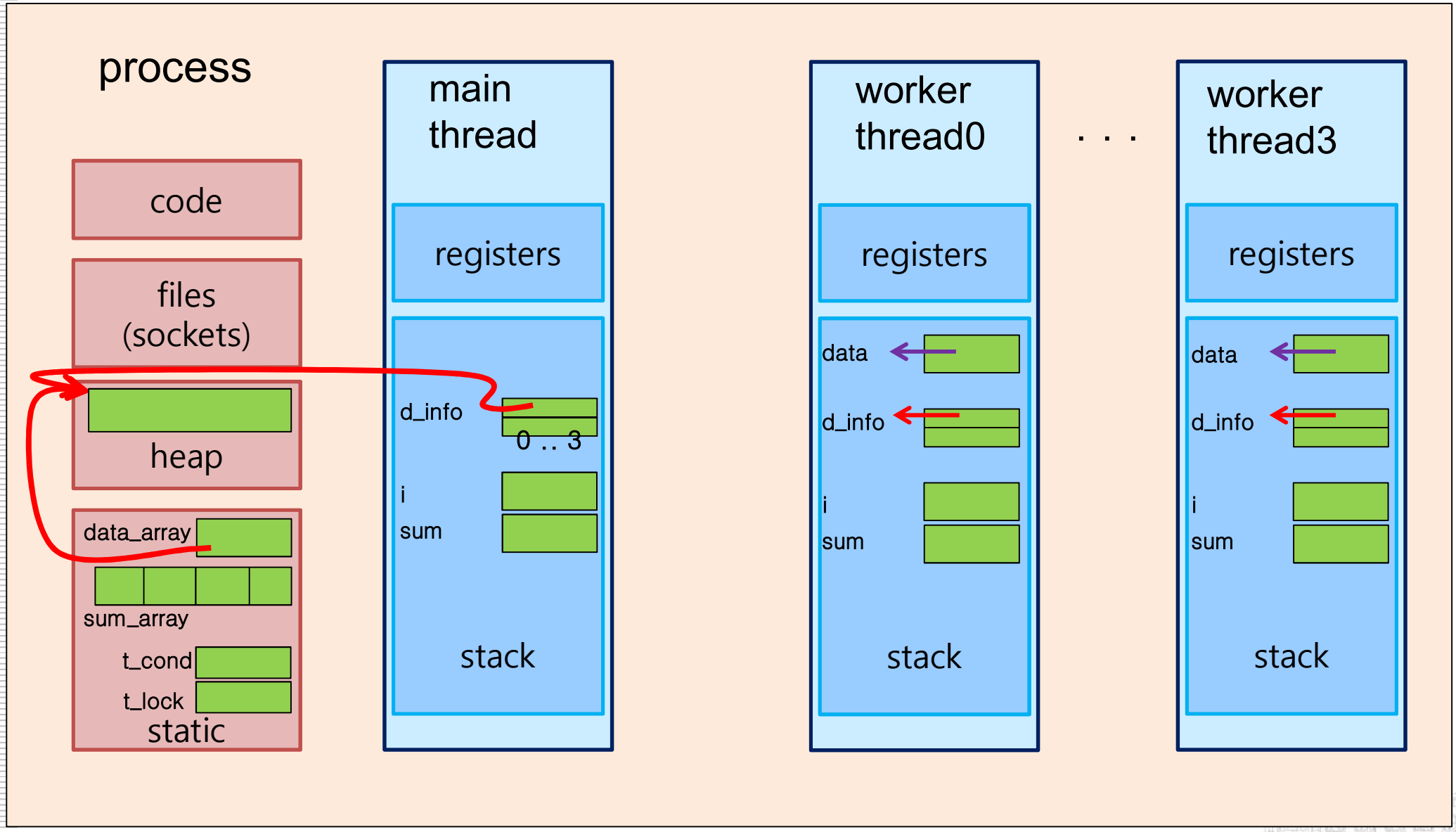


공유자원 보호 예제: calc_multi.c



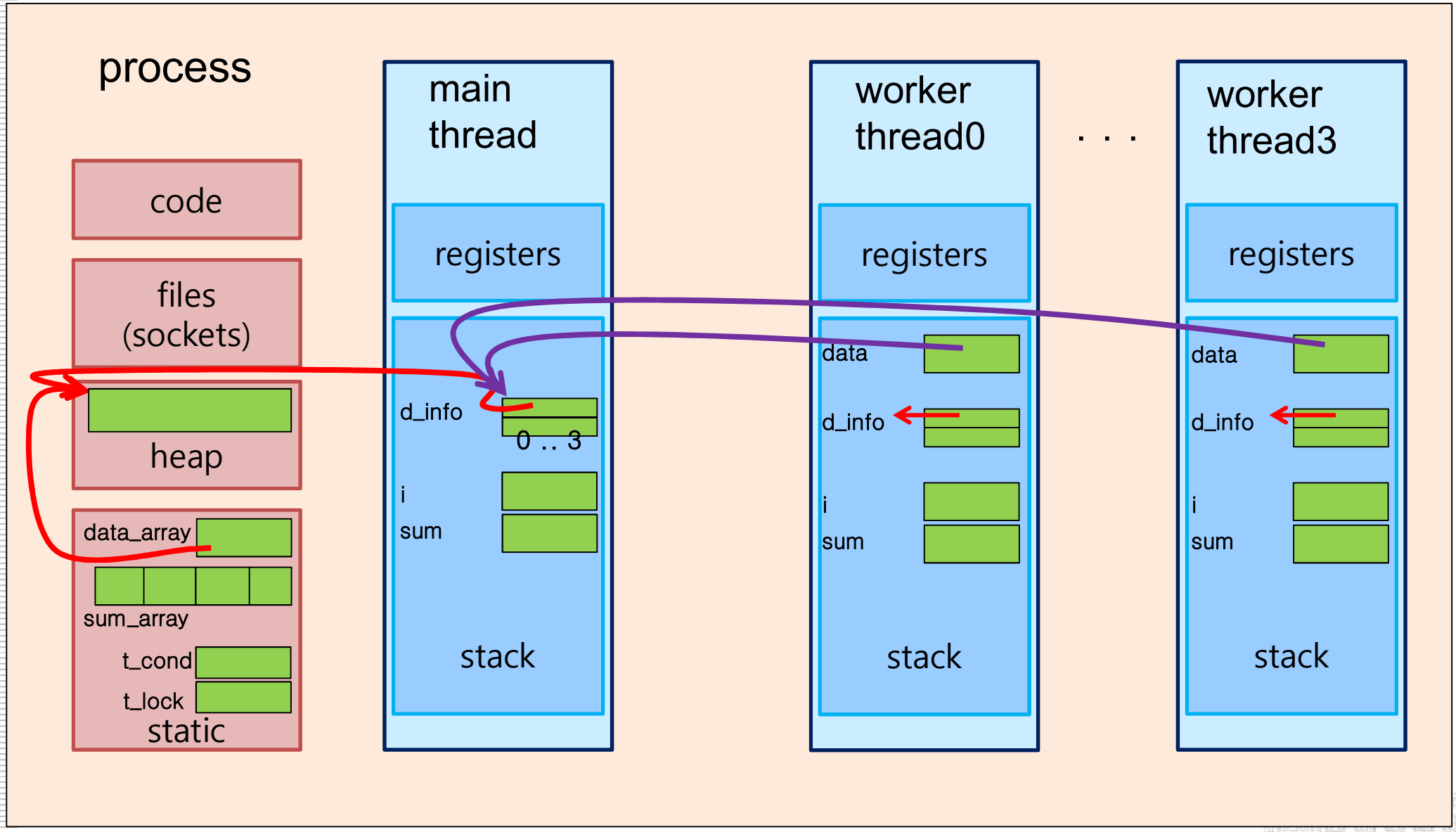
After executing data_array = malloc(sizeof(int)*ARRAY_SIZE)); // in main()

공유자원 보호 예제: calc_multi.c



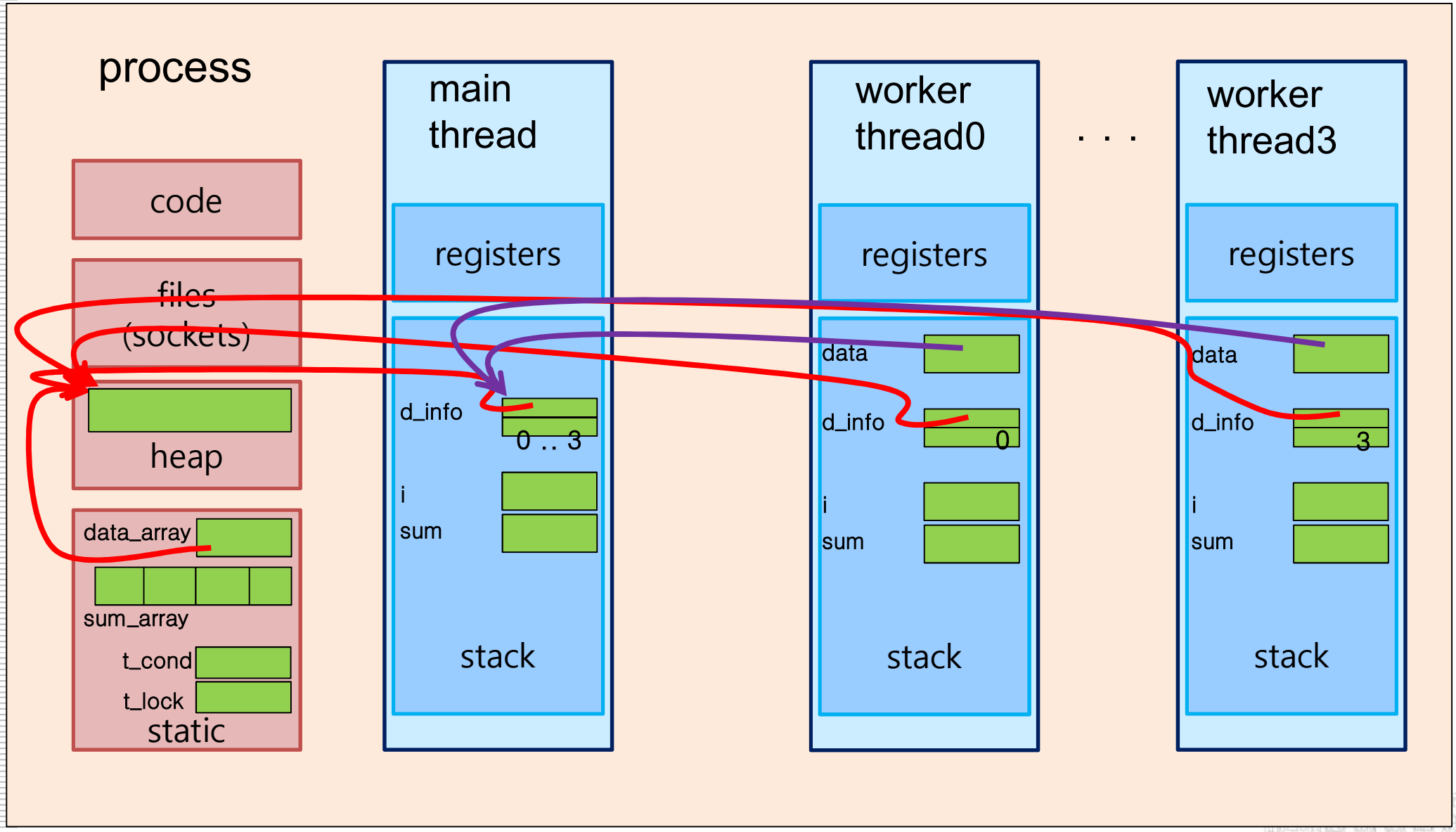
After executing d_info.d_point = data_array; // in main()
d_info.idx = i;

공유자원 보호 예제: calc_multi.c



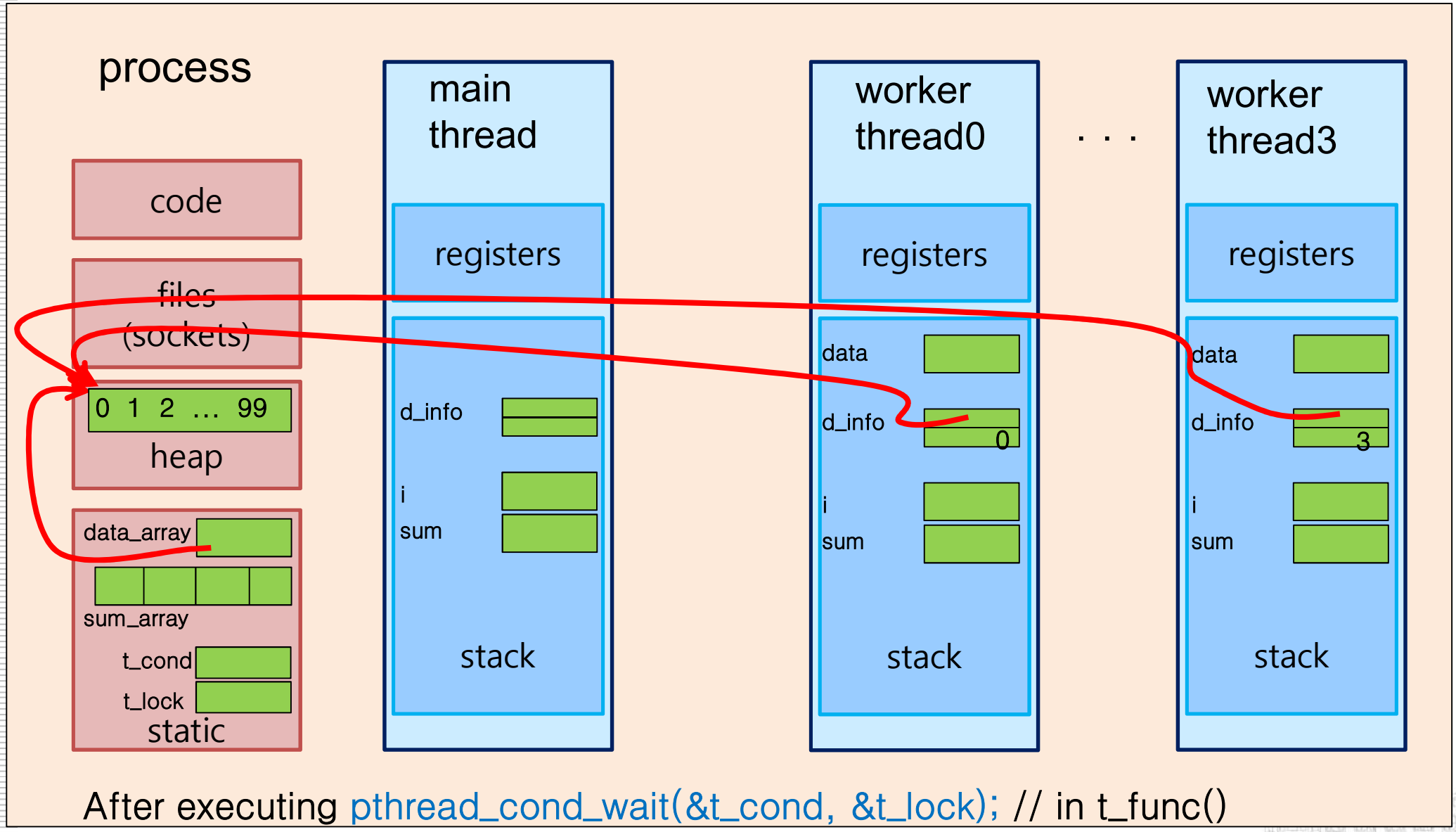
After executing pthread_create(…, t_func, (void *)&d_info); // in main()
void *t_func(void *data){} // in t_func()

공유자원 보호 예제: calc_multi.c



After executing d_info = *((struct data_info *)data); // in t_func()

공유자원 보호 예제: calc_multi.c

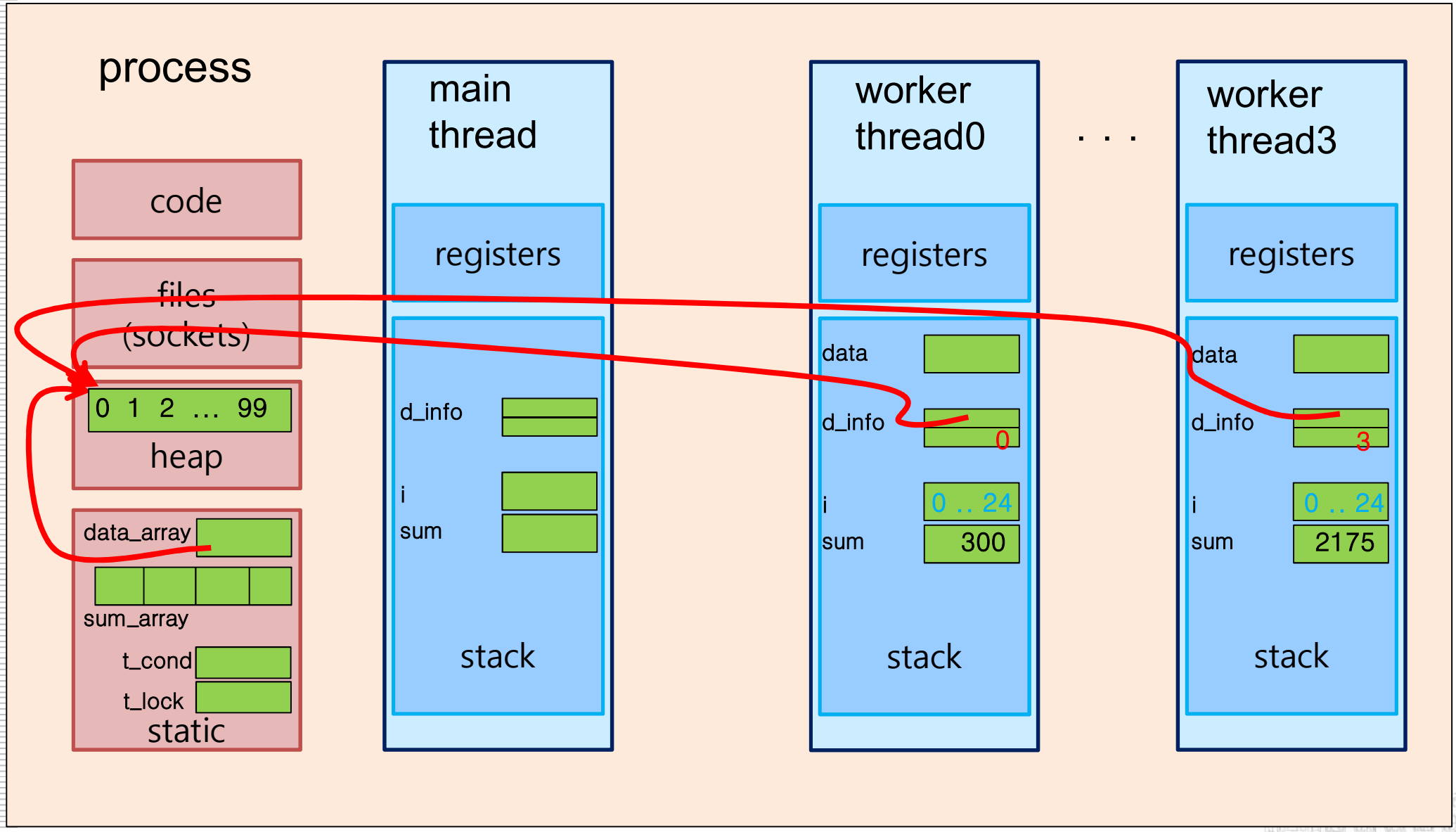


After executing pthread_cond_wait(&t_cond, &t_lock); // in t_func()

*data_array = i; *data_array++; // in main(), i=0..99

pthread_cond_broadcast(&t_cond); // in main()

공유자원 보호 예제: calc_multi.c



After executing sum += d_info.d_point[(d_info.idx*25)+i]; // t_func()

// 0+0..24, 25+0..24, 50 +0..24, 75 +0..24

공유자원 보호 예제: calc_multi.c

Lab

osnw00000000@osnw00000000-osnw: ~/lab08

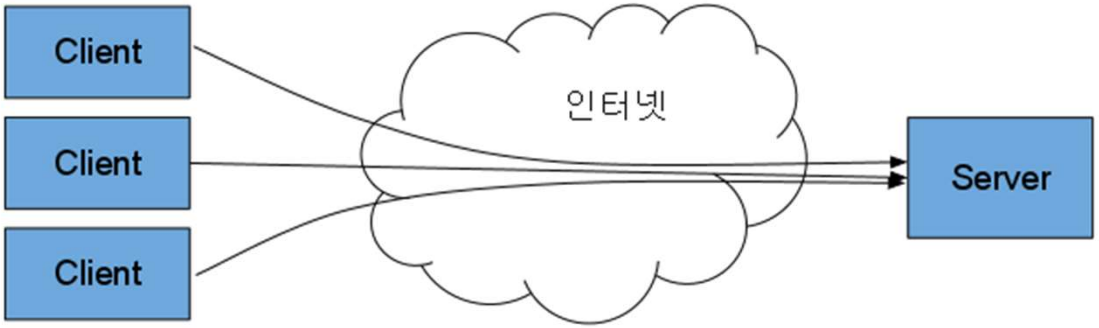
```
osnw00000000@osnw00000000-osnw:~/lab08$ ./calc_multi
Start 0 Thread
(0) 300
Start 3 Thread
(3) 2175
Start 1 Thread
(1) 925
Start 2 Thread
(2) 1550
SUM (0-99) : 4950
osnw00000000@osnw00000000-osnw:~/lab08$ ./calc_multi
Start 0 Thread
(0) 300
Start 2 Thread
(2) 1550
Start 1 Thread
(1) 925
Start 3 Thread
(3) 2175
SUM (0-99) : 4950
osnw00000000@osnw00000000-osnw:~/lab08$ ./calc_multi
Start 0 Thread
(0) 300
Start 2 Thread
(2) 1550
Start 1 Thread
(1) 925
Start 3 Thread
(3) 2175
SUM (0-99) : 4950
```

osnw00000000@osnw00000000-osnw: ~/lab08

```
osnw00000000@osnw00000000-osnw:~/lab08$ ./calc_multi
Start 2 Thread
Start 0 Thread
Start 3 Thread
(3) 2175
(2) 1550
(0) 300
Start 1 Thread
(1) 925
SUM (0-99) : 4950
osnw00000000@osnw00000000-osnw:~/lab08$ ./calc_multi
Start 0 Thread
(0) 300
Start 3 Thread
(3) 2175
Start 1 Thread
(1) 925
Start 2 Thread
(2) 1550
SUM (0-99) : 4950
osnw00000000@osnw00000000-osnw:~/lab08$ ./calc_multi
Start 0 Thread
(0) 300
Start 3 Thread
(3) 2175
Start 1 Thread
(1) 925
Start 2 Thread
(2) 1550
SUM (0-99) : 4950
```

멀티 프로세스와 소켓 프로그래밍

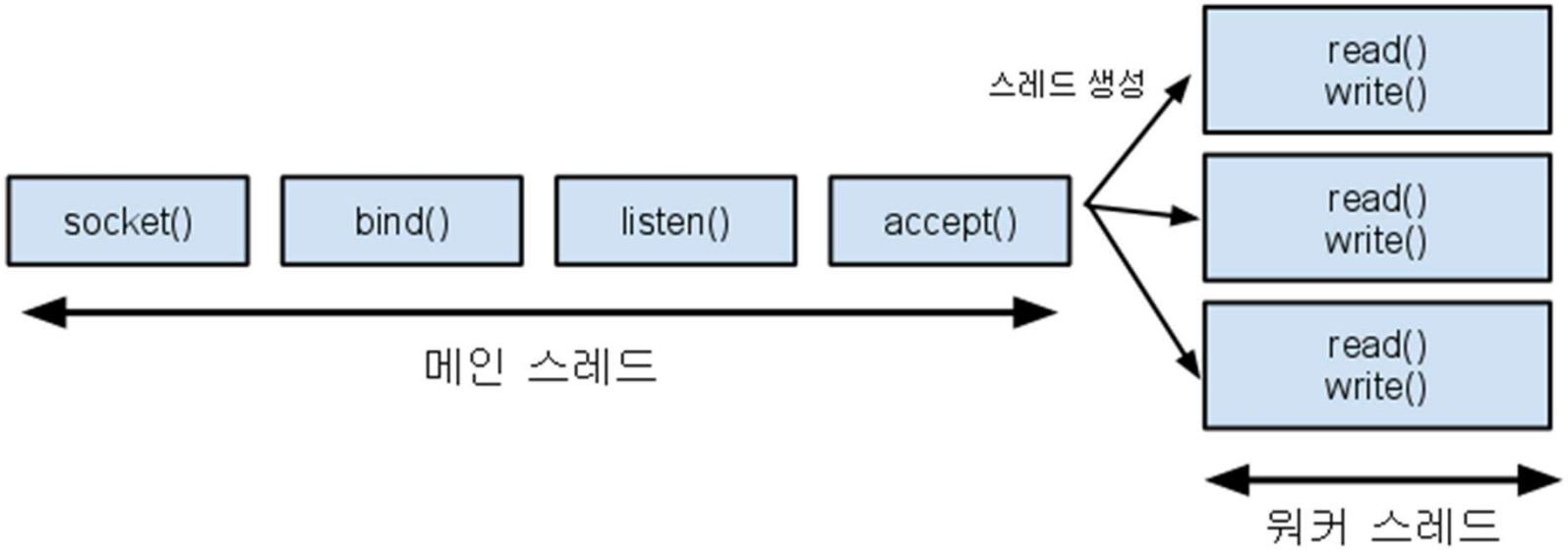
- 서버 프로그램은 다수의 클라이언트를 동시에 처리할 수 있어야 한다.



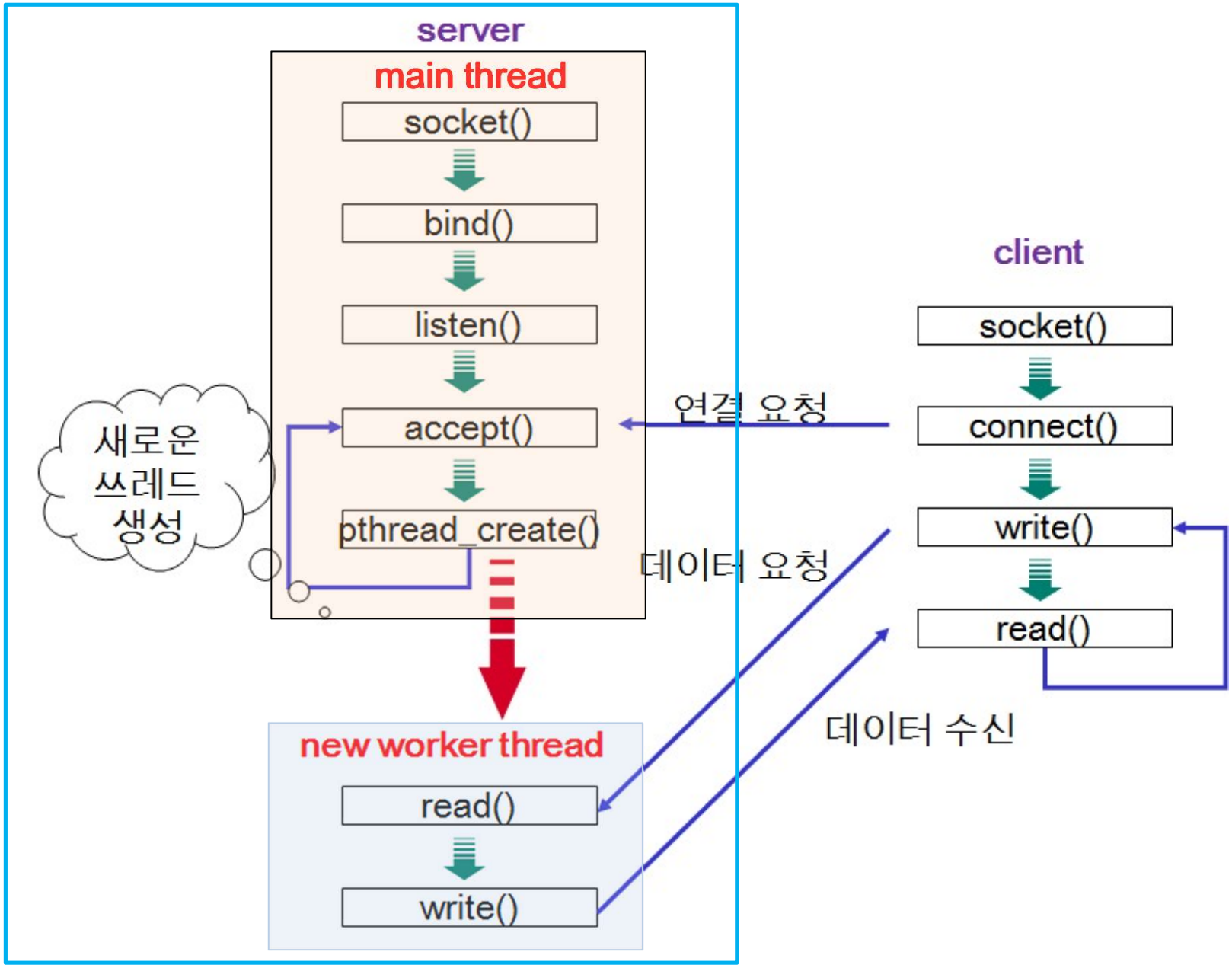
- 대표적인 다중 클라이언트 처리 기술(다중 접속처리 서버 기술)
  - 멀티 프로세스**(Multi-process)**
  - 멀티 쓰레드**(Multi-thread)**
  - 입출력다중화**(I/O Multiplexing)**

멀티 스레드 소켓 프로그램의 흐름

- 멀티 프로세스와 동일한 흐름
- **accept** 함수를 호출한 후 스레드 생성
  - fork 함수 대신 ptread_create() 함수를 호출
  - pthread_detach() 함수로 worker스레드를 main스레드로부터 분리하여 worker 스레드의 종료를 기다리지 않도록 해야 함



멀티 스레드 소켓 프로그램의 흐름



예제 프로그램 : echo_server_thread.c

```
void *thread_func(void *data)
{
      int  sockfd = *((int *)data);

      read(sockfd, ... );     write(sockfd, ... );      close(sockfd);
}

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

      listen_fd = socket();    bind(listen_fd, ... );      listen(listen_fd, ..);
      while(1)
      {
          client_fd = accept(listen_fd, ... );
          pthread_create(&thread_id, NULL, thread_func, (void *) &client_fd);
          pthread_detach(thread_id);
       }
}
```

예제 프로그램 : echo_server_thread.c

Lab

```
osnw00000000@osnw00000000-osnw: ~/lab08
osnw00000000@osnw00000000-osnw:~/lab08$ ./echo_server_thread
new client connected from 127.0.0.1:47924
Read Data 127.0.0.1(47924) : osnw2023
worker thread end
```

```
osnw00000000@osnw00000000-osnw: ~/lab08
osnw00000000@osnw00000000-osnw:~/lab08$ ./echo_client
osnw2023
read : osnw2023
osnw00000000@osnw00000000-osnw:~/lab08$
```

```
osnw00000000@osnw00000000-osnw: ~
osnw00000000@osnw00000000-osnw:~$ ps -eLf | grep echo_server_thread
osnw000+  118297  118054  118297  0    1 05:01 pts/3    00:00:00 ./echo_server_thread
osnw000+  118299  118270  118299  0    1 05:01 pts/4    00:00:00 grep --color=auto echo_server_thread
osnw00000000@osnw00000000-osnw:~$ ps -eLf | grep echo_server_thread
osnw000+  118297  118054  118297  0    2 05:01 pts/3    00:00:00 ./echo_server_thread
osnw000+  118297  118054  118301  0    2 05:01 pts/3    00:00:00 ./echo_server_thread
osnw000+  118303  118270  118303  0    1 05:01 pts/4    00:00:00 grep --color=auto echo_server_thread
osnw00000000@osnw00000000-osnw:~$ ps -eLf | grep echo_server_thread
osnw000+  118297  118054  118297  0    1 05:01 pts/3    00:00:00 ./echo_server_thread
osnw000+  118305  118270  118305  0    1 05:01 pts/4    00:00:00 grep --color=auto echo_server_thread
osnw00000000@osnw00000000-osnw:~$
```

Thank You !

뇌를 자극하는 TCP/IP 소켓 프로그래밍