# 시스템프로그래밍 4차 과제 보고서

### 소속 / 학번 / 이름

: 소프트웨어학부 / 20212277 / 김선재

### 개발 환경

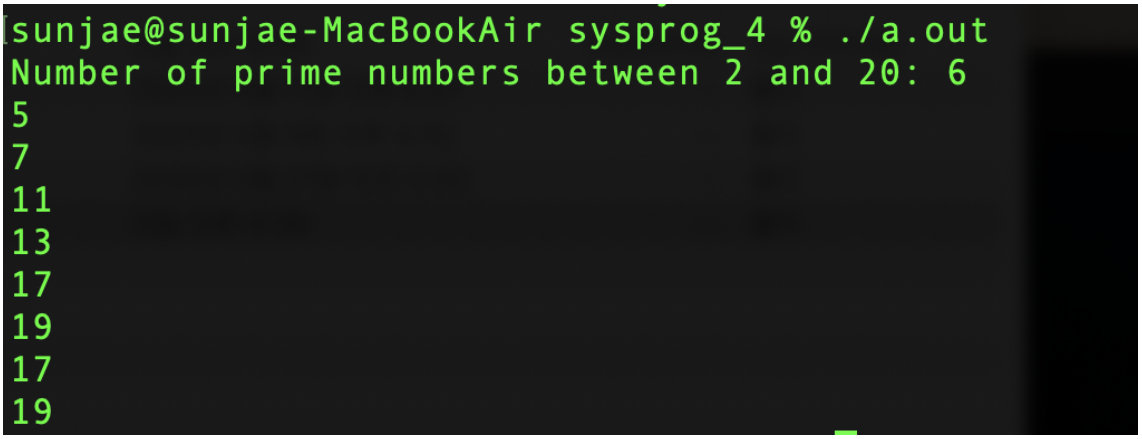
MacOS (UNIX)

### 작성 및 수정한 소스코드 설명

1. prime.c 에서 Race Condition 원인 찾기

: Race Condition이란, 하나의 자원을 두고 두 개 이상의 스레드나 프로세스가 경쟁하는 상태를 의미하고, 아래 코드에서는 여러 스레드가 모든 소수의 개수를 저장하는 변수 `total`과 소수들을 저장하는`primes`배열에 동시 접근하고 수정하기 때문에 동기화 처리를 해주지 않으면 아래처럼 비정상적인 결과가 나오게 된다.

(ex: N=20)




아래 코드에서 동기화 처리가 필요한 부분을 찾으면,

먼저 work함수 내에서 primes 배열에 찾은 소수를 저장하고, total 값을 1증가시키는 부분이다. 이 부분에서 스레드들이 동기화 처리되지 않으면 여러 스레드가 동시에 두 변수의 값을 읽거나 수정할 수 있기 때문에 위처럼 소수가 빠지거나, 중복되어 저장되거나, 개수가 제대로 카운트되지 않는 것처럼 비정상적인 결과가 나오게 된다.

```
#include <stdio.h>
#include <math.h>
#include <pthread.h>

#define THREADS 4 // 스레드 수
#define N 3000 // 0-3000

int primes[N];
int pflag[N];
int total = 0;
```

```
int is_prime(int v) // 소수인지 확인
{
    int i;
    int bound = floor(sqrt ((double)v)) + 1;
    for (i = 2; i < bound; i++) {
        /* No need to check against known composites */
        if (!pflag[i])
            continue;
        if (v % i == 0) {
            pflag[v] = 0;
            return 0;
        }
    }
    return (v > 1);
}

void *work(void *arg) // pthread 의 시작 함수
{
    int start;
    int end;
    int i;
    start = (N/THREADS) * (*(int *)arg);  // 스레드가 처리하기 시작할 숫자
    end = start + N/THREADS; // end-1까지 처리 (총 N/THREAD 개만큼)
    for (i = start; i < end; i++) {
        if ( is_prime(i) ) { /* Race Condition 발생 */
            primes[total] = i; // 여러 스레드가 동시에 배열에 소수 저장 -> 같은 인덱스
            total++;        // 여러 스레드가 동시에 개수 증가
        }
    }
    return NULL;
}

int main(int argn, char **argv)
{
    int i;
    pthread_t tids[THREADS-1];
    for (i = 0; i < N; i++) {
        pflag[i] = 1;
    }

    for (i = 0; i < THREADS-1; i++) {
        pthread_create(&tids[i], NULL, work, (void *)&i);
         /* 반복문 안에서 i 값이 불필요하게 수정될 수 있다.
         -> 각 스레드에서 독립적인 값을 가지도록 수정해야함 */
    }

    i = THREADS-1;
    work((void *)&i);
```

```
    printf("Number of prime numbers between 2 and %d: %d\n",
           N, total);
    for (i = 0; i < total; i++) {
        printf("%d\n", primes[i]);
    }

    return 0;
}
```

1. **20212277-mut.c (테스트 환경 : THREAD→4, N→3000)**

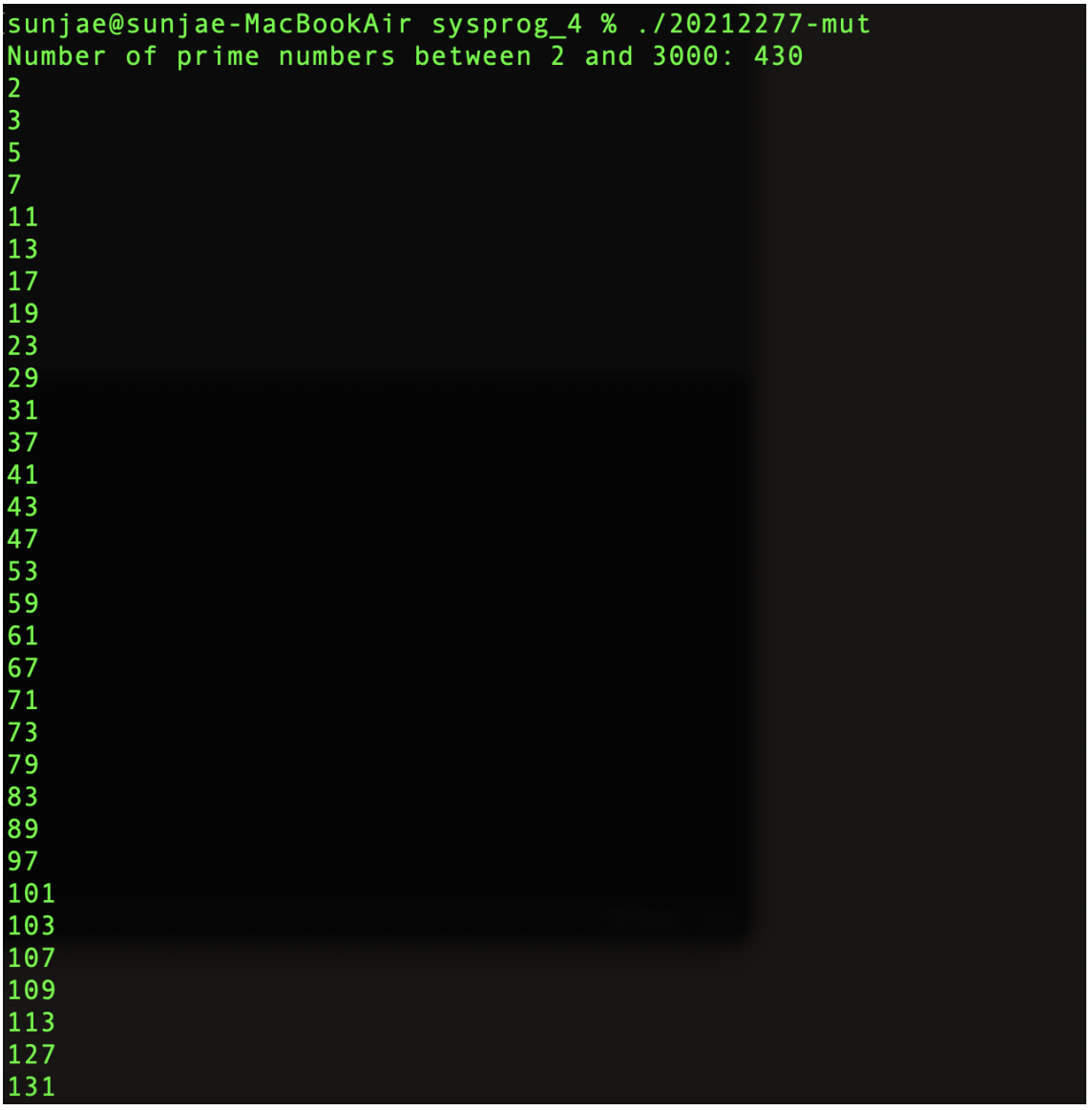
: 뮤텍스로 사용할 변수를 선언하고, work 함수 내부 critical section의 전/후에 lock / unlock 함수를 사용하여 상호 배제를 보장한다. 스레드를 생성한 후 생성된 스레드가 작업을 마칠때까지 대기하는 pthread_join을 호출하였다. 나머지 스레드가 작업을 모두 완료하면, 마지막 남은 메인 스레드가 남은 작업을 처리한다. 가장 마지막에는 사용한 뮤텍스를 해제한다.

```
#include <stdio.h>
#include <math.h>
#include <pthread.h>

#define THREADS 4 // 스레드 수
#define N 3000 // 0-3000

int primes[N];
int pflag[N];
int total = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;  // mutex 초기화

int is_prime(int v)
{
   // 생략 ...
}

void *work(void *arg) // pthread 의 시작 함수
{
    int start;
    int end;
    int i;

    start = (((N/THREADS) * (*(int *)arg)))+2 ;
    end = (*(int *)arg == THREADS - 1) ? (N%2==0) ? N : N+1 : start + (N / TH
    for (i = start; i < end; i++) {
        if ( is_prime(i) ) {
            pthread_mutex_lock(&mutex); // 잠금
            primes[total] = i;
            total++;
```

```
            pthread_mutex_unlock(&mutex); // 해제

        }
    }
    return NULL;
}

int main(int argn, char **argv)
{
    int i;
    pthread_t tids[THREADS-1];
    int thread_ids[THREADS-1]; // 각 스레드에서 고유한 시작,종료점을 가지도록

    for (i = 0; i < N; i++) {
        pflag[i] = 1;
    }
    for (i = 0; i < THREADS-1; i++) {
        thread_ids[i]=i;
        pthread_create(&tids[i], NULL, work, (void *)&thread_ids[i]);
    }

    for (i = 0; i < THREADS-1; i++) {
        pthread_join(tids[i],NULL); // 자식 스레드가 종료할 때까지 대기
    }

    i = THREADS-1;
    work((void *)&i);

    printf("Number of prime numbers between 2 and %d: %d\n", N, total);
    for (i = 0; i < total; i++) {
        printf("%d\n", primes[i]);
    }
    pthread_mutex_destroy(&mutex); // 뮤텍스 종료

    return 0;
}
```

- 실행 결과

: 2~N까지 존재하는 소수가 중복되거나 빠지지 않고 올바르게 카운트되고, 출력된 것을 확인할 수 있다.




(결과가 길어서 이하 생략하였습니다.)

2. **20212277-sem.c**

: 세마포어 변수를 포인터 형으로 선언하고, main 함수 안에서 sem_open 함수를 통해 고유한 이름을 가진 세마포어를 초깃값 1로 생성한다. 생성하면 critical section 전/후에서 sem_wait, sem_post를 통해 잠그고 작업을 끝낸 뒤 잠금을 해제해 스레드 간 동기화가 이루어지도록 한다. 모든 작업 후 sem_close를 호출하여 세마포어를 해제한다.

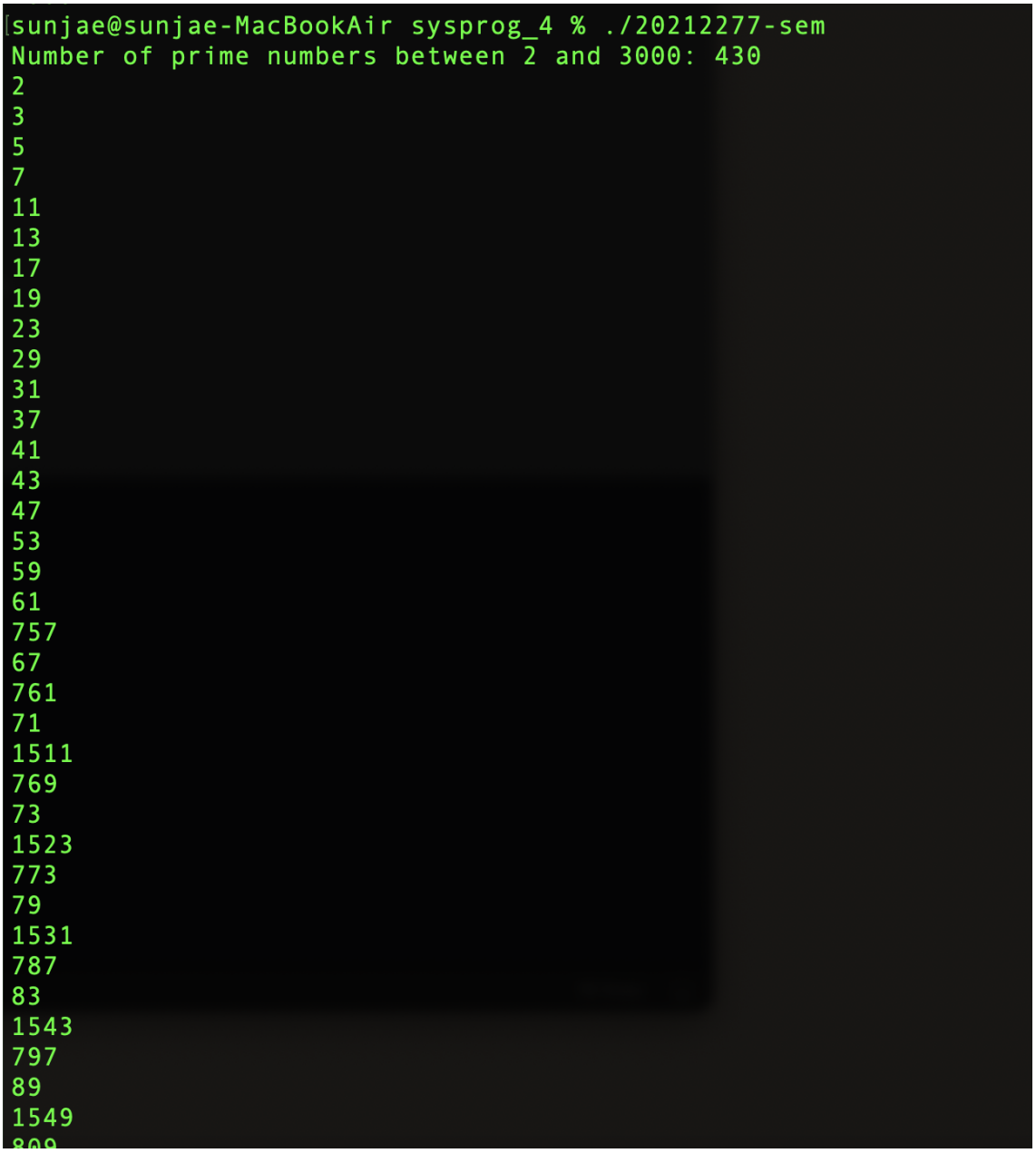
```
#include <stdio.h>
#include <math.h>
#include <pthread.h>
#include <semaphore.h>

#define THREADS 4
#define N 3000
```

```
int primes[N];
int pflag[N];
int total = 0;

sem_t *sem; // 세마포어 변수

int is_prime(int v)
{
   // 생략 ...
}

void *work(void *arg)
{
    int start;
    int end;
    int i;

    start = (((N/THREADS) * (*(int *)arg)))+2 ;
    end = (*(int *)arg == THREADS - 1) ? (N%2==0) ? N : N+1 : start + (N / TH
    for (i = start; i < end; i++) {
        if (is_prime(i)) {
            sem_wait(sem); // 잠금
            primes[total] = i;
            total++;
            sem_post(sem); // 해제
        }
    }
    return NULL;
}

int main(int argn, char **argv)
{
    int i;
    pthread_t tids[THREADS-1];
    int thread_ids[THREADS-1];
    // 세마포어 할당
    if((sem = sem_open("/semaphore",O_CREAT,0644, 1)) == SEM_FAILED){
        return -1;
    }
    for (i = 0; i < N; i++) {
        pflag[i] = 1;
    }
    for (i = 0; i < THREADS-1; i++) {
        thread_ids[i] = i;
        pthread_create(&tids[i], NULL, work, (void *)&thread_ids[i]);
    }
    for (i = 0; i < THREADS-1; i++) {
        pthread_join(tids[i],NULL);
```

```
    }

    i = THREADS-1;
    work((void *)&i);

    printf("Number of prime numbers between 2 and %d: %d\n",
           N, total);

    for (i = 0; i < total; i++) {
        printf("%d\n", primes[i]);
    }

    sem_close(sem); // 해제

    return 0;
}
```

- 실행 결과

: 이전과 같이 소수의 개수가 정확히 카운트 되었고, 소수들이 제대로 저장되어 출력되었다.




3. **20212277-cv.c**

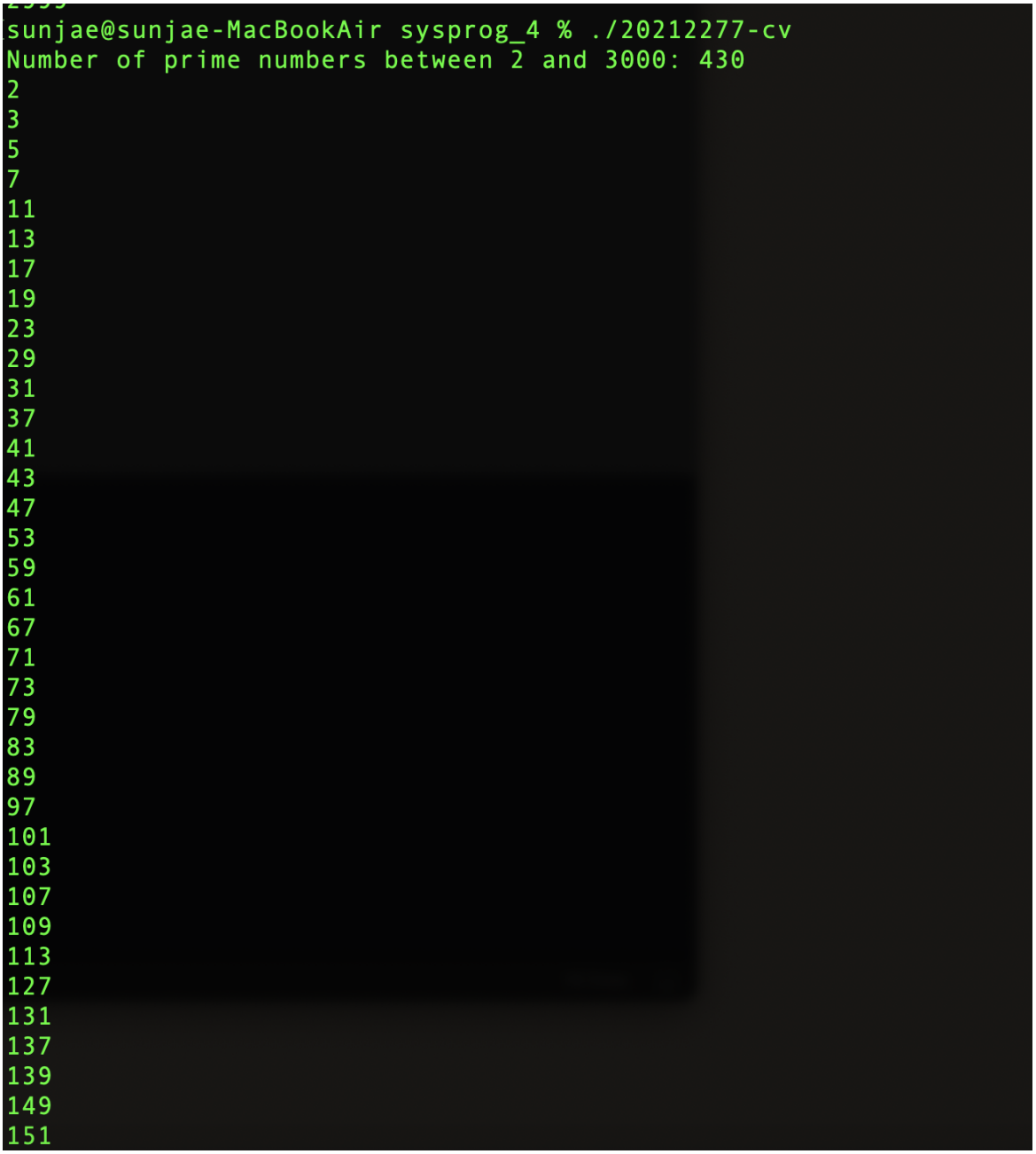
: 조건 변수는 뮤텍스와 함께 사용해서 동기화를 지원하도록 했다. 뮤텍스를 통해 위처럼 상호 배제를 지원하고, 조건 변수를 통해 스레드 간 신호를 전달하여 뮤텍스를 보조하고 더 정교한 동기화를 지원한다. done 변수를 통해 각 스레드가 작업이 끝나면 변수값을 1증가시키도록하고, 마지막 스레드까지 작업을 완료하면 시그널을 보낸다. 메인 스레드는 마지막 스레드가 작업을 완료할 때까지 대기하다가, 시그널을 받고 작업을 종료시킨다.

```
#include <stdio.h>
#include <math.h>
#include <pthread.h>

#define THREADS 4
#define N 3000
```

```
int primes[N];
int pflag[N];
int total = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // 뮤텍스 설정
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;  // 조건 변수 설정
int done = 0;

int is_prime(int v)
{
   // 생략 ...
}

void *work(void *arg)
{
    int start;
    int end;
    int i;

    start = (((N/THREADS) * (*(int *)arg)))+2 ;
    end = (*(int *)arg == THREADS - 1) ? (N%2==0) ? N : N+1 : start + (N / TH
    for (i = start; i < end; i++) {
        if (is_prime(i)) {
            pthread_mutex_lock(&mutex);
            primes[total] = i;
            total++;
            pthread_mutex_unlock(&mutex);
        }
    }
    pthread_mutex_lock(&mutex);
    done++; // 작업을 완료한 스레드가 1 증가시킨다.
    if (done == THREADS) {
        pthread_cond_signal(&cond);
    }
    pthread_mutex_unlock(&mutex);

    return NULL;
}

int main(int argn, char **argv)
{
    int i;
    pthread_t tids[THREADS-1];
    int thread_ids[THREADS-1];

    for (i = 0; i < N; i++) {
        pflag[i] = 1;
```

```
    }
    for (i = 0; i < THREADS-1; i++) {
        thread_ids[i] = i;
        pthread_create(&tids[i], NULL, work, (void *)&thread_ids[i]);
    }

    for (i = 0; i < THREADS-1; i++) {
        pthread_join(tids[i], NULL);
    }

    i = THREADS-1;
    work((void *)&i);

    pthread_mutex_lock(&mutex);
    while (done < THREADS) {
        pthread_cond_wait(&cond, &mutex);
    }
    pthread_mutex_unlock(&mutex);

    printf("Number of prime numbers between 2 and %d: %d\n",N, total);

    for (i = 0; i < total; i++) {
        printf("%d\n", primes[i]);
    }

    pthread_mutex_destroy(&mutex); // 뮤텍스 해제
    pthread_cond_destroy(&cond); // 조건 변수 해제
    return 0;
}
```

- 실행 결과




## Makefile

- 소스 파일명: 20212277-mut.c 20212277-sem.c 20212277-cv.c
- 실행 파일명: 20212277-mut 20212277-sem 20212277-cv

```
CC = gcc
CFLAGS = -Wall -pthread

all: 20212277-mut 20212277-sem 20212277-cv

20212277-mut: 20212277-mut.c
    $(CC) $(CFLAGS) 20212277-mut.c -o 20212277-mut
```

```
20212277-sem: 20212277-sem.c
    $(CC) $(CFLAGS) 20212277-sem.c -o 20212277-sem

20212277-cv: 20212277-cv.c
    $(CC) $(CFLAGS) 20212277-cv.c -o 20212277-cv

clean:
    rm -f 20212277-mut 20212277-sem 20212277-cv
```

### 발생한 문제 및 해결 과정

- `pthread_create()` 함수 사용

: 첫번째 인자로 전달된 스레드에 식별자를 저장하고, 스레드의 속성을 attr 포인터에 전달된 값으로 지정한다. NULL일 경우 기본 속성을 사용한다. 세번째 인자는 새로 생성된 스레드가 실행할 함수에 대한 포인터로, void *형을 인자로 받고 void *형을 반환한다. 마지막 네 번째 인자는 start_routine 함수에 전달될 인자이다.

start_routine 함수로 work함수가 실행되며, 각 스레드는 정해진 범위에서 소수를 찾아 배열에 저장하고, total 값을 업데이트한다.

main 함수에서는 정해진 개수만큼 스레드를 생성하고 work 함수를 실행하도록 하며, 모든 스레드의 작업이 끝나면 pthread_join 함수를 통해 스레드를 종료시킨다.

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*sta
```

- semaphore 사용 시 발생한 경고

: 세마포어 사용 코드에서 세마포어를 초기화하고 해제하기 위해 사용한 sem_init() 과 sem_destroy 함수가 사용되지 않는다는 경고이다. 찾아본 결과, 개발 환경인 mac os에서 해당 함수를 지원하지 않아서 발생한 이슈였다. 대신에 sem_open, sem_close라는 이름의 함수를 사용하도록 코드를 수정하였다.

```
2.c:53:5: warning: 'sem_init' is deprecated [-Wdeprecated-declarations]
...
2.c:64:5: warning: 'sem_destroy' is deprecated [-Wdeprecated-declarations]
```

```
#include <semaphore.h>
// 이름 있는 세마 포어를 생성하거나 접근한다. sem_init 보다 느리다.
sem_t * sem_open (const char *name, int oflag, mode_t mode, unsigned value)
...
// 잠금 및 해제
sem_wait(sem);
sem_post(sem);

// sem_open에 의해 지정된 세마포어를 해제
int sem_close (sem)
sem_t *sem; // 세마포어 선언
```

- start, end index 범위 문제

: 기존 코드에서 아래와 같이 start, end 인덱스를 결정하고있는데, 이렇게 하면 N = 17인 경우, 16부터는 검사하지 않고 작업이 종료되게 된다. 0~N 중에서 실질적으로 스레드가 검사하려는 소수는 2부터 시작하고, 맨 마지막 스레드의 경우 end 인덱스가 끝까지 처리하고 작업을 종료할 수 있도록 아래와 같이 수정하였다.

```
..// work 함수 내부

 start = (N/THREADS) * (*(int *)arg);
 end = start + N/THREADS;


 // 수정 후
 start = (((N/THREADS) * (*(int *)arg))) + 2 ;
 end = (*(int *)arg == THREADS - 1) ? N+1 : start + (N / THREADS);
```

- 소수 검사 함수에서 N이 짝수일 경우 primes[0] 값이 overwrite 되는 문제

: 기존 코드에서는 N이 짝수일 때 검사 대상이 2일 경우 소수가 출력될 때 2가 아니라 0이 출력되는 문제가 있었다. 아래 코드에 printf문으로 확인해본 결과 반복문의 맨 마지막 스레드 작업 수행 시 마지막 반복문에서 값을 0으로 덮어쓰고 있었다.

위에서 N이 짝수이든 홀수이든 상관없이 end 인덱스의 최댓값을 N+1으로 잡았기 때문에 아래처럼 검사 대상(i)가 N일 경우 primes[0] 값이 overwrite되었다.

```
void *work(void *arg)
{
    int start;
    int end;
    int i;
    start = (((N/THREADS) * (*(int *)arg)))+2 ;
    end = (*(int *)arg == THREADS - 1) ? N+1 : start + (N / THREADS);
    for (i = start; i < end; i++) {
        if (is_prime(i)) {
            ...
        }
        printf("%d %d\n",i, primes[0]);
    }
    return NULL;
}
```

```
9 2
10 2
11 2
17 2
18 2
19 2
20 0
Number of prime numbers between 2 and 20: 8
0
```

>> 출력결과 N=20 일 경우 마지막 반복문에서 0으로 덮어쓰는 것을 확인할 수 있다.

따라서 아래와 같이 짝수일 경우에는 최댓값을 N이 되도록 수정하였다.

```
start = (((N/THREADS) * (*(int *)arg)))+2 ;
end = (*(int *)arg == THREADS - 1) ? (N%2==0) ? N : N+1 : start + (N / THREA
...
```

```
Number of prime numbers between 2 and 3000: 430
2
3
5
7
11
13
17
```

>> 올바르게 출력됨

참고 사이트

https://medium.com/helderco/semaphores-in-mac-os-x-fd7a7418e13b

https://www.ibm.com/docs/ko/aix/7.3?topic=s-sem-close-subroutine

https://www.ibm.com/docs/ko/aix/7.3?topic=s-sem-open-subroutine