운영체제 실습과제

과제 #05

교수님: 정준호 교수님

강좌 명: 운영체제

학과: 컴퓨터공학과

학년: 3학년

학번: 2017112138

이름: 정여준

문제

식사하는 철학자 문제의 해결책은 경우에 따라서 기아문제가 발생할 수 있다. 철학자가 결코 굶어 죽지 않도록 제한된 대기 이후에는 식사 보장할 수 있는 알고리즘으로 개선하는 프로그램을 작성하시오.

<소스코드>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
#include <sys/time.h>

#define NUM_MEN  5
#define NLOOPS   5

#define LEFT(i)  ((i+NUM_MEN-1)%NUM_MEN)
#define RIGHT(i) ((i+1)%NUM_MEN)

enum { THINKING, EATING, HUNGRY };

sem_t    Philosopher[NUM_MEN]; // 현재 Philosopher[i]가 먹을 수 있는지 없는지에 대한 여부 semaphore.
sem_t    Mutex; // Critical section 보호를 위한 Mutex Semaphore.
int              State[NUM_MEN];

void ThreadUsleep(int usecs)
{
        pthread_cond_t   cond;
        pthread_mutex_t  mutex;
        struct timespec  ts;
        struct timeval   tv;

        if (pthread_cond_init(&cond, NULL) < 0) { //condition value를 생성.
                perror("pthread_cond_init");
                pthread_exit(NULL);
        }
        if (pthread_mutex_init(&mutex, NULL) < 0) { //mutex를 생성.
                perror("pthread_mutex_init");
                pthread_exit(NULL);
        }

        // 현재 시간(절대 시간)을 알아와서, 현재시간 + usec만큼의 시간을 구함.
        gettimeofday(&tv, NULL);
        ts.tv_sec = tv.tv_sec + usecs / 1000000;
        ts.tv_nsec = (tv.tv_usec + (usecs % 1000000)) * 1000;
        if (ts.tv_nsec >= 1000000000) {
                ts.tv_nsec -= 1000000000;
                ts.tv_sec++;
```

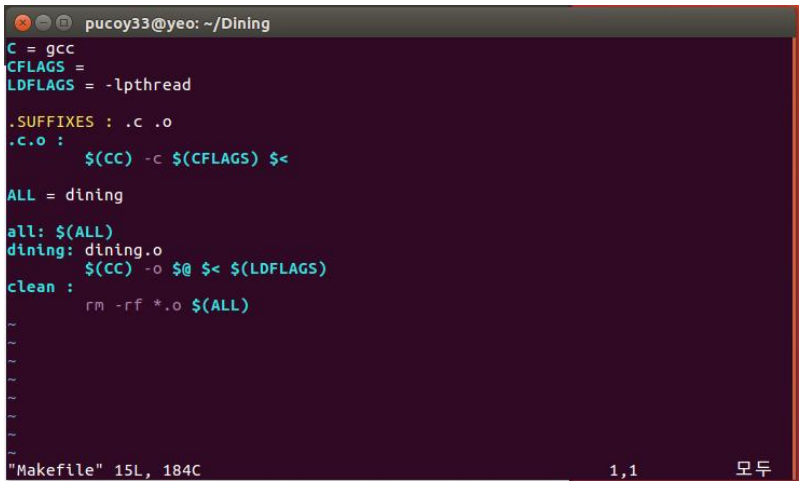
```
        }

        if (pthread_mutex_lock(&mutex) < 0) {
                perror("pthread_mutex_lock");
                pthread_exit(NULL);
        }
        if (pthread_cond_timedwait(&cond, &mutex, &ts) < 0) { // timedwait를 이용해서,
&ts( 현재시간 + usec만큼의 절대 시간)동안 cond가 발생하기를 기다리다가 절대시간이 경과되면
wait상태에서 빠져나옴.
                perror("pthread_cond_timedwait");
                pthread_exit(NULL);
        }

        if (pthread_cond_destroy(&cond) < 0) {
                perror("pthread_cond_destroy");
                pthread_exit(NULL);
        }
        if (pthread_mutex_destroy(&mutex) < 0) {
                perror("pthread_mutex_destroy");
                pthread_exit(NULL);
        }
}

void Thinking(int id)
{
        printf("Philosopher%d: Thinking.....₩n", id);
        ThreadUsleep((rand() % 200) * 10000);
        printf("Philosopher%d: Want to eat.....₩n", id);
}

void Eating(int id)
{
        printf("Philosopher%d: Eating.....₩n", id);
        ThreadUsleep((rand() % 100) * 10000);
}

void Test(int id) {
        if (State[id] == HUNGRY && State[LEFT(id)] != EATING && State[RIGHT(id)] != EATING) {
                State[id] = EATING;

                if (sem_post(&Philosopher[id]) < 0) {
                        perror("sem_post");
                        pthread_exit(NULL);
                }
        }
}

void PickUp(int id) {

        if (sem_wait(&Mutex) < 0) {
                perror("sem_wait");
                pthread_exit(NULL);
        }

        State[id] = HUNGRY;
```

```
        Test(id);

        if (sem_post(&Mutex) < 0) {
                perror("sem_post");
                pthread_exit(NULL);
        }

        if (sem_wait(&Philosopher[id]) < 0) {
                perror("sem_wait");
                pthread_exit(NULL);
        }
}

void PutDown(int id) {
        if (sem_wait(&Mutex) < 0) {
                perror("sem_wait");
                pthread_exit(NULL);
        }

        State[id] = THINKING;

        Test(LEFT(id)); // 내 왼쪽에 있는 애가 먹을 수 있는 상태면 signal해줌.
        Test(RIGHT(id)); // 내 오른쪽에 있는 애가 먹을 수 있는 상태면 signal해줌.

        if (sem_post(&Mutex) < 0) {
                perror("sem_post");
                pthread_exit(NULL);
        }
}

void DiningPhilosopher(int * pId) {
        int     i;
        int     id = *pId;

        for (i = 0; i < NLOOPS; i++) {
                Thinking(id); //랜덤한 시간동안 생각을 한다.

                PickUp(id);

                Eating(id);

                PutDown(id);
        }

        printf("Philosopher%d: thinking & eating %d times.....₩n", id, i);

        pthread_exit(NULL);
}

main() {
        pthread_t       tid[NUM_MEN]; //철학자 수 만큼의 pthread
        int             i, id[NUM_MEN];

        srand(0x8888);
```

```
        for (i = 0; i < NUM_MEN; i++) {
                if (sem_init(&Philosopher[i], 0, 0) < 0) { // Philosohper가 먹을 수 있는 상태
semaphore배열을 다 0으로 초기화.(다 먹을 수 없다고 가정)
                        perror("sem_init");
                        exit(1);
                }

                id[i] = i;
        }

        if (sem_init(&Mutex, 0, 1) < 0) { // Mutex를 1로 초기화.
                perror("sem_init");
                exit(1);
        }

        for (i = 0; i < NUM_MEN; i++) {
                if (pthread_create(&tid[i], NULL, (void *)DiningPhilosopher, (void *)&id[i]) <
0) {//스레드들을 생성하는데 스레드 함수는 DiningPhilosopher, 파라미터는 id[i]의 주소이다.
                        perror("pthread_create");
                        exit(1);
                }
        }

        for (i = 0; i < NUM_MEN; i++) {
                if (pthread_join(tid[i], NULL) < 0) {//pthread_join을 통해 pthread들이
끝나기를 기다린다.
                        perror("pthread_join");
                        exit(1);
                }
        }

        for (i = 0; i < NUM_MEN; i++) {
                if (sem_destroy(&Philosopher[i]) < 0) { // semaphore 제거_
                        perror("sem_destroy");
                }
        }

        if (sem_destroy(&Mutex) < 0) {
                perror("sem_destroy");
        }
}
```

일단 교착상태에 빠지지 않기 위해서 양 옆에 있는 사람이 모두 젓가락을 들지 않았을 경우 들도록 한다. 이런 상황에서 deadlock 상태는 제거되지만 기아 문제의 가능성은 아직 있다. 이러한 문제를 해결하기위해 시간을 통해서 시간이 경과되면 wait 상태에서 빠져나오게 한다. 그래서 현재 시간을 구해와 얼만큼의 시간이 소요됐는지 구하는 함수를 생성한다. 그리고 이 함수를 통해 시간이 경과될 경우 wait상태에서 빠져나오게 구현한다. 이렇게 하면 위에서 설정한 5명의 철학자 모두 5번씩 먹을 수 있게 되어 아무도 굶어 죽는 현상이 발생하지 않는다.

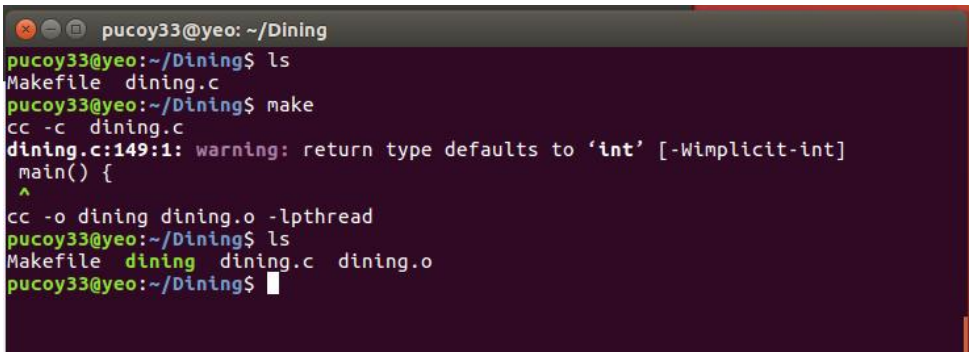
이렇게 생성한 파일을 저장해준다. 그리고 이러한 코드를 실행가능한 파일로 생성하기 위해서 Makefile을 생성해야한다.



<소스코드>

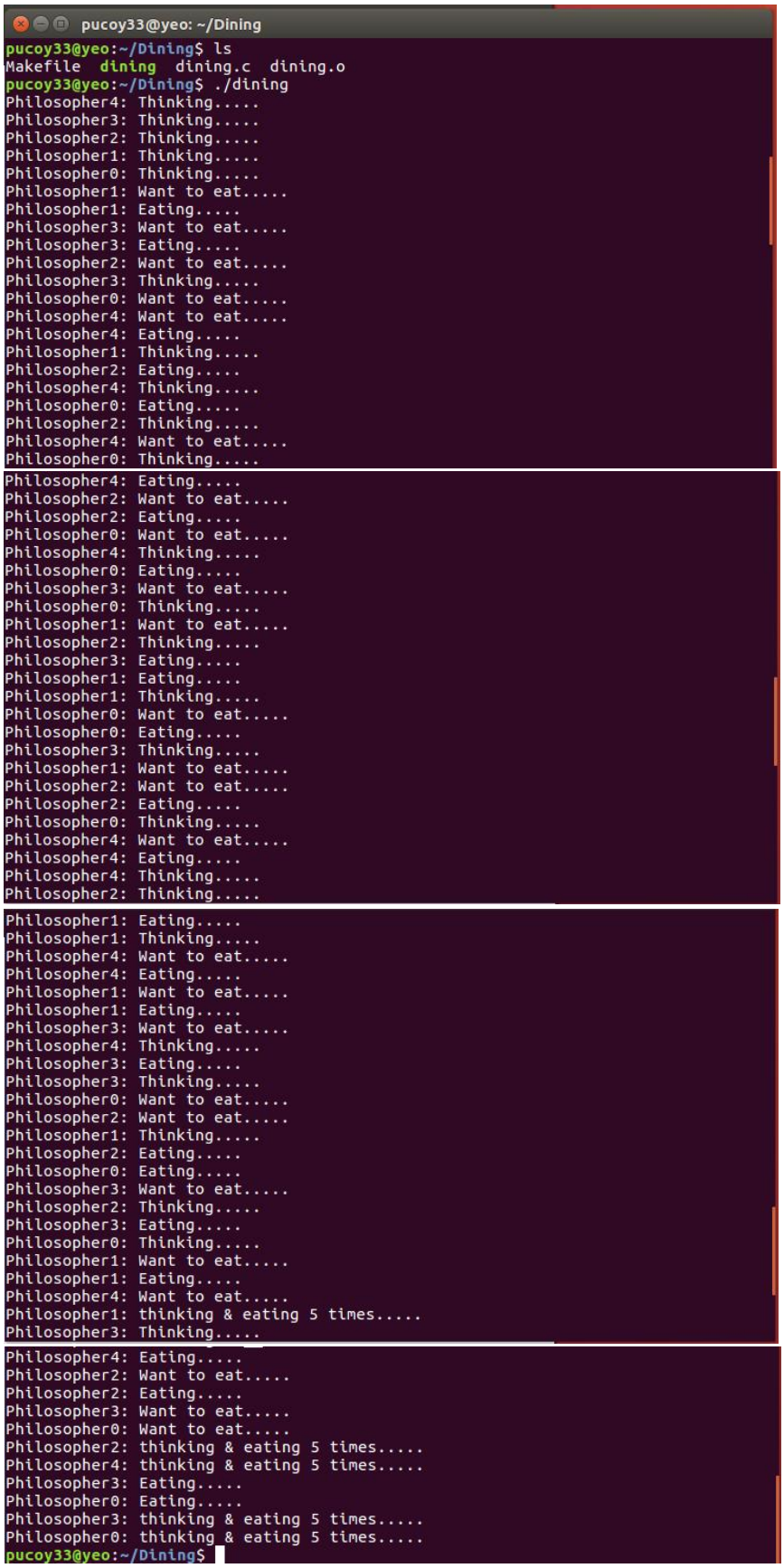
```
CC = gcc
CFLAGS =
LDFLAGS = -lpthread

.SUFFIXES :.c.o
.c.o :
        $(CC) - c $(CFLAGS) $ <

        ALL = dining

        all : $(ALL)
        dining: dining.o
        $(CC) - o $@ $ < $(LDFLAGS)

        clean :
        rm - rf * .o $(ALL)
```

이렇게 해서 실행가능한 파일 dining을 만들어 줄 Makefile을 생성한다.



make 명령어를 통해서 실행가능한 파일 dining이 생성된 것을 확인할 수 있다.

이를 실행해보자.


pucoy33@yeo: ~/Dining
pucoy33@yeo:~/Dining$ ls
Makefile dining dining.c dining.o
pucoy33@yeo:~/Dining$ ./dining
Philosopher4: Thinking.....
Philosopher3: Thinking.....
Philosopher2: Thinking.....
Philosopher1: Thinking.....
Philosopher0: Thinking.....
Philosopher1: Want to eat.....
Philosopher1: Eating.....
Philosopher3: Want to eat.....
Philosopher3: Eating.....
Philosopher2: Want to eat.....
Philosopher3: Thinking.....
Philosopher0: Want to eat.....
Philosopher4: Want to eat.....
Philosopher4: Eating.....
Philosopher1: Thinking.....
Philosopher2: Eating.....
Philosopher4: Thinking.....
Philosopher0: Eating.....
Philosopher2: Thinking.....
Philosopher4: Want to eat.....
Philosopher0: Thinking.....
Philosopher4: Eating.....
Philosopher2: Want to eat.....
Philosopher2: Eating.....
Philosopher0: Want to eat.....
Philosopher4: Thinking.....
Philosopher0: Eating.....
Philosopher3: Want to eat.....
Philosopher0: Thinking.....
Philosopher1: Want to eat.....
Philosopher2: Thinking.....
Philosopher3: Eating.....
Philosopher1: Eating.....
Philosopher1: Thinking.....
Philosopher0: Want to eat.....
Philosopher0: Eating.....
Philosopher3: Thinking.....
Philosopher1: Want to eat.....
Philosopher2: Want to eat.....
Philosopher2: Eating.....
Philosopher0: Thinking.....
Philosopher4: Want to eat.....
Philosopher4: Eating.....
Philosopher4: Thinking.....
Philosopher2: Thinking.....
Philosopher1: Eating.....
Philosopher1: Thinking.....
Philosopher4: Want to eat.....
Philosopher4: Eating.....
Philosopher1: Want to eat.....
Philosopher1: Eating.....
Philosopher3: Want to eat.....
Philosopher4: Thinking.....
Philosopher3: Eating.....
Philosopher3: Thinking.....
Philosopher0: Want to eat.....
Philosopher2: Want to eat.....
Philosopher1: Thinking.....
Philosopher2: Eating.....
Philosopher0: Eating.....
Philosopher3: Want to eat.....
Philosopher2: Thinking.....
Philosopher3: Eating.....
Philosopher0: Thinking.....
Philosopher1: Want to eat.....
Philosopher1: Eating.....
Philosopher4: Want to eat.....
Philosopher1: thinking & eating 5 times.....
Philosopher3: Thinking.....
Philosopher4: Eating.....
Philosopher2: Want to eat.....
Philosopher2: Eating.....
Philosopher3: Want to eat.....
Philosopher0: Want to eat.....
Philosopher2: thinking & eating 5 times.....
Philosopher4: thinking & eating 5 times.....
Philosopher3: Eating.....
Philosopher0: Eating.....
Philosopher3: thinking & eating 5 times.....
Philosopher0: thinking & eating 5 times.....
pucoy33@yeo:~/Dining$

위의 화면은 실행한 결과이다. 보면 처음에 5명의 철학자 모두 생각하는 것으로 초기화가 되고 1번 철학자부터 먹기 시작한다. 그럴 경우 3번 철학자가 먹을 수 있게 되고 먹는다. 이 다음부터는 먹고 싶은 사람이 출력이 되고 그 사람의 양 옆의 사람이 모두 생각하는 상태로 변경될경우 먹게 된다. 이렇게 반복하다 보면 1번 철학자부터 5번을 모두 먹게되고 다음으로 2번 4번 3번 0번 철학자가 모두 5번씩 먹는 것을 마치고 생각한느 상태로 변환된 후 실행이 종료된다. 이렇게 순서를 주고 돌아가면서 규칙에 맞게 먹게하고 시간이 초과할 경우 생각하는 상태로 변환하게 하므로써 교착상태에도 빠지지 않고 굶어죽는 현상도 없앨 수 있다.

<소감>

수업에서 배운 Deadlock상태를 해결하는 방법과 직접 기아문제를 해결하는 코드를 구현해보면서 식사하는 철학자들 문제에 대한 해답을 얻은 것 같아서 뿌듯합니다. 그리고 이러한 실습과제를 통해서 리눅스에서 사용하는 언어에 대한 지식이 늘어서 좋습니다. 또한 코드를 구현하는 실력도 증가하는 것 같아서 좋습니다. 수업에 대한 이해도와 저의 코딩 실력을 모두 향상시킬 수 있는 과제인 것 같습니다.