01 mycp, myls, myshell

Index

- mycp
- mycp_adv
- myls
- Semaphore
- Dead lock
- Starvation

mycp

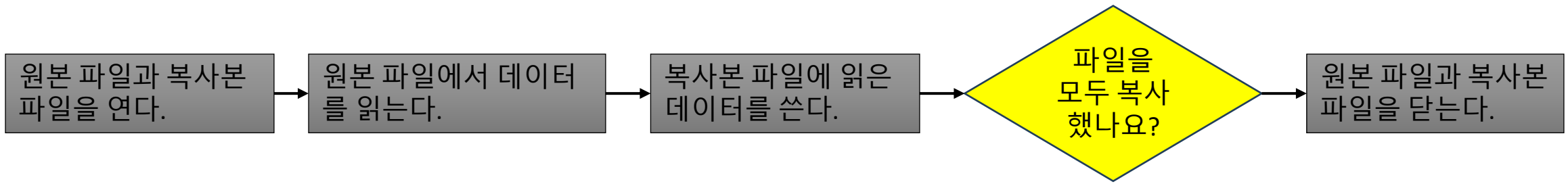
❖ Input

- Usage: ./mycp [*origin file*] [*destination file*]

❖ Output

- 원본 파일 origin file 의 user data가 적혀 있는 복사본 파일 destination file 생성

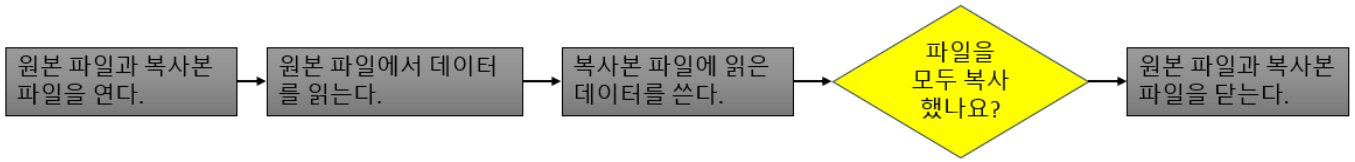
❖ mycp algorithm



mycp

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *argv[]){
    int fd_origin, fd_dest, read_size, write_size =0;
    char buf[MAX_BUF];

    if (argc!=3){
        printf("USAGE: %s origin dest\n",argv[0]);
        exit(-1);
    }

    fd_origin = [1] fill out here using system call ;
    fd_dest = [1-1] fill out here using system call ;
    if (fd_origin < 0 || fd_dest <0){
        //open error handling
        perror("fd open error\n");
    }

//read from the origin file
    while((read_size = [2] fill out here using system call ) != 0 ){
        //write to the dest file
        write_size = [3] fill out here using system call ;
    }

    [4] fill out here using system call

}
```

❖ mycp algorithm



Using System call

- open
- read
- write
- close

mycp

❖ mycp 실행화면

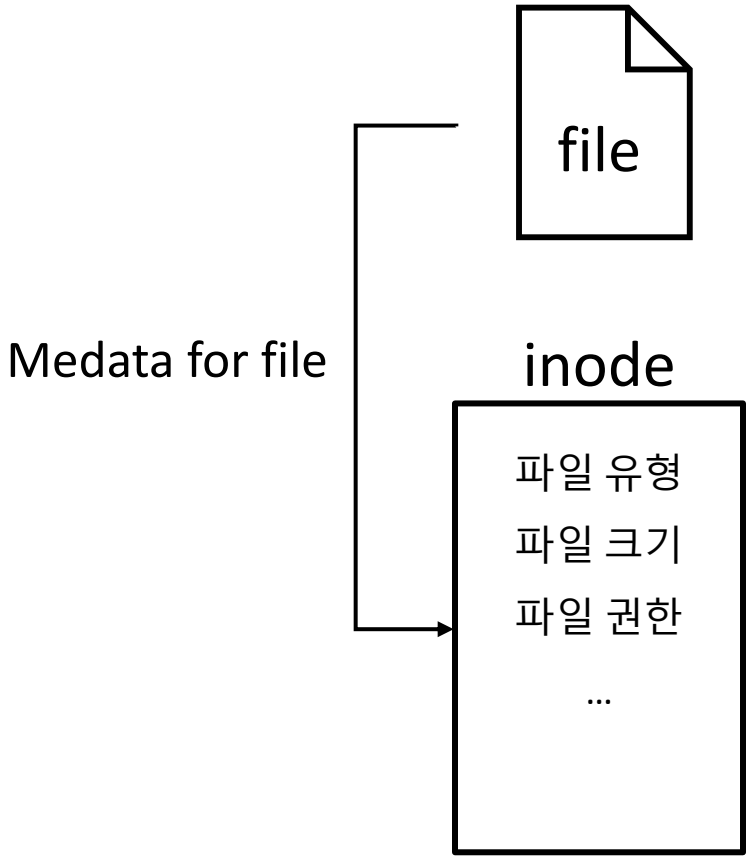
- cat [file]
- 파일의 데이터 비교

```
[ec2-user@ip-172-31-15-105 taba9]$ ls -l
total 28
-rwxr-xr-x. 1 ec2-user ec2-user 17744 Aug 27 17:49 mycp
-rw-r--r--. 1 ec2-user ec2-user   766 Aug 27 18:06 mycp.c
-rw-r--r--. 1 ec2-user ec2-user    17 Aug 27 18:24 origin
[ec2-user@ip-172-31-15-105 taba9]$ ./mycp origin dest
[ec2-user@ip-172-31-15-105 taba9]$ cat dest
I am origin file
[ec2-user@ip-172-31-15-105 taba9]$ cat origin
I am origin file
[ec2-user@ip-172-31-15-105 taba9]$
```

mycp_adv

❖ 파일 file

- User data
  - 실제 파일에 저장되는 데이터
- Metadata
  - File의 정보와 user data의 위치를 가리키는 데이터
  - 파일 유형, 파일 크기, 파일 권한…



mycp_adv

❖ Input

- Usage: ./mycp_adv [*origin file*] [*destination file*]

❖ Output

- 원본 파일 origin file 의 user data가 적혀 있는 복사본 파일 destination file 생성
- 파일의 속성 정보 (Metadata)를 포함한 완전 복사

mycp_adv

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>
#define MAX_BUF 64

int main(int argc, char *argv[]){
    int fd_origin, fd_dest, read_size, write_size = 0;
    char buf[MAX_BUF];
    struct stat* stat_origin = (struct stat*)malloc(sizeof(struct stat));
    mode_t flag_origin;

    if (argc != 3){
        printf("USAGE: %s origin dest\n", argv[0]);
        exit(-1);
    }
    fd_origin = open(argv[1], O_RDONLY);

    [1] get file attribute structure from fd_origin

    flag_origin = stat_origin-> [2] let's get member from struct stat "stat_origin->field_here

    fd_dest = open(argv[2], O_RDWR | O_CREAT | O_EXCL | O_SYNC, flag_origin);
while((read_size = read(fd_origin, buf, MAX_BUF)) != 0){
        write_size = write(fd_dest, buf, read_size);
    }
    close(fd_origin);
    close(fd_dest);
}
```

Using System call

fstat

mycp_adv

❖ mycp_adv 실행화면

- cat [file]
- 파일의 데이터 비교

```
[ec2-user@ip-172-31-15-105 taba9]$ ./mycp_adv origin dest
[ec2-user@ip-172-31-15-105 taba9]$ ls -l
total 56
-rw-r--r--. 1 ec2-user ec2-user    17 Aug 27 20:23 dest
-rwxr-xr-x. 1 ec2-user ec2-user 17744 Aug 27 17:49 mycp
-rwxr-xr-x. 1 ec2-user ec2-user 17792 Aug 27 20:23 mycp_adv
-rw-r--r--. 1 ec2-user ec2-user   805 Aug 27 20:18 mycp_adv.c
-rw-r--r--. 1 ec2-user ec2-user   766 Aug 27 18:06 mycp.c
-rw-r--r--. 1 ec2-user ec2-user    17 Aug 27 18:24 origin
[ec2-user@ip-172-31-15-105 taba9]$ cat origin
I am origin file
[ec2-user@ip-172-31-15-105 taba9]$ cat dest
I am origin file
```

mycp-adv

❖ stat [file]

- 파일의 메타데이터 비교

```
[ec2-user@ip-172-31-15-105 taba9]$ stat origin
  File: origin
  Size: 17              Blocks: 8          IO Block: 4096   regular file
Device: 10302h/66306d   Inode: 8413986     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/ec2-user)   Gid: ( 1000/ec2-user)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2024-08-27 18:24:23.238560580 +0000
Modify: 2024-08-27 18:24:12.954589960 +0000
Change: 2024-08-27 18:24:12.954589960 +0000
 Birth: 2024-08-27 18:24:12.954589960 +0000
[ec2-user@ip-172-31-15-105 taba9]$ stat dest
  File: dest
  Size: 17              Blocks: 8          IO Block: 4096   regular file
Device: 10302h/66306d   Inode: 8413988     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/ec2-user)   Gid: ( 1000/ec2-user)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2024-08-27 20:24:06.411915356 +0000
Modify: 2024-08-27 20:23:42.476984046 +0000
Change: 2024-08-27 20:23:42.476984046 +0000
 Birth: 2024-08-27 20:23:42.476984046 +0000
```

myls

❖ Input

- Usage: ./myls *[argument]* *선택
- [argument]는 디렉토리

❖ Output

- 디렉토리 내의 파일과 디렉토리 목록을 보여줌
- 1. ./myls
  - 현재 디렉토리 내의 파일과 디렉토리 목록을 보여줌
- 2. ./myls [argument]
  - argument에 해당하는 디렉토리 내의 파일과 디렉토리 목록을 보여줌

myls

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <dirent.h>
#include <sys/types.h>

int main(int argc, char *argv[]) {
// 변수 선언
    DIR *dir = NULL;
    struct dirent* dentry = NULL;
    char *dir_name = ".";

    // 예외 처리
    if (argc == 1) { // args 없는 경우 현재 디렉토리 "." 내용을 보여줌.
        dir = opendir(dir_name);
    } else if (argc == 2) {
        dir_name = argv[1]; // 디렉토리 이름을 argv에서 가져옴
        dir = [1] fill out here using directory syscall
    } else {
        printf("argc %d : We only accept 1 or 2 args for now\n", argc);
        exit(-1);
    }
    while ((dentry = [2] using dir syscall ) != NULL) {
        printf("%s \n", dentry->d_name); // 디렉토리 항목 이름을 출력
    }
    [3] using dir syscall
}
```

Using System call

opendir

readdir

closedir

myls

❖ myls 실행화면

- 인자(argv) 0개와 1개 비교

```
[ec2-user@ip-172-31-15-105 taba9]$ ./myls
.
..
mycp
mycp.c
origin
mycp_adv.c
mycp_adv
dest
myls.c
myls
```

```
[ec2-user@ip-172-31-15-105 taba9]$ ./myls ..
.
..
.bash_logout
.bash_profile
.ssh
.bash_history
.vimrc
.bashrc
day2
taba7
taba9
.viminfo
```

그 이외 파일 입출력 관련 시스템 콜

- create()
- mkdir(), readdir(), rmdir()
- pipe()
- mknod()
- link(),unlink()
- dup(),dup2()
- stat(),fstat()
- chmod(), fchmod()
- Ioctl(), fcntl()
- Sync(), fsync()

binary semaphore

❖ thread_bin_sem.c

(1) sem_init(semaphore, p_shared, initial_value): 세마포어를 초기화 및 생성

- semaphore: 세마포어의 주소
- p_shared

  0인 경우, 스레드 간에 공유

  0인 아닌 경우, 프로세스 간에 공유
- Initial_value: 공유 자원에 들어 갈 수 있는 스레드의 개수

(2) sem_destroy(semaphore): 세마포어를 제거

- assert(int expression): expressio이 0이면 error 발생(거짓이면 에러 발생)
- Semaphore vs Mutex
  - Semaphore: 스레드
  - Mutex: 프로세스, 스레드

* 임계 구역 보호, 스레드 간의 동기화

binary semaphore

pthread			
반환값	pthread_create() pthread_join()	성공	0
		실패	**1**

❖ thread_bin_sem.c

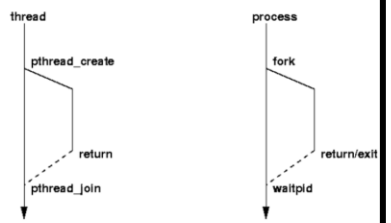
(1) sem_init(semaphore, p_shared, initial_value): 세마포어를 초기화 및 생성

(2) sem_destroy(semaphore): 세마포어를 제거

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

int count = 0;
int nthread = 1;
int worker_loop_cnt = 1;
sem_t semaphore;
static void *work(void *num);

int main(int argc, char *argv[]){
    pthread_t *th;
    void *value;
    long i;

    if(argc < 3){
        fprintf(stderr, "%s parameter : nthread, worker_loop_cnt\n", argv[0]);
        exit(-1);
    }

    nthread = atoi(argv[1]);
    worker_loop_cnt = atoi(argv[2]);
    th = malloc(sizeof(pthread_t) * nthread);

(1) sem_init(&semaphore, 0, 1);
    printf("main: begin (count = %d)\n", count);
    for(i = 0; i < nthread; i++)
        assert(pthread_create(&th[i], NULL, work, (void*)i) == 0);
    for(i = 0; i < nthread; i++)
        assert(pthread_join(th[i], &value) == 0);
(2) sem_destroy(&semaphore);
    printf("main: done (count = %d)\n", count);
}
```

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

- **새로운 프로세스를 생성하는 함수**
- 인자1 : 생성된 스레드의 ID를 저장할 변수의 포인터
- 인자2 : 스레드의 특성을 설정할 때 사용, 주로 NULL이 온다.
- 인자3 : 스레드가 생성되고 나서 실행될 함수
- 인자4 : 세번째 인자에서 호출되는 함수에 전달하고자 하는 인자



int **pthread_join**(pthread_t th, void **thread_return)

- 스레드가 종료될 때까지 호출한 스레드가 기다리도록 하게 함
- 인자1 : 스레드 ID. 이 ID가 종료할 때까지 실행을 지연
- 인자2 : 쓰레드 종료시 반환값

binary semaphore

❖ thread_bin_sem.c

(3) sem_wait(semaphore): semaphore 잠금

- semaphore: 세마포어의 주소
- 카운트 1 감소

(4) sem_post(semaphore): semaphore 해제

- semaphore: 세마포어의 주소
- 카운트 1 증가

binary semaphore

❖ thread_bin_sem.c

(3) sem_wait(): semaphore 잠금

(4) sem_post(): semaphore 해제

```
38 static void *work(void *num) {
39     long number = (long)num;
40
41   (3) sem_wait(&semaphore);
42
43     for(int i = 0; i < worker_loop_cnt; i++)
44         count++;
45     printf("Thread number %d: %d \n", number, count);
46
47   (4) sem_post(&semaphore);
48
49     return NULL;
50 }
```

binary semaphore

❖ 실행파일

```
[ec2-user@ip-172-31-15-105 day7]$ gcc -pthread -o thread_bin_sem thread_bin_sem.c
```

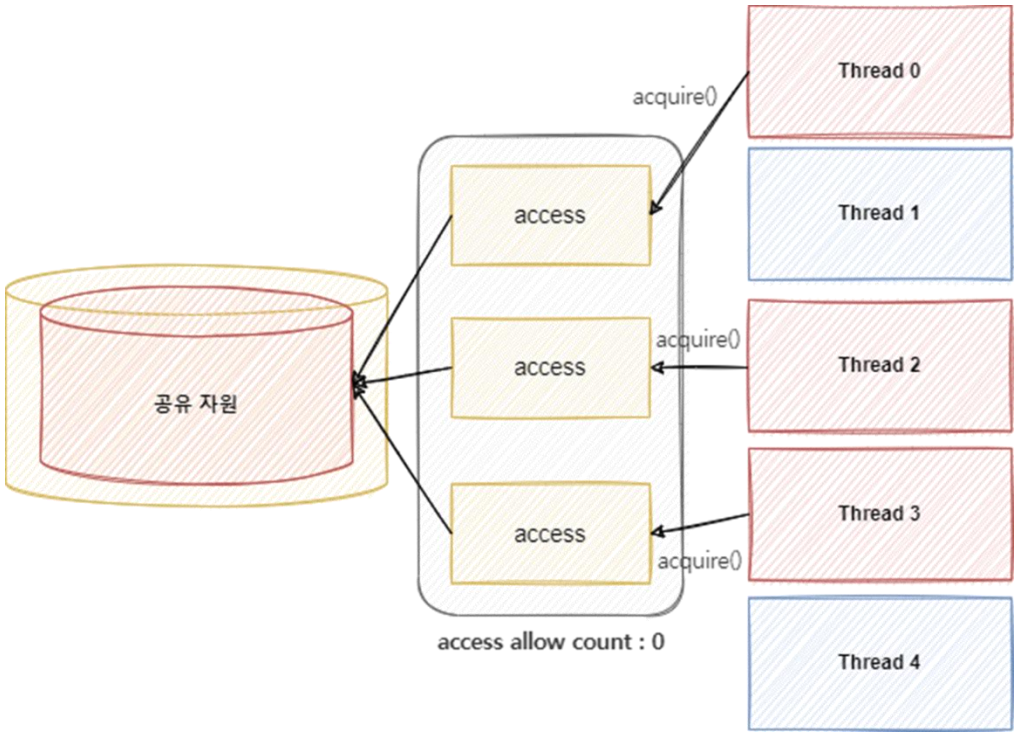
❖결과

```
[ec2-user@ip-172-31-15-105 day7]$ ./thread_bin_sem 5 10000
main: begin (count = 0)
Thread number 0: 10000
Thread number 2: 20000
Thread number 3: 30000
Thread number 1: 40000
Thread number 4: 50000
main: done (count = 50000)
```

counting semaphore

❖ thread_counting_sem.c

- 선언부




```
 1 #include <stdlib.h>
 2 #include <unistd.h>
 3 #include <stdio.h>
 4 #include <assert.h>
 5 #include <pthread.h>
 6 #include <semaphore.h>
 7
 8 #define SEM_COUNT 3
 9
10 int count[SEM_COUNT];
11 int working [SEM_COUNT];
12 int nthread = 1;
13 int worker_loop_cnt = 1;
14
15 pthread_mutex_t lock;
16 sem_t semaphore;
17
18 static void *work(void *num);
19
```

counting semaphore

❖ thread_counting_sem.c

(1) pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)

- pthread_mutex_t *mutex: 뮤텍스 객체에 대한 포인터
- const pthread_mutexattr_t *attr: 기본 속성일 경우 NULL

counting semaphore

❖ thread_counting_sem.c

- main 함수

```
20 int main(int argc, char *argv[])
21 {
22     pthread_t *th;
23     void *value;
24     long i;
25
26     if (argc < 3) {
27         fprintf(stderr, "%s parameter : nthread, worker_loop_cnt\n", argv[0]);
28         exit(-1);
29     }
30
31     nthread = atoi(argv[1]);
32     worker_loop_cnt = atoi(argv[2]);
33
34     th = malloc(sizeof(pthread_t) * nthread);
35
36 (1) pthread_mutex_init(&lock, NULL); // initialize the lock
37     sem_init(&semaphore, 0, SEM_COUNT); // init sem
38
39     for(i = 0; i < nthread; i++) {
40         assert(pthread_create(&th[i], NULL, work, (void*) i) == 0);
41     }
42
43     for(i = 0; i < nthread; i++) {
44         assert(pthread_join(th[i], &value) == 0);
45     }
47     sem_destroy(&semaphore);
48
49     free(th);
50
51     printf("Count array :\n");
52     for(int i = 0; i < SEM_COUNT; i++) {
53         printf("%d ", count[i]);
54     }
55
56     printf("\nComplete\n");
57 }
```

counting semaphore

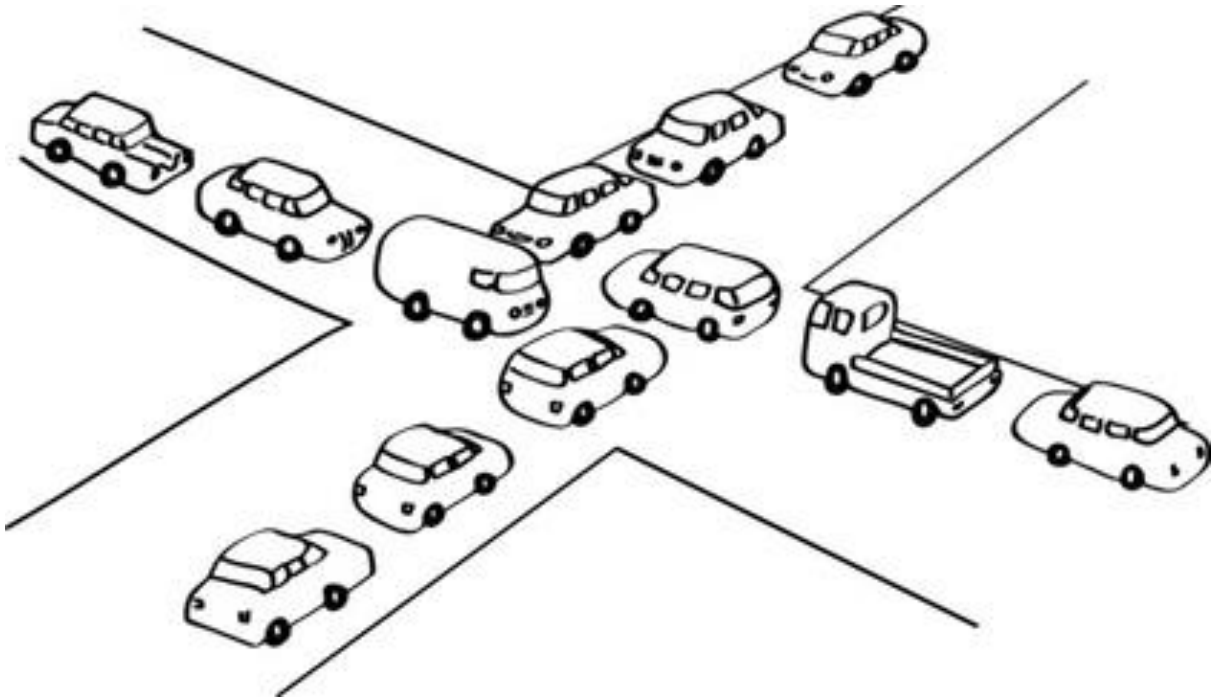
❖ thread_counting_sem.c

- work 함수

```
59 static void *work(void* num)
60 {
61     long number = (long)num;
62     int count_index = -1;
63
64     sem_wait(&semaphore); // sem count down
65
66     pthread_mutex_lock(&lock); // lock
67     for(int i = 0; i < SEM_COUNT; i++) {
68         if(working[i] == 0) {
69             working[i] = 1;
70             count_index = i;
71             break;
72         }
73     }
74     pthread_mutex_unlock(&lock); // unlock
75
76     if(count_index == -1){
77         fprintf(stderr, "Thread number %d: count_index < 0\n", number);
78         exit(-1);
79     }
80
81     for(int i = 0; i < worker_loop_cnt; i++)
82         count[count_index]++;
83
84     //printf("Thread number %d: %d \n", number, count[count_index]);
85
86     pthread_mutex_lock(&lock); // lock
87     working[count_index] = 0;
88     pthread_mutex_unlock(&lock); // unlock
89
90     sem_post(&semaphore); // sem count up
91
92     return NULL;
93 }
```

Dead lock

❖ thread_dead_lock.c



Dead lock

❖ thread_dead_lock.c

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

int first_count = 0;
int second_count = 0;
int nthread = 1;
int nthread_one = 1;

int main_loop_cnt = 1;

pthread_mutex_t first_lock;
pthread_mutex_t second_lock;
```

```c
int main(int argc, char *argv[])
{
    pthread_t *th;
    void *value;
    long i;

    if (argc < 3) {
        fprintf(stderr, "%s parameter : nthread, main_loop_cnt\n", argv[0]);
        exit(-1);
    }

    nthread = atoi(argv[1]);
    nthread_one = nthread/2;

    main_loop_cnt = atoi(argv[2]);

    th = malloc(sizeof(pthread_t) * nthread);

    pthread_mutex_init(&first_lock, NULL); // initialize the lock
    pthread_mutex_init(&second_lock, NULL); // initialize the lock

    for(int loop = 0; loop < main_loop_cnt; loop++){
        printf("---- loop %d ----\n", loop);

        for(i = 0; i < nthread_one; i++)
            assert(pthread_create(&th[i], NULL, work_one, (void*) i) == 0);

        for(i = nthread_one; i < nthread; i++)
            assert(pthread_create(&th[i], NULL, work_two, (void*) i) == 0);

        for(i = 0; i < nthread; i++)
            assert(pthread_join(th[i], &value) == 0);

        first_count = 0;
        second_count = 0;
    }
}
```

Dead lock

❖ thread_dead_lock.c

```
17 static void *work_one(void* num)
18 {
19     long number = (long)num;
20     int answer = 0;
21  pthread_mutex_lock(&first_lock); // lock
22     pthread_mutex_lock(&second_lock); // lock
23
24     answer = first_count + second_count;
25
26     printf("Work_one : %d \n", answer);
27
28     first_count++;
29     second_count++;
30
31     pthread_mutex_unlock(&second_lock); // unlock
32     pthread_mutex_unlock(&first_lock); // unlock
33
34     return NULL;
35 }
```

```
37 static void *work_two(void* num)
38 {
39     long number = (long)num;
40
41     int answer = 0;
42
43     pthread_mutex_lock(&second_lock); // lock
44     pthread_mutex_lock(&first_lock); // lock
45
46     answer = first_count + second_count;
47
48     printf("Work_two : %d \n", answer);
49
50     first_count++;
51     second_count++;
52
53     pthread_mutex_unlock(&first_lock); // unlock
54     pthread_mutex_unlock(&second_lock); // unlock
55
56     return NULL;
57 }
```

Lock: Starvation

❖ starvation.c

```
 1 #include <stdlib.h>
 2 #include <unistd.h>
 3 #include <stdio.h>
 4 #include <assert.h>
 5 #include <pthread.h>
 6 #include <semaphore.h>
 7
 8 sem_t mutex;
 9 int count = 0;
10 int nthread = 1;
11 int worker_loop_cnt = 1;
```

Lock: Starvation

❖ starvation.c

```
13 void *work(void *number)
14 {
15     int i;
16     long thread_id = (long)number;
17
18     for (i = 0; i < worker_loop_cnt; i++) {
19         sem_wait(&mutex);
20         count++;
21         printf("Thread %d: count = %d\n", thread_id, count);
22         sem_post(&mutex);
23     }
24
25     pthread_exit(NULL);
26 }
```

Lock: Starvation

❖ starvation.c

(1) sem_init(semaphore, p_shared, initial_value): 세마포어를 초기화 및 생성

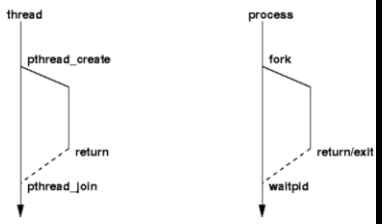
- semaphore: 세마포어의 주소
- p_shared

  0인 경우, 스레드 간에 공유

  0인 아닌 경우, 프로세스 간에 공유
- Initial_value: 공유 자원에 들어 갈 수 있는 스레드의 개수

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

- **새로운 프로세스를 생성하는 함수**
- 인자1 : 생성된 스레드의 ID를 저장할 변수의 포인터
- 인자2 : 스레드의 특성을 설정할 때 사용, 주로 NULL이 온다.
- 인자3 : 스레드가 생성되고 나서 실행될 함수
- 인자4 : 세번째 인자에서 호출되는 함수에 전달하고자 하는 인자



int **pthread_join**(pthread_t th, void **thread_return)

- 스레드가 종료될 때까지 호출한 스레드가 기다리도록 하게 함
- 인자1 : 스레드 ID. 이 ID가 종료할 때까지 실행을 지연
- 인자2 : 쓰레드 종료시 반환값

```c
int main(int argc, char *argv[])
{
    pthread_t *th;
    void *value;
    long i;

    if(argc < 3){
        fprintf(stderr, "%s parameter : nthread, worker_loop_cnt\n", argv[0]);
        exit(-1);
    }
    nthread = atoi(argv[1]);
    worker_loop_cnt = atoi(argv[2]);

    th = malloc(nthread * sizeof(pthread_t));
    if (th == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(-1);
    }

    sem_init(&mutex, 0, 1);

    for (i = 0; i < nthread; i++) {
        long *arg = malloc(sizeof(*arg));
        *arg = i;
        assert(pthread_create(&th[i], NULL, work, arg) == 0);
    }

    for (i = 0; i < nthread; i++)
        assert(pthread_join(th[i], &value) == 0);

    sem_destroy(&mutex);
    free(th);

    return 0;
}
```

Lock: Starvation

❖ 결과

```
[ec2-user@ip-172-31-15-105 day7]$ ./starvation 5
*arg: 0
*arg: 1
*arg: 2
*arg: 3
*arg: 4
*number: 15727264
Thread 15727264: count = 1
Thread 15727264: count = 2
Thread 15727264: count = 3
Thread 15727264: count = 4
Thread 15727264: count = 5
*number: 15727584
Thread 15727584: count = 6
Thread 15727584: count = 7
Thread 15727584: count = 8
Thread 15727584: count = 9
*number: 15726944
*number: 15726624
*number: 15725264
Thread 15727584: count = 10
Thread 15726944: count = 11
Thread 15726944: count = 12
Thread 15726944: count = 13
Thread 15726944: count = 14
Thread 15726944: count = 15
Thread 15726624: count = 16
Thread 15726624: count = 17
Thread 15726624: count = 18
Thread 15726624: count = 19
Thread 15726624: count = 20
Thread 15725264: count = 21
Thread 15725264: count = 22
Thread 15725264: count = 23
Thread 15725264: count = 24
Thread 15725264: count = 25
```