

# HW3: Deadlock - Dining Philosopher



과목명 | 운영체제

담당교수 | 황선태

학과 | 소프트웨어학부

학년 | 2학년

학번 | 20143050

이름 | 김현석

제출일 | 2017.05.14

# 1. Dead lock 구현 코드

## Phil\_A 코드

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
```

```
#define SEMPERM 0600
#define TRUE 1
#define FALSE 0
```

```
typedef union    _semun { //semaphore의구조체
int val;
```

```
struct semid_ds *buf;
```

```
ushort *array;
```

```
} semun
```

```
int initsem(key_t semkey, int n) { //semaphore를초기화시킬때사용
int status = 0, semid;
```

```

// 세마포어값을설정
if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT |
IPC_EXCL)) == -1)
{
if (errno == EEXIST)
semid = semget(semkey, 1, 0);
}
else
{
semun arg;
arg.val = n
// 세마포어제어( setval: 값설정)
status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
{
perror("initsem failed");
return (-1);
}
return (semid);
}

```

int p(int semid) { //p연산을하며acquire 할때사용된다.

```

struct sembuf p_buf;

```

```

p_buf.sem_num = 0;

```

```

p_buf.sem_op = -1;

```

```
p_buf.sem_flg = SEM_UNDO;
```

```
if (semop(semid, &p_buf, 1) == -1)
{
    printf("p(semid) failed");
    exit(1);
}
return (0);
}
```

```
int v(int semid) { //v연산을하며release 할때사용된다.
    struct sembuf v_buf;
```

```
v_buf.sem_num = 0;
```

```
v_buf.sem_op = 1;
```

```
v_buf.sem_flg = SEM_UNDO;
```

```
if (semop(semid, &v_buf, 1) == -1)
{
    printf("v(semid) failed");
    exit(1);
}
return (0);
}
```

```
// Shared variable by file
```

```
void reset(char *fileVar, int value) { // fileVar라는이름의텍스트화일을새로만들고0값을기록한다.
```

```
if (access(fileVar, F_OK) == -1) {
```

```

FILE *fp = fopen(fileVar, "w");
time_t timer;
time(&timer);
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),value);
fclose(fp);
}
}

```

void Store(char \*fileVar, int i) { // fileVar 화일끝에i 값을append한다

```

FILE *fp = fopen(fileVar, "a");
fprintf(fp, "PID:%d %d\n",getpid(), i);
fclose(fp);
}

```

int Load(char \*fileVar) { // fileVar 화일의마지막값을읽어온다.  
int num;

```

FILE *fp = fopen(fileVar, "r");

```

```

fseek(fp, -4, SEEK_END);

```

```

while (1) {
fscanf(fp, "%d", &num);
if (feof(fp)) {
break
}
}
fclose(fp);

```

```

return num;

```

```
}
```

```
void add(char *fileVar, int i) { // fileVar 파일의 마지막값을읽어서i를  
더한후에이를끝에append 한다.
```

```
FILE *fp = fopen(fileVar, "a");
```

```
time_t timer;
```

```
time(&timer);
```

```
int add_num = Load(fileVar);
```

```
add_num += i
```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),  
add_num);
```

```
fclose(fp);
```

```
}
```

```
void sub(char *fileVar, int i) { // fileVar 파일의 마지막값을읽어서i를  
뺀후에이를끝에append 한다.
```

```
FILE *fp = fopen(fileVar, "a");
```

```
int sub_num = Load(fileVar);
```

```
time_t timer;
```

```
time(&timer);
```

```
sub_num -= i
```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),
sub_num);
```

```
fclose(fp);
}
```

```
// Class Lock
```

```
typedef struct _lock { //lock의구조체semid로이루어졌다.
int semid;
} Lock
```

```
void initLock(Lock *l, key_t semkey) { //lock 초기화함수로세마포
를연결하며만약없다면초기값을1로주면서새로만들어서연결한다.
if ((l->semid = initsem(semkey, 1)) < 0)
exit(1);
}
```

```
void Acquire(Lock *l) { //p연산을이용해acquire를구현한다.
p(l->semid);
}
```

```
void Release(Lock *l) { //v연산을이용해release를구현한다.
v(l->semid);
}
```

```
// Class CondVar
```

```
typedef struct _cond { //condvar의구조체대기열과semid를포함한다.
int semid;
```

```
char* queueLength;  
} CondVar
```

```
void initCondVar(CondVar *c, key_t semkey, char* queueLength)  
{// queueLength를받아condvar queueLength로지정한다. 또한세마포  
를연결하는데없으면초기값을0로주면서새로만들어서연결한다.
```

```
c->queueLength = queueLength
```

```
reset(c->queueLength,0); // queueLength=0
```

```
if ((c->semid = initsem(semkey, 0)) < 0)  
exit(1);  
}
```

```
void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를  
구현한다. add를통해queueLength를증가시켜준다.  
add(c->queueLength, 1);  
Release(lock);  
p(c->semid);  
Acquire(lock);  
}
```

```
void Signal(CondVar *c) { //condvar를받아대기중인queueLength가  
있다면활성화시켜주고sub를통해queueLength를감소시켜준다.  
if (Load(c->queueLength) > 0) {  
v(c->semid);  
sub(c->queueLength, 1);
```



```
}  
}
```

```
void Broadcast(CondVar *c) { //condvar를받아대기중인  
queueLength가있다면"모두" 활성화시켜주고sub를통해queueLength를  
감소시켜준다.
```

```
while (Load(c->queueLength) > 0) {  
v(c->semid);  
sub(c->queueLength, 1);  
}  
}
```

```
void Take_R1(Lock *L1, CondVar *C1){ //R1 젓가락을가져오기위한  
함수
```

```
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준  
다.
```

```
while(Load("R1.txt")==0){ //남은자원이없으면대기한다.
```

```
printf("message: %d가R1을기다림\n",getpid());
```

```
Wait(C1,L1); //wait 상태로들어간다
```

```
printf("message: %d가R1을기다리다가깨어남\n",getpid());
```

```
}
```

```
Store("R1.txt",0); //젓가락을얻었으므로R1.txt에0을저장시켜준다.
```

```
printf("message: %d가R1을가져옴\n",getpid());
```

```
Release(L1); //Take_R1을수행하였으므로L1을Release 시켜준다.
```

```
}
```

```
void Take_R2(Lock *L2, CondVar *C2){ //R2 젓가락을가져오기위한  
함수
```

```
Acquire(L2); //L2에관하여acquire함으로써mutual exclusive 하게해준
```

다.

```
while(Load("R2.txt")==0){ //남은자원이없으면대기한다.
printf("message: %d가R2을기다림\n",getpid());
Wait(C2,L2); //wait 상태로들어간다.
printf("message: %d가R2을기다리다가깨어남\n",getpid());
}
Store("R2.txt",0); //젓가락을얻었으므로R2.txt에0을저장시켜준다.
printf("message: %d가R2을가져옴\n",getpid());
```

```
Release(L2);//Take_R2을수행하였으므로L2을Release 시켜준다.
}
```

```
void Put_R1(Lock *L1, CondVar *C1){//R1 젓가락을놓기위한함수식
사를다하였을때수행하게된다.
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준다.
Store("R1.txt",1); //젓가락을놓았으므로R1.txt 에1을저장시켜준다.
Signal(C1); //queueR1에대기중인철학자를깨워준다.
printf("message: %d가R1을내려둠\n",getpid());
Release(L1); //Put_R1을수행하였으므로L1을Release 시켜준다.
}
```

```
void Put_R2(Lock *L2, CondVar *C2){ //R2 젓가락을놓기위한함수
식사를다하였을때수행하게된다.
Acquire(L2); //L1에관하여acquire함으로써mutual exclusive 하게해준다.
Store("R2.txt",1); //젓가락을놓았으므로R2.txt 에1을저장시켜준다.
Signal(C2); //queueR1에대기중인철학자를깨워준다.
printf("message: %d가R2을내려둠\n",getpid());
Release(L2); //Put_R2을수행하였으므로L2을Release 시켜준다.
```

```
}
```

```
void Phil_A(Lock *L1, CondVar *C1, Lock *L2, CondVar *C2){ //A  
철학자가밥을먹기위해수행하는함수  
Take_R1(L1,C1); //R1를가져오기위한함수  
printf("message: %d가생각을시작함\n",getpid());  
sleep(1);  
printf("message: %d가생각을멈춤\n",getpid());  
Take_R2(L2, C2); //R2를가져오기위한함수  
printf("message: %d가먹기시작함\n",getpid());  
sleep(1);  
printf("message: %d가먹기를멈춤\n",getpid());  
Put_R1(L1,C1); //R2를내려놓기위한함수  
Put_R2(L2, C2); //R1를내려놓기위한함수  
}
```

```
int main() {  
    // 서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지  
    않게해야한다.
```

```
    // 실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을  
    사용해야한다.
```

```
    // $ ipcs                // 남아있는세마포확T인
```

```
    // $ ipcrm -s <semid>    // <semid>라는세마포제거
```

```
    key_t semkey = 0x100; //semkey값을지정한다.  
    key_t semkey2 = 0x200; //semkey값을지정한다.  
    key_t semkey3 = 0x300; //semkey값을지정한다.
```

key\_t semkey\_chopsticks1 = 0x400; //semkey\_chopsticks1 값을 지정한다.

key\_t semkey\_chopsticks2 = 0x500; //semkey\_chopsticks2 값을 지정한다.

key\_t semkey\_chopsticks3 = 0x600; //semkey\_chopsticks3 값을 지정한다.

reset("R1.txt",1); //R1.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

reset("R2.txt",1); //R2.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

reset("R3.txt",1); //R3.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

Lock lock1; //wait, acquire, release에사용된다

Lock lock2; //wait, acquire, release에사용된다

Lock lock3; //wait, acquire, release에사용된다

CondVar C1; //R1에사용되는condvar

CondVar C2; //R2에사용되는condvar

CondVar C3; //R3에사용되는condvar

initLock(&lock1, semkey); //lock 초기화함수

initLock(&lock2, semkey2); //lock 초기화함수

initLock(&lock3, semkey3); //lock 초기화함수

initCondVar(&C1, semkey\_chopsticks1, "queueR1.txt"); //R1에사용  
하는condvar 함수초기화이때queueR1의파일명을지정한다.

initCondVar(&C2, semkey\_chopsticks2, "queueR2.txt"); //R2에사용  
하는condvar 함수초기화이때queueR2의파일명을지정한다.

initCondVar(&C3, semkey\_chopsticks3, "queueR3.txt"); //R3에사용

하는condvar 함수초기화이때queueR3의파일명을지정한다.

```
for(int i=0;i<100;i++)  
    Phil_A(&lock1,&C1, &lock2, &C2);  
  
    return 0;  
}
```

# Phil\_B 코드

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define SEMPERM 0600
#define TRUE 1
#define FALSE 0

typedef union _semun { //semaphore의구조체
int val;

struct semid_ds *buf;

ushort *array;

} semun

int initsem(key_t semkey, int n) { //semaphore를초기화시킬때사용
int status = 0, semid;

// 세마포어값을설정
if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT |
```

```

IPC_EXCL)) == -1)
{
if (errno == EEXIST)
semid = semget(semkey, 1, 0);
}
else
{
semun arg;
arg.val = n
// 세마포어제어( setval: 값설정)
status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
{
perror("initsem failed");
return (-1);
}
return (semid);
}

```

int p(int semid) { //p연산을하며acquire 할때사용된다.

```

struct sembuf p_buf;

```

```

p_buf.sem_num = 0;

```

```

p_buf.sem_op = -1;

```

```

p_buf.sem_flg = SEM_UNDO;

```

```

if (semop(semid, &p_buf, 1) == -1)

```

```

{
printf("p(semid) failed");
exit(1);
}
return (0);
}

```

```

int v(int semid) { //v연산을하며release 할때사용된다.
struct sembuf v_buf;

```

```

v_buf.sem_num = 0;

```

```

v_buf.sem_op = 1;

```

```

v_buf.sem_flg = SEM_UNDO;

```

```

if (semop(semid, &v_buf, 1) == -1)
{
printf("v(semid) failed");
exit(1);
}
return (0);
}

```

```

// Shared variable by file

```

```

void reset(char *fileVar, int value) { // fileVar라는이름의텍스트화일
을새로만들고0값을기록한다.

```

```

if (access(fileVar, F_OK) == -1) {
FILE *fp = fopen(fileVar, "w");
time_t timer;
time(&timer);

```



```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),value);
fclose(fp);
}
}
```

```
void Store(char *fileVar, int i) { // fileVar 파일끝에i 값을append한
다
FILE *fp = fopen(fileVar, "a");
fprintf(fp, "PID:%d %d\n",getpid(), i);
fclose(fp);
}
```

```
int Load(char *fileVar) { // fileVar 파일의마지막값을읽어온다.
int num;
```

```
FILE *fp = fopen(fileVar, "r");
```

```
fseek(fp, -4, SEEK_END);
```

```
while (1) {
```

```
fscanf(fp, "%d", &num);
```

```
if (feof(fp)) {
```

```
break
```

```
}
```

```
}
```

```
fclose(fp);
```

```
return num;
```

```
}
```

```
void add(char *fileVar, int i) { // fileVar 파일의 마지막값을읽어서i를  
더한후에이를끝에append 한다.
```

```
FILE *fp = fopen(fileVar, "a");
```

```
time_t timer;
```

```
time(&timer);
```

```
int add_num = Load(fileVar);
```

```
add_num += i
```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),  
add_num);
```

```
fclose(fp);
```

```
}
```

```
void sub(char *fileVar, int i) { // fileVar 파일의 마지막값을읽어서i를  
뺀후에이를끝에append 한다.
```

```
FILE *fp = fopen(fileVar, "a");
```

```
int sub_num = Load(fileVar);
```

```
time_t timer;
```

```
time(&timer);
```

```
sub_num -= i
```

```
fprintf(fp, "PID: %d  time:  %s%d\n",  getpid(),  ctime(&timer),  
sub_num);
```

```
fclose(fp);
```

```
}
```

```
// Class Lock
```

```
typedef struct _lock { //lock의구조체semid로이루어졌다.
```

```
int semid;
```

```
} Lock
```

```
void initLock(Lock *l, key_t semkey) { //lock 초기화함수로세마포  
를연결하며만약없다면초기값을1로주면서새로만들어서연결한다.
```

```
if ((l->semid = initsem(semkey, 1)) < 0)
```

```
exit(1);
```

```
}
```

```
void Acquire(Lock *l) { //p연산을이용해acquire를구현한다.
```

```
p(l->semid);
```

```
}
```

```
void Release(Lock *l) { //v연산을이용해release를구현한다.
```

```
v(l->semid);
```

```
}
```

```
// Class CondVar
```

```
typedef struct _cond { //condvar의구조체대기열과semid를포함한다.
```

```
int semid;
```

```
char* queueLength;
```

```
} CondVar
```

```
void initCondVar(CondVar *c, key_t semkey, char* queueLength)
{
    // queueLength를받아condvar queueLength로지정한다. 또한세마포
    //를연결하는데없으면초기값을0로주면서새로만들어서연결한다.
    c->queueLength = queueLength
    reset(c->queueLength,0); // queueLength=0
    if ((c->semid = initsem(semkey, 0)) < 0)
        exit(1);
}
```

```
void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를
    구현한다. add를통해queueLength를증가시켜준다.
    add(c->queueLength, 1);
    Release(lock);
    p(c->semid);
    Acquire(lock);
}
```

```
void Signal(CondVar *c) { //condvar를받아대기중인queueLength가
    있다면활성화시켜주고sub를통해queueLength를감소시켜준다.
    if (Load(c->queueLength) > 0) {
        v(c->semid);
        sub(c->queueLength, 1);
    }
}
```

```
void Broadcast(CondVar *c) { //condvar를받아대기중인
```

queueLength가있다면"모두"   활성화시켜주고sub를통해queueLength를 감소시켜준다.

```
while (Load(c->queueLength) > 0) {  
v(c->semid);  
sub(c->queueLength, 1);  
}  
}
```

void Take\_R2(Lock \*L2, CondVar \*C2){ //R2 젓가락을가져오기위한 함수

Acquire(L2);//L2에관하여acquire함으로써mutual exclusive 하게해준다.

while(Load("R2.txt")==0){ //만약자원이없다면대기한다.

printf("message: %d가R2을기다림\n",getpid());

Wait(C2,L2); //wait 상태로들어간다.

printf("message: %d가R2을기다리다가깨어남\n",getpid());

}

Store("R2.txt",0); //젓가락을얻었으므로R2.txt에0을저장시켜준다

printf("message: %d가R2을가져옴\n",getpid());

Release(L2); //Take\_R2을수행하였으므로L2을Release 시켜준다.

}

void Take\_R3(Lock \*L3, CondVar \*C3){//R3 젓가락을가져오기위한 함수

Acquire(L3);//L3에관하여acquire함으로써mutual exclusive 하게해준다.

while(Load("R3.txt")==0){ //만약자원이없으면대기상태에들어간다.

printf("message: %d가R3을기다림\n",getpid());

Wait(C3,L3); //wait 상태로들어간다.

printf("message: %d가R3을기다리다가깨어남\n",getpid());

```

}
Store("R3.txt",0); //젓가락을얻었으므로R3.txt에0을저장시켜준다
printf("message: %d가R3을가져옴\n",getpid());

Release(L3); //Take_R3을수행하였으므로L3을Release 시켜준다.
}

```

```

void Put_R2(Lock *L2, CondVar *C2){ //R2 젓가락을놓기위한함수
식사를다하였을때수행하게된다.
Acquire(L2); //L2에관하여acquire함으로써mutual exclusive 하게해준다.
Store("R2.txt",1); //젓가락을놓았으므로R2.txt 에1을저장시켜준다.
Signal(C2); //queueR2에대기중인철학자를깨워준다.
printf("message: %d가R2을내려둠\n",getpid());
Release(L2); //Put_R2을수행하였으므로L1을Release 시켜준다.
}

```

```

void Put_R3(Lock *L3, CondVar *C3){ //R3 젓가락을놓기위한함수
식사를다하였을때수행하게된다.
Acquire(L3); //L1에관하여acquire함으로써mutual exclusive 하게해준다.
Store("R3.txt",1); //젓가락을놓았으므로R3.txt 에1을저장시켜준다.
Signal(C3); //위에서처럼자원을젓가락을놓았으므로전체자원의개수를증가시켜준다.
printf("message: %d가R3을내려둠\n",getpid());
Release(L3); //Put_R3을수행하였으므로L1을Release 시켜준다.
}

```

```

void Phil_B(Lock *L2, CondVar *C2,Lock *L3, CondVar *C3){
Take_R2(L2,C2); //R2를가져오기위한함수
printf("message: %d가생각을시작함\n",getpid());

```

```

sleep(1);
printf("message: %d가생각을멈춤\n",getpid());
Take_R3(L3, C3); //R3를가져오기위한함수
printf("message: %d가먹기시작함\n",getpid());
sleep(1);
printf("message: %d가먹기를멈춤\n",getpid());
Put_R2(L2,C2); //R2를내려놓기위한함수
Put_R3(L3,C3); //R3를내려놓기위한함수
}

```

```

int main() {
//   서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지않
//   게해야한다.

```

```

//   실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을사
//   용해야한다.

```

```

//   $ ipcs           // 남아있는세마포확T인

```

```

//   $ ipcrm -s <semid>   // <semid>라는세마포제거

```

```

key_t semkey = 0x100; //semkey값을지정한다.
key_t semkey2 = 0x200; //semkey값을지정한다.
key_t semkey3 = 0x300; //semkey값을지정한다.
key_t semkey_chopsticks1 = 0x400; //semkey_chopsticks1 값을지
//   정한다.
key_t semkey_chopsticks2 = 0x500; //semkey_chopsticks2 값을지
//   정한다.
key_t semkey_chopsticks3 = 0x600; //semkey_chopsticks3 값을지
//   정한다.

```

reset("R1.txt",1); //R1.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

reset("R2.txt",1); //R2.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

reset("R3.txt",1); //R3.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

Lock lock1; //wait, acquire, release에사용된다

Lock lock2; //wait, acquire, release에사용된다

Lock lock3; //wait, acquire, release에사용된다

CondVar C1; //R1에사용되는condvar

CondVar C2; //R2에사용되는condvar

CondVar C3; //R3에사용되는condvar

initLock(&lock1, semkey); //lock 초기화함수

initLock(&lock2, semkey2); //lock 초기화함수

initLock(&lock3, semkey3); //lock 초기화함수

initCondVar(&C1, semkey\_chopsticks1, "queueR1.txt"); //R1에사용  
하는condvar 함수초기화이때queueR1의파일명을지정한다.

initCondVar(&C2, semkey\_chopsticks2, "queueR2.txt"); //R2에사용  
하는condvar 함수초기화이때queueR2의파일명을지정한다.

initCondVar(&C3, semkey\_chopsticks3, "queueR3.txt"); //R3에사용  
하는condvar 함수초기화이때queueR3의파일명을지정한다.

for(int i=0;i<100;i++)

Phil\_B(&lock2,&C2, &lock3, &C3);

return 0;}



# Phil\_C 코드

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define SEMPERM 0600
#define TRUE 1
#define FALSE 0

typedef union _semun { //semaphore의구조체
int val;

struct semid_ds *buf;

ushort *array;

} semun

int initsem(key_t semkey, int n) { //semaphore를초기화시킬때사용
int status = 0, semid;

// 세마포어값을설정
if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT |
```

```

IPC_EXCL)) == -1)
{
if (errno == EEXIST)
semid = semget(semkey, 1, 0);
}
else
{
semun arg;
arg.val = n
// 세마포어제어( setval: 값설정)
status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
{
perror("initsem failed");
return (-1);
}
return (semid);
}

```

int p(int semid) { //p연산을하며acquire 할때사용된다.

```

struct sembuf p_buf;

```

```

p_buf.sem_num = 0;

```

```

p_buf.sem_op = -1;

```

```

p_buf.sem_flg = SEM_UNDO;

```

```

if (semop(semid, &p_buf, 1) == -1)

```

```

{
printf("p(semid) failed");
exit(1);
}
return (0);
}

```

```

int v(int semid) { //v연산을하며release 할때사용된다.
struct sembuf v_buf;

```

```

v_buf.sem_num = 0;

```

```

v_buf.sem_op = 1;

```

```

v_buf.sem_flg = SEM_UNDO;

```

```

if (semop(semid, &v_buf, 1) == -1)
{
printf("v(semid) failed");
exit(1);
}
return (0);
}

```

```

// Shared variable by file

```

```

void reset(char *fileVar, int value) { // fileVar라는이름의텍스트화일
을새로만들고0값을기록한다.

```

```

if (access(fileVar, F_OK) == -1) {
FILE *fp = fopen(fileVar, "w");
time_t timer;
time(&timer);

```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),value);
fclose(fp);
}
}
```

```
void Store(char *fileVar, int i) { // fileVar 화일끝에i 값을append한
다
FILE *fp = fopen(fileVar, "a");
fprintf(fp, "PID:%d %d\n",getpid(), i);
fclose(fp);
}
```

```
int Load(char *fileVar) { // fileVar 화일의마지막값을읽어온다.
int num;
```

```
FILE *fp = fopen(fileVar, "r");
```

```
fseek(fp, -4, SEEK_END);
```

```
while (1) {
fscanf(fp, "%d", &num);
if (feof(fp)) {
break
}
}
fclose(fp);

return num;
}
```

```
void add(char *fileVar, int i) { // fileVar 화일의마지막값을읽어서i를
```

더한 후에 이를 끝에 append 한다.

```
FILE *fp = fopen(fileVar, "a");
```

```
time_t timer;
```

```
time(&timer);
```

```
int add_num = Load(fileVar);
```

```
add_num += i
```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),  
add_num);
```

```
fclose(fp);
```

```
}
```

```
void sub(char *fileVar, int i) { // fileVar 파일의 마지막 값을 읽어서 i를  
빼한 후에 이를 끝에 append 한다.
```

```
FILE *fp = fopen(fileVar, "a");
```

```
int sub_num = Load(fileVar);
```

```
time_t timer;
```

```
time(&timer);
```

```
sub_num -= i
```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),  
sub_num);
```

```
fclose(fp);  
}
```

```
// Class Lock
```

```
typedef struct _lock { //lock의구조체semid로이루어졌다.  
int semid;  
} Lock
```

```
void initLock(Lock *l, key_t semkey) { //lock 초기화함수로세마포  
를연결하며만약없다면초기값을1로주면서새로만들어서연결한다.  
if ((l->semid = initsem(semkey, 1)) < 0)  
exit(1);  
}
```

```
void Acquire(Lock *l) { //p연산을이용해acquire를구현한다.  
p(l->semid);  
}
```

```
void Release(Lock *l) { //v연산을이용해release를구현한다.  
v(l->semid);  
}
```

```
// Class CondVar
```

```
typedef struct _cond { //condvar의구조체대기열과semid를포함한다.  
int semid;  
char* queueLength;  
} CondVar
```

```

void initCondVar(CondVar *c, key_t semkey, char* queueLength)
{
    // queueLength를받아condvar queueLength로지정한다. 또한세마포
    를연결하는데없으면초기값을0로주면서새로만들어서연결한다.
    c->queueLength = queueLength
    reset(c->queueLength,0); // queueLength=0
    if ((c->semid = initsem(semkey, 0)) < 0)
        exit(1);
}

```

```

void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를
구현한다. add를통해queueLength를증가시켜준다.
add(c->queueLength, 1);
Release(lock);
p(c->semid);
Acquire(lock);
}

```

```

void Signal(CondVar *c) { //condvar를받아대기중인queueLength가
있다면활성화시켜주고sub를통해queueLength를감소시켜준다.
if (Load(c->queueLength) > 0) {
    v(c->semid);
    sub(c->queueLength, 1);
}
}

```

```

void Broadcast(CondVar *c) { //condvar를받아대기중인
queueLength가있다면"모두" 활성화시켜주고sub를통해queueLength를
감소시켜준다.
while (Load(c->queueLength) > 0) {
    v(c->semid);
    sub(c->queueLength, 1);
}
}

```

```
}  
}
```

```
void Take_R1(Lock *L1, CondVar *C1){ //R1 젓가락을가져오기위한  
함수  
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준  
다.
```

```
while(Load("R1.txt") == 0){ //만약자원이없으면대기상태로들어간다.  
printf("message: %d가R1을기다림\n",getpid());  
Wait(C1,L1); //wait 상태로들어간다.  
printf("message: %d가R1을기다리다가깨어남\n",getpid());  
}  
Store("R1.txt",0);  
printf("message: %d가R1을가져옴\n",getpid());
```

```
Release(L1); //Take_R1을수행하였으므로L1을Release 시켜준다.  
}
```

```
void Take_R3(Lock *L3, CondVar *C3){  
Acquire(L3); //L3에관하여acquire함으로써mutual exclusive 하게해준  
다.  
while(Load("R3.txt")==0){ //만약자원이없으면대기상태로들어간다.  
printf("message: %d가R3을기다림\n",getpid());  
Wait(C3,L3); //wait 상태로들어간다.  
printf("message: %d가R3을기다리다가깨어남\n",getpid());  
}  
Store("R3.txt",0); //젓가락을얻었으므로R3.txt에0을저장시켜준다.  
printf("message: %d가R3을가져옴\n",getpid());
```

```
Release(L3); //Take_R3을수행하였으므로L3을Release 시켜준다.
```



```
}
```

```
void Put_R1(Lock *L1, CondVar *C1){ //R1 젓가락을놓기위한함수  
식사를다하였을때수행하게된다.
```

```
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준  
다.
```

```
Store("R1.txt",1); //젓가락을놓았으므로R1.txt 에1을저장시켜준다.
```

```
Signal(C1);//queueR1에대기중인철학자를깨워준다.
```

```
printf("message: %d가R1을내려둠\n",getpid());
```

```
Release(L1); //Put_R1을수행하였으므로L1을Release 시켜준다.
```

```
}
```

```
void Put_R3(Lock *L3, CondVar *C3){ //R3 젓가락을놓기위한함수  
식사를다하였을때수행하게된다.
```

```
Acquire(L3); //L3에관하여acquire함으로써mutual exclusive 하게해준  
다.
```

```
Store("R3.txt",1);//젓가락을놓았으므로R3.txt 에1을저장시켜준다.
```

```
Signal(C3); //queueR3에대기중인철학자를깨워준다.
```

```
printf("message: %d가R3을내려둠\n",getpid());
```

```
Release(L3); //Put_R3을수행하였으므로L3을Release 시켜준다.
```

```
}
```

```
void Phil_C(Lock *L1, CondVar *C1,Lock *L3, CondVar *C3){//C  
철학자가밥을먹기위해수행하는함수
```

```
Take_R3(L3, C3); //R3를가져오기위한함수
```

```
printf("message: %d가생각을시작함\n",getpid());
```

```
sleep(1);
```

```
printf("message: %d가생각을멈춤\n",getpid());
```

```
Take_R1(L1,C1); //R1를가져오기위한함수
```

```
printf("message: %d가먹기시작함\n",getpid());
```

```
sleep(1);
```

```
printf("message: %d가먹기를멈춤\n",getpid());
Put_R3(L3,C3); //R3를내려놓기위한함수
Put_R1(L1,C1); //R1를내려놓기위한함수
}
//Phil_B(), Phil_C()도구현
```

```
int main() {
// 서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지않
게해야한다.
```

```
// 실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을
사용해야한다.
```

```
// $ ipcs // 남아있는세마포확T인
```

```
// $ ipcrm -s <semid> // <semid>라는세마포제거
```

```
key_t semkey = 0x100; //semkey값을지정한다.
key_t semkey2 = 0x200; //semkey값을지정한다.
key_t semkey3 = 0x300; //semkey값을지정한다.
key_t semkey_chopsticks1 = 0x400; //semkey_chopsticks1 값을지
정한다.
key_t semkey_chopsticks2 = 0x500; //semkey_chopsticks2 값을지
정한다.
key_t semkey_chopsticks3 = 0x600; //semkey_chopsticks3 값을지
정한다.
```

```
reset("R1.txt",1); //R1.txt파일생성만약있다면생성하지않는다. 이때1이
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.
```

reset("R2.txt",1); //R2.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

reset("R3.txt",1); //R3.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

Lock lock1; //wait, acquire, release에사용된다

Lock lock2; //wait, acquire, release에사용된다

Lock lock3; //wait, acquire, release에사용된다

CondVar C1; //R1에사용되는condvar

CondVar C2; //R2에사용되는condvar

CondVar C3; //R3에사용되는condvar

initLock(&lock1, semkey); //lock 초기화함수

initLock(&lock2, semkey2); //lock 초기화함수

initLock(&lock3, semkey3); //lock 초기화함수

initCondVar(&C1, semkey\_chopsticks1, "queueR1.txt"); //R1에사용  
하는condvar 함수초기화이때queueR1의파일명을지정한다.

initCondVar(&C2, semkey\_chopsticks2, "queueR2.txt"); //R2에사용  
하는condvar 함수초기화이때queueR2의파일명을지정한다.

initCondVar(&C3, semkey\_chopsticks3, "queueR3.txt"); //R3에사용  
하는condvar 함수초기화이때queueR3의파일명을지정한다.

for(int i=0;i<100;i++)

Phil\_C(&lock1,&C1, &lock3, &C3);

return 0;

}

## 2.Prevention 구현 코드

Phil\_A 함수 부분만 수정하였기 때문에 다른 부분은 위와 동일하여 Phil\_A 함수 부분만 올리도록 하겠습니다.

```
void Phil_A(Lock *L1, CondVar *C1, Lock *L2, CondVar *C2){//A
철학자가밥을먹기위해수행하는함수
Take_R2(L2, C2); //R1를가져오기위한함수
printf("message: %d가생각을시작함\n",getpid());
sleep(1);
printf("message: %d가생각을멈춤\n",getpid());
Take_R1(L1,C1); //R2를가져오기위한함수
printf("message: %d가먹기시작함\n",getpid());
sleep(1);
printf("message: %d가먹기를멈춤\n",getpid());
Put_R2(L2,C2); //R2를내려놓기위한함수
Put_R1(L1,C1); //R1를내려놓기위한함수
}
```

# 3.Avoiding 구현 코드

## Phil\_A 코드

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define SEMPERM 0600
#define TRUE 1
#define FALSE 0

typedef union _semun { //semaphore의구조체
int val;
struct semid_ds *buf;
ushort *array
} semun;

int initsem(key_t semkey, int n) { //semaphore를초기화시킬때사용
int status = 0, semid;
// 세마포어값을설정
if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT |
IPC_EXCL)) == -1)
```

```

{
if (errno == EEXIST)
semid = semget(semkey, 1, 0);
}
else
{
semun arg;
arg.val = n;
// 세마포어제어( setval: 값설정)
status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
{
perror("initsem failed");
return (-1);
}
return (semid);
}

```

```

int p(int semid) { //p연산을하며acquire 할때사용된다.
struct sembuf p_buf;
p_buf.sem_num = 0;
p_buf.sem_op = -1;
p_buf.sem_flg = SEM_UNDO;
if (semop(semid, &p_buf, 1) == -1)
{
printf("p(semid) failed");
exit(1);
}
return (0);
}

```

```
int v(int semid) { //v연산을하며release 할때사용된다.
```

```
struct sembuf v_buf;
```

```
v_buf.sem_num = 0;
```

```
v_buf.sem_op = 1;
```

```
v_buf.sem_flg = SEM_UNDO;
```

```
if (semop(semid, &v_buf, 1) == -1)
```

```
{
```

```
printf("v(semid) failed");
```

```
exit(1);
```

```
}
```

```
return (0);
```

```
}
```

```
// Shared variable by file
```

```
void reset(char *fileVar,int i) { // fileVar라는이름의텍스트화일을새로  
만들고0값을기록한다.
```

```
if (access(fileVar, F_OK) == -1) {
```

```
FILE *fp = fopen(fileVar, "w");
```

```
time_t timer;
```

```
time(&timer);
```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),i);
```

```
fclose(fp);
```

```
}
```

```
}
```

```
void Store(char *fileVar, int i) { // fileVar 화일끝에i 값을append한  
다
```

```
FILE *fp = fopen(fileVar, "a");
```

```
fprintf(fp, "PID: %d %d\n",getpid(), i);
```

```
fclose(fp);
```

```
}
```

```
int Load(char *fileVar) { // fileVar 화일의마지막값을읽어온다.
```

```
int num;
```

```
FILE *fp = fopen(fileVar, "r");
```

```
fseek(fp, -4, SEEK_END);
```

```
while (1) {
```

```
fscanf(fp, "%d", &num);
```

```
if (feof(fp)) {
```

```
break
```

```
}
```

```
}
```

```
fclose(fp);
```

```
return num;
```

```
}
```

```
void add(char *fileVar, int i) { // fileVar 화일의마지막값을읽어서i를  
더한후에이를끝에append 한다.
```

```
FILE *fp = fopen(fileVar, "a");
```

```
time_t timer;
```

```
time(&timer);
```

```
int add_num = Load(fileVar);
```

```
add_num += i;
```

```
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),  
add_num);
```

```
fclose(fp);
```

```
}
```

```
void sub(char *fileVar, int i) { // fileVar 화일의마지막값을읽어서i를  
뺀후에이를끝에append 한다.
```



```

FILE *fp = fopen(fileVar, "a");
int sub_num = Load(fileVar);
time_t timer;
time(&timer);
sub_num -= i;
fprintf(fp, "PID: %d time: %s %d\n", getpid(), ctime(&timer),
sub_num);
fclose(fp);

}
// Class Lock
typedef struct _lock { //lock의구조체semid로이루어졌다.
int semid;
} Lock;

void initLock(Lock *l, key_t semkey) { //lock 초기화함수로세마포
를연결하며만약없다면초기값을1로주면서새로만들어서연결한다.
if ((l->semid = initsem(semkey, 1)) < 0)
exit(1);
}

void Acquire(Lock *l) { //p연산을이용해acquire를구현한다.
p(l->semid);
}

void Release(Lock *l) { //v연산을이용해release를구현한다.

v(l->semid);
}

// Class CondVar

```

```
typedef struct _cond { //condvar의구조체대기열과semid를포함한다.  
int semid;  
char* queueLength;  
} CondVar;
```

```
void initCondVar(CondVar *c, key_t semkey, char* queueLength)  
{// queueLength를받아condvar queueLength로지정한다. 또한세마포  
를연결하는데없으면초기값을0로주면서새로만들어서연결한다.  
c->queueLength = queueLength;  
reset(c->queueLength,0); // queueLength=0  
if ((c->semid = initsem(semkey, 0)) < 0)  
exit(1);  
}
```

```
void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를  
구현한다. add를통해queueLength를증가시켜준다.  
add(c->queueLength, 1);  
Release(lock);  
p(c->semid);  
Acquire(lock);  
}
```

```
void Signal(CondVar *c) { //condvar를받아대기중인queueLength가  
있다면활성화시켜주고sub를통해queueLength를감소시켜준다.  
if (Load(c->queueLength) > 0) {  
v(c->semid);  
sub(c->queueLength, 1);  
}  
}
```

```
void Broadcast(CondVar *c) { //condvar를받아대기중인
```

queueLength가있다면"모두"   활성화시켜주고sub를통해queueLength를 감소시켜준다.

```
while (Load(c->queueLength) > 0) {  
v(c->semid);  
sub(c->queueLength, 1);  
}  
}
```

void Take\_R1(Lock \*L1,CondVar \*C1) { //R1 젓가락을가져오기위한 함수

Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준다.

```
while ((Load("R1.txt") == 0) ||((Load("banker_resource.txt")<2) &&  
(Load("state.txt")==0))) {
```

//만약자원이없거나, 남은자원이2개보다작고어떤철학자도식사를하지않는다면이때wait 상태에들어가야한다.

```
printf("message: %d R1을기다림\n",getpid());
```

Wait(C1,L1); //wait 상태로들어간다.

```
printf("message: %d R1을기다리다가깨어남\n",getpid());
```

```
}
```

Store("R1.txt", 0); //젓가락을얻었으므로R1.txt에0을저장시켜준다.

sub("banker\_resource.txt",1); //젓가락을A 철학자가얻음으로써전체 resource는감소한다.

```
printf("message: %d R1을얻음\n",getpid());
```

Release(L1); //Take\_R1을수행하였으므로L1을Release 시켜준다.

```
}
```

void Take\_R2(Lock \*L1,CondVar \*C2) { //R2 젓가락을가져오기위한 함수

Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준다.

```

while (Load("R2.txt") == 0 ) {
/*만약자원이없으면대기상태에들어가야한다.   이때는이미젓가락즉자원을
하나얻은상태이므로
이때는자원의개수가2보다작은상태임을고려할필요가없어진다.*/
printf("message: %d R2을기다림\n",getpid());
Wait(C2,L1); //wait 상태로들어간다.
printf("message: %d R2을기다리다가깨어남\n",getpid());
}
Store("R2.txt", 0); //젓가락을얻었으므로R2.txt에0을저장시켜준다.
    sub("banker_resource.txt",1); //젓가락을A 철학자가얻음으로써전체
resource는감소한다.
add("state.txt",1);    //이때A철학자가젓가락을두개얻어서밥을먹을수있으
므로state.txt에1을저장하여식사를하였음을나타낸다.
printf("message: %d R2을얻음\n",getpid());
Release(L1); //Take_R2을수행하였으므로L1을Release 시켜준다.
}

```

```

void Put_R1(Lock *L1,CondVar *C1) { //R1 젓가락을놓기위한함수
식사를다하였을때수행하게된다.
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준
다.
Store("R1.txt", 1); //젓가락을놓았으므로R1.txt 에1을저장시켜준다.
    add("banker_resource.txt",1); //위에서처럼자원을젓가락을놓았으
므로전체자원의개수를증가시켜준다.
sub("state.txt",1);    //젓가락을내려놓는다는것은식사를마쳤다는의미이기
때문에state에1 감소시켜준다.
    Signal(C1); //queueR1에대기중인철학자를깨워준다.
    printf("message: %d R1을내려놓는다.\n",getpid());
Release(L1); //Put_R1을수행하였으므로L1을Release 시켜준다.
}

```

```

void Put_R2(Lock *L1,CondVar *C2) { //R2 젓가락을놓기위한함수
식사를다하였을때수행하게된다.
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준다.
Store("R2.txt", 1); //젓가락을놓았으므로R2.txt 에1을저장시켜준다.
    add("banker_resource.txt",1); //위에서처럼자원을젓가락을놓았으므로전체자원의개수를증가시켜준다.
Signal(C2); //queueR1에대기중인철학자를깨워준다.
    printf("message: %d R2을내려놓는다.\n",getpid());
Release(L1); //Put_R2을수행하였으므로L1을Release 시켜준다.
}

```

```

void Phil_A(Lock *L1,CondVar *C1, CondVar *C2) { //A 철학자가
밥을먹기위해수행하는함수
    Take_R1(L1, C1); //R1를가져오기위한함수
    printf("message: %d 생각을시작함\n",getpid());
    sleep(1);
    printf("message: %d 생각을멈춤\n",getpid());
    Take_R2(L1, C2); //R2를가져오기위한함수
    printf("message: %d 가먹기시작함\n", getpid());
    sleep(1);
    printf("message: %d 다먹었음\n",getpid());
    Put_R1(L1,C1); //R1를내려놓기위한함수
    Put_R2(L1,C2); //R2를내려놓기위한함수
}

```

```

int main(int argc, char * argv[]) {
    // 서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지
    않게해야한다.

```

```

    // 실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을

```

사용해야한다.

```
// $ ipcs // 남아있는세마포획T인
```

```
// $ ipcrm -s <semid> // <semid>라는세마포제거
```

```
key_t semkey = 0x100; //semkey값을지정한다.
```

```
key_t semkey_chopsticks1 = 0x200; //semkey_chopsticks1 값을지정한다.
```

```
key_t semkey_chopsticks2 = 0x300; //semkey_chopsticks2 값을지정한다.
```

```
key_t semkey_chopsticks3 = 0x400; //semkey_chopsticks3 값을지정한다.
```

```
reset("banker_resource.txt",3); //전체resource 관리를위해  
banker_resource.txt 생성
```

```
reset("state.txt",0); //철학자가밥을먹는지보기위한state.txt 생성
```

```
reset("R1.txt",1); //R1.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.
```

```
reset("R2.txt",1); //R2.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.
```

```
reset("R3.txt",1); //R3.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.
```

```
Lock lock1; //wait, acquire, release에사용된다
```

```
CondVar C1; //R1에사용되는condvar
```

```
CondVar C2; //R2에사용되는condvar
```

```
CondVar C3; //R3에사용되는condvar
```

```

initLock(&lock1, semkey); //lock 초기화함수
initCondVar(&C1, semkey_chopsticks1, "queueR1.txt"); //R1에사용
하는condvar 함수초기화이때queueR1의파일명을지정한다.
initCondVar(&C2, semkey_chopsticks2, "queueR2.txt"); //R2에사용
하는condvar 함수초기화이때queueR2의파일명을지정한다.
initCondVar(&C3, semkey_chopsticks3, "queueR3.txt"); //R3에사용
하는condvar 함수초기화이때queueR3의파일명을지정한다.

for(int i=0;i<100;i++)
Phil_A(&lock1,&C1, &C2);

return 0;
}

```

## Phil\_B 코드

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define SEMPERM 0600
#define TRUE 1
#define FALSE 0

```

```

typedef union    _semun { //semaphore의구조체
int val;
struct semid_ds *buf;
ushort *array;
} semun

```

```

int initsem(key_t semkey, int n) { //semaphore를초기화시킬때사용
int status = 0, semid;
// 세마포어값을설정
if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT |
IPC_EXCL)) == -1)
{
if (errno == EEXIST)
semid = semget(semkey, 1, 0);
}
else
{
semun arg;
arg.val = n
// 세마포어제어( setval: 값설정)
status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
{
perror("initsem failed");
return (-1);
}
return (semid);
}

```

```

int p(int semid) { //p연산을하며acquire 할때사용된다.

```



```

struct sembuf p_buf;
p_buf.sem_num = 0;
p_buf.sem_op = -1;
p_buf.sem_flg = SEM_UNDO;
if (semop(semid, &p_buf, 1) == -1)
{
printf("p(semid) failed");
exit(1);
}
return (0);
}

```

int v(int semid) { //v연산을하며release 할때사용된다.

```

struct sembuf v_buf;
v_buf.sem_num = 0;
v_buf.sem_op = 1;
v_buf.sem_flg = SEM_UNDO;
if (semop(semid, &v_buf, 1) == -1)
{
printf("v(semid) failed");
exit(1);
}
return (0);
}

```

// Shared variable by file

void reset(char \*fileVar,int i) { // fileVar라는이름의텍스트화일을새로  
만들고0값을기록한다.

```

if (access(fileVar, F_OK) == -1) {
FILE *fp = fopen(fileVar, "w");
time_t timer;

```

```

time(&timer);
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),i);
fclose(fp);
}
}

```

```

void Store(char *fileVar, int i) { // fileVar 파일끝에i 값을append한
다
FILE *fp = fopen(fileVar, "a");
fprintf(fp, "PID: %d %d\n",getpid(), i);
fclose(fp);
}

```

```

int Load(char *fileVar) { // fileVar 파일의마지막값을읽어온다.
int num;
FILE *fp = fopen(fileVar, "r");
fseek(fp, -4, SEEK_END);
while (1) {
fscanf(fp, "%d", &num);
if (feof(fp)) {
break
}
}
}

```

```

fclose(fp);
return num;
}

```

```

void add(char *fileVar, int i) { // fileVar 파일의마지막값을읽어서i를
더한후에이를끝에append 한다.
FILE *fp = fopen(fileVar, "a");

```

```

time_t timer;
time(&timer);
int add_num = Load(fileVar);
add_num += i
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),
add_num);
fclose(fp);
}

```

void sub(char \*fileVar, int i) { // fileVar 파일의 마지막값을읽어서i를 뺀후에이를끝에append 한다.

```

FILE *fp = fopen(fileVar, "a");
int sub_num = Load(fileVar);
time_t timer;
time(&timer);
sub_num -= i
fprintf(fp, "PID: %d time: %s %d\n", getpid(), ctime(&timer),
sub_num);
fclose(fp);

}

```

// Class Lock

typedef struct \_lock { //lock의구조체semid로이루어졌다.

int semid;

} Lock

void initLock(Lock \*l, key\_t semkey) { //lock 초기화함수로세마포를연결하며만약없다면초기값을1로주면서새로만들어서연결한다.

```

if ((l->semid = initsem(semkey, 1)) < 0)

```

```

exit(1);

```

```

}

```

```
void Acquire(Lock *l) { //p연산을이용해acquire를구현한다.  
p(l->semid);  
}
```

```
void Release(Lock *l) { //v연산을이용해release를구현한다.
```

```
v(l->semid);  
}
```

```
// Class CondVar  
typedef struct _cond { //condvar의구조체대기열과semid를포함한다.  
int semid;  
char* queueLength;  
} CondVar
```

```
void initCondVar(CondVar *c, key_t semkey, char* queueLength)  
{// queueLength를받아condvar queueLength로지정한다. 또한세마포  
를연결하는데없으면초기값을0로주면서새로만들어서연결한다.  
c->queueLength = queueLength  
reset(c->queueLength,0); // queueLength=0  
if ((c->semid = initsem(semkey, 0)) < 0)  
exit(1);  
}
```

```
void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를  
구현한다. add를통해queueLength를증가시켜준다.  
add(c->queueLength, 1);  
Release(lock);  
p(c->semid);  
Acquire(lock);
```

```
}
```

```
void Signal(CondVar *c) { //condvar를받아대기중인queueLength가  
있다면활성화시켜주고sub를통해queueLength를감소시켜준다.
```

```
if (Load(c->queueLength) > 0) {
```

```
v(c->semid);
```

```
sub(c->queueLength, 1);
```

```
}
```

```
}
```

```
void Broadcast(CondVar *c) { //condvar를받아대기중인  
queueLength가있다면"모두" 활성화시켜주고sub를통해queueLength를  
감소시켜준다.
```

```
while (Load(c->queueLength) > 0) {
```

```
v(c->semid);
```

```
sub(c->queueLength, 1);
```

```
}
```

```
}
```

```
void Take_R2(Lock *L1,CondVar *C2) { //R2 젓가락을가져오기위한  
함수
```

```
Acquire(L1); //L2에관하여acquire함으로써mutual exclusive 하게해준  
다.
```

```
while ((Load("R2.txt") == 0) || ((Load("banker_resource.txt")<2)&&  
(Load("state.txt")==0))) {
```

```
//만약자원이없거나, 남은자원이2개보다작고어떤철학자도식사를하지않는  
다면이때wait 상태에들어가야한다.
```

```
printf("message: %d R2을기다림\n",getpid());
```

```
Wait(C2,L1); //wait 상태로들어간다.
```

```
printf("message: %d R2을기다리다가깨어남\n",getpid());
```

```
}
```

```

Store("R2.txt", 0); //젓가락을얻었으므로R2.txt에0을저장시켜준다.
    sub("banker_resource.txt",1); //젓가락을B 철학자가얻음으로써전체
resource는감소한다.
printf("message: %d R2을얻음\n",getpid());
Release(L1); //Take_R2을수행하였으므로L1을Release 시켜준다.
}

```

```

void Take_R3(Lock *L1,CondVar *C3) { //R3 젓가락을가져오기위한
함수
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준
다.
while (Load("R3.txt") == 0 ) {
/*만약자원이없으면대기상태에들어가야한다. 이때는이미젓가락즉자원을
하나얻은상태이므로
이때는자원의개수가2보다작은상태임을고려할필요가없어진다.*
printf("message: %d R3을기다림\n",getpid());
Wait(C3,L1); //wait 상태로들어간다.
printf("message: %d R3을기다리다가깨어남\n",getpid());
}
Store("R3.txt", 0); //젓가락을얻었으므로R3.txt에0을저장시켜준다.
    sub("banker_resource.txt",1); //젓가락을B 철학자가얻음으로써전체
resource는감소한다.
add("state.txt",1); //이때B 철학자가젓가락을두개얻어서밥을먹을수있으
므로state.txt에1을저장하여식사를하였음을나타낸다.
printf("message: %d R3을얻음\n",getpid());
Release(L1); //Take_R3을수행하였으므로L1을Release 시켜준다.
}

```

```

void Put_R2(Lock *L1,CondVar *C2) { //R2 젓가락을놓기위한함수
식사를다하였을때수행하게된다.
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준

```

다.

```
Store("R2.txt", 1); //젓가락을놓았으므로R2.txt 에1을저장시켜준다.
```

```
    add("banker_resource.txt",1); //위에서처럼자원을젓가락을놓았으므로전체자원의개수를증가시켜준다.
```

```
sub("state.txt",1); //젓가락을내려놓는다는것은식사를마쳤다는의미이기때문에state에1 감소시켜준다.
```

```
    Signal(C2); //queueR2에대기중인철학자를깨워준다.
```

```
    printf("message: %d R2을내려놓는다.\n",getpid());
```

```
Release(L1); //Put_R2을수행하였으므로L1을Release 시켜준다.
```

```
}
```

```
void Put_R3(Lock *L1,CondVar *C3) { //R3 젓가락을놓기위한함수식사를다하였을때수행하게된다.
```

```
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준다.
```

```
Store("R3.txt", 1); //젓가락을놓았으므로R3.txt 에1을저장시켜준다.
```

```
    add("banker_resource.txt",1); //위에서처럼자원을젓가락을놓았으므로전체자원의개수를증가시켜준다.
```

```
Signal(C3); //queueR3에대기중인철학자를깨워준다.
```

```
    printf("message: %d R3을내려놓는다.\n",getpid());
```

```
Release(L1); //Put_R3을수행하였으므로L1을Release 시켜준다.
```

```
}
```

```
void Phil_B(Lock *L1,CondVar *C2,CondVar *C3) {//B 철학자가밥을먹기위해수행하는함수
```

```
    Take_R2(L1, C2); //R2를가져오기위한함수
```

```
    printf("message: %d 생각을시작함\n",getpid());
```

```
sleep(1);
```

```
printf("message: %d 생각을멈춤\n",getpid());
```

```
Take_R3(L1, C3); //R3를가져오기위한함수
```

```
printf("message: %d 가먹기시작함\n", getpid());
```

```

sleep(1);
printf("message: %d 다먹었음\n",getpid());
Put_R2(L1,C2); //R2를내려놓기위한함수
Put_R3(L1,C3); //R3를내려놓기위한함수
}

```

```

int main(int argc, char * argv[]) {
// 서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지않
게해야한다.

```

```

// 실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을
사용해야한다.

```

```

// $ ipcs // 남아있는세마포확T인

```

```

// $ ipcrm -s <semid> // <semid>라는세마포제거

```

```

key_t semkey = 0x100; //semkey값을지정한다.
key_t semkey_chopsticks1 = 0x200; //semkey_chopsticks2 값을지
정한다.
key_t semkey_chopsticks2 = 0x300; //semkey_chopsticks2 값을지
정한다.
key_t semkey_chopsticks3 = 0x400; //semkey_chopsticks3 값을지
정한다.

```

```

reset("banker_resource.txt",3); //전체resource 관리를위해
banker_resource.txt 생성
reset("state.txt",0); //철학자가밥을먹는지보기위한state.txt 생성
reset("R1.txt",1); //R1.txt파일생성만약있다면생성하지않는다. 이때1이

```



란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

reset("R2.txt",1); //R2.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

reset("R3.txt",1); //R3.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.

Lock lock1; //wait, acquire, release에사용된다

CondVar C1; //R1에사용되는condvar

CondVar C2; //R2에사용되는condvar

CondVar C3; //R3에사용되는condvar

initLock(&lock1, semkey); //lock 초기화함수

initCondVar(&C1, semkey\_chopsticks1, "queueR1.txt"); //R1에사용  
하는condvar 함수초기화이때queueR1의파일명을지정한다.

initCondVar(&C2, semkey\_chopsticks2, "queueR2.txt"); //R2에사용  
하는condvar 함수초기화이때queueR2의파일명을지정한다.

initCondVar(&C3, semkey\_chopsticks3, "queueR3.txt"); //R3에사용  
하는condvar 함수초기화이때queueR3의파일명을지정한다.

for(int i=0;i<100;i++)

Phil\_B(&lock1,&C2, &C3);

return 0;

}

# Phil\_C 코드

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define SEMPERM 0600
#define TRUE 1
#define FALSE 0

typedef union _semun { //semaphore의구조체
int val;
struct semid_ds *buf;
ushort *array;
} semun

int initsem(key_t semkey, int n) { //semaphore를초기화시킬때사용
int status = 0, semid;
// 세마포어값을설정
if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT |
IPC_EXCL)) == -1)
{
if (errno == EEXIST)
semid = semget(semkey, 1, 0);
```

```

}
else
{
semun arg;
arg.val = n
// 세마포어제어( setval: 값설정)
status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
{
perror("initsem failed");
return (-1);
}
return (semid);
}

```

```

int p(int semid) { //p연산을하며acquire 할때사용된다.
struct sembuf p_buf;
p_buf.sem_num = 0;
p_buf.sem_op = -1;
p_buf.sem_flg = SEM_UNDO;
if (semop(semid, &p_buf, 1) == -1)
{
printf("p(semid) failed");
exit(1);
}
return (0);
}

```

```

int v(int semid) { //v연산을하며release 할때사용된다.
struct sembuf v_buf;

```

```

v_buf.sem_num = 0;
v_buf.sem_op = 1;
v_buf.sem_flg = SEM_UNDO;
if (semop(semid, &v_buf, 1) == -1)
{
printf("v(semid) failed");
exit(1);
}
return (0);
}

```

```

// Shared variable by file
void reset(char *fileVar,int i) {// fileVar라는이름의텍스트화일을새로
만들고0값을기록한다.
if (access(fileVar, F_OK) == -1) {
FILE *fp = fopen(fileVar, "w");
time_t timer;
time(&timer);
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),i);
fclose(fp);
}
}

```

```

void Store(char *fileVar, int i) {// fileVar 화일끝에i 값을append한
다
FILE *fp = fopen(fileVar, "a");
fprintf(fp, "PID: %d %d\n",getpid(), i);
fclose(fp);
}

```

```

int Load(char *fileVar) {// fileVar 화일의마지막값을읽어온다.

```

```

int num;
FILE *fp = fopen(fileVar, "r");
fseek(fp, -4, SEEK_END);
while (1) {
fscanf(fp, "%d", &num);
if (feof(fp)) {
break
}
}

```

```

fclose(fp);
return num;
}

```

void add(char \*fileVar, int i) { // fileVar 화일의마지막값을읽어서i를 더한후에이를끝에append 한다.

```

FILE *fp = fopen(fileVar, "a");
time_t timer;
time(&timer);
int add_num = Load(fileVar);
add_num += i
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer),
add_num);
fclose(fp);
}

```

void sub(char \*fileVar, int i) { // fileVar 화일의마지막값을읽어서i를 뺀후에이를끝에append 한다.

```

FILE *fp = fopen(fileVar, "a");
int sub_num = Load(fileVar);
time_t timer;

```

```

time(&timer);
sub_num -= i
fprintf(fp, "PID: %d time: %s %d\n", getpid(), ctime(&timer),
sub_num);
fclose(fp);

}

// Class Lock
typedef struct _lock { //lock의구조체semid로이루어졌다.
int semid;
} Lock

void initLock(Lock *l, key_t semkey) { //lock 초기화함수로세마포
를연결하며만약없다면초기값을1로주면서새로만들어서연결한다.
if ((l->semid = initsem(semkey, 1)) < 0)
exit(1);
}

void Acquire(Lock *l) { //p연산을이용해acquire를구현한다.
p(l->semid);
}

void Release(Lock *l) { //v연산을이용해release를구현한다.

v(l->semid);
}

// Class CondVar
typedef struct _cond { //condvar의구조체대기열과semid를포함한다.
int semid;
char* queueLength;

```

```
} CondVar
```

```
void initCondVar(CondVar *c, key_t semkey, char* queueLength)
{ // queueLength를받아condvar queueLength로지정한다. 또한세마포
  를연결하는데없으면초기값을0로주면서새로만들어서연결한다.
```

```
  c->queueLength = queueLength
  reset(c->queueLength,0); // queueLength=0
  if ((c->semid = initsem(semkey, 0)) < 0)
    exit(1);
}
```

```
void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를
  구현한다. add를통해queueLength를증가시켜준다.
```

```
  add(c->queueLength, 1);
  Release(lock);
  p(c->semid);
  Acquire(lock);
}
```

```
void Signal(CondVar *c) { //condvar를받아대기중인queueLength가
  있다면활성화시켜주고sub를통해queueLength를감소시켜준다.
```

```
  if (Load(c->queueLength) > 0) {
    v(c->semid);
    sub(c->queueLength, 1);
  }
}
```

```
void Broadcast(CondVar *c) { //condvar를받아대기중인
  queueLength가있다면"모두" 활성화시켜주고sub를통해queueLength를
  감소시켜준다.
```

```
  while (Load(c->queueLength) > 0) {
```

```

v(c->semid);
sub(c->queueLength, 1);
}
}

```

```

void Take_R1(Lock *L1,CondVar *C1) { //R1 젓가락을가져오기위한
함수
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준
다.
while (Load("R1.txt") == 0) {
/*만약자원이없으면대기상태에들어가야한다.   이때는이미젓가락즉자원을
하나얻은상태이므로
이때는자원의개수가2보다작은상태임을고려할필요가없어진다.*
printf("message: %d R1을기다림\n",getpid());
Wait(C1,L1); //wait 상태로들어간다.
printf("message: %d R1을기다리다가깨어남\n",getpid());
}
Store("R1.txt", 0); //젓가락을얻었으므로R1.txt에0을저장시켜준다.
    sub("banker_resource.txt",1);   //젓가락을C철학자가얻음으로써전체
resource는감소한다.
add("state.txt",1); //이때C철학자가젓가락을두개얻어서밥을먹을수있으므
로state.txt에1을저장하여식사를하였음을나타낸다.
printf("message: %d R1을얻음\n",getpid());
Release(L1); //Take_R1을수행하였으므로L1을Release 시켜준다.
}

```

```

void Take_R3(Lock *L1,CondVar *C3) { //R3 젓가락을가져오기위한
함수
Acquire(L1);//L1에관하여acquire함으로써mutual exclusive 하게해준
다.

```



```

while ((Load("R3.txt") == 0) ||((Load("banker_resource.txt")<2)&&
(Load("state.txt")==0))) {
//만약자원이없거나, 남은자원이2개보다작고어떤철학자도식사를하지않는
다면이때wait 상태에들어가야한다.
printf("message: %d R3을기다림\n",getpid());
Wait(C3,L1); //wait 상태로들어간다.
printf("message: %d R3을기다리다가깨어남\n",getpid());
}
Store("R3.txt", 0); //젓가락을얻었으므로R3.txt에0을저장시켜준다.
sub("banker_resource.txt",1); //젓가락을C 철학자가얻음으로써전체
resource는감소한다.
printf("message: %d R3을얻음\n",getpid());
Release(L1); //Take_R3을수행하였으므로L1을Release 시켜준다.
}

```

```

void Put_R1(Lock *L1,CondVar *C1) {//R1 젓가락을놓기위한함수식
사를다하였을때수행하게된다.
Acquire(L1);//L1에관하여acquire함으로써mutual exclusive 하게해준
다.
Store("R1.txt", 1);//젓가락을놓았으므로R1.txt 에1을저장시켜준다.
add("banker_resource.txt",1);//위에서처럼자원을젓가락을놓았으므로
전체자원의개수를증가시켜준다.
Signal(C1);//queueR1에대기중인철학자를깨워준다.
printf("message: %d R1을내려놓는다.\n",getpid());
Release(L1); //Put_R1을수행하였으므로L1을Release 시켜준다.
}

```

```

void Put_R3(Lock *L1,CondVar *C3) {//R3 젓가락을놓기위한함수식
사를다하였을때수행하게된다.
Acquire(L1); //L1에관하여acquire함으로써mutual exclusive 하게해준
다.

```

```

Store("R3.txt", 1); //젓가락을놓았으므로R3.txt 에1을저장시켜준다.
    add("banker_resource.txt",1); //위에서처럼자원을젓가락을놓았으므로전체자원의개수를증가시켜준다.
sub("state.txt",1); //젓가락을내려놓는다는것은식사를마쳤다는의미이기때문에state에1 감소시켜준다.
Signal(C3); //queueR3에대기중인철학자를깨워준다.
    printf("message: %d R3을내려놓는다.\n",getpid());
Release(L1); //Put_R3을수행하였으므로L1을Release 시켜준다.
}

```

```

void Phil_C(Lock *L1,CondVar *C1, CondVar *C3) { //C 철학자가
밥을먹기위해수행하는함수
    Take_R3(L1, C3); //R3를가져오기위한함수
    printf("message: %d 생각을시작함\n",getpid());
    sleep(1);
    printf("message: %d 생각을멈춤\n",getpid());
    Take_R1(L1, C1); //R1를가져오기위한함수
    printf("message: %d 가먹기시작함\n", getpid());
    sleep(1);
    printf("message: %d 다먹었음\n",getpid());
    Put_R3(L1,C3); //R3를내려놓기위한함수
    Put_R1(L1,C1); //R1를내려놓기위한함수
}

```

```

int main(int argc, char * argv[]) { // 서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지않게해야한다.

```

```

// 실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을사용해야한다.

```

```

// $ ipcs // 남아있는세마포확T인

```

```
// $ ipcrm -s <semid>      // <semid>라는세마포제거
```

```
key_t semkey = 0x100; //semkey값을지정한다.
```

```
key_t semkey_chopsticks1 = 0x200; //semkey_chopsticks1 값을지정한다.
```

```
key_t semkey_chopsticks2 = 0x300; //semkey_chopsticks2 값을지정한다.
```

```
key_t semkey_chopsticks3 = 0x400; //semkey_chopsticks3 값을지정한다.
```

```
reset("banker_resource.txt",3);      //전체resource      관리를위해  
banker_resource.txt 생성
```

```
reset("state.txt",0); //철학자가밥을먹는지보기위한state.txt 생성
```

```
reset("R1.txt",1); //R1.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다..
```

```
reset("R2.txt",1); //R2.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.
```

```
reset("R3.txt",1); //R3.txt파일생성만약있다면생성하지않는다. 이때1이  
란자원이존재하는상태0이란자원을누가쓰고있는상태를의미한다.
```

```
Lock lock1; //wait, acquire, release에사용된다
```

```
CondVar C1; //R1에사용되는condvar
```

```
CondVar C2; //R2에사용되는condvar
```

```
CondVar C3; //R3에사용되는condvar
```

```
initLock(&lock1, semkey); //lock 초기화함수
```

```
initCondVar(&C1, semkey_chopsticks1, "queueR1.txt"); //R1에사용
```

하는condvar 함수초기화이때queueR1의파일명을지정한다.  
initCondVar(&C2, semkey\_chopsticks2, "queueR2.txt"); //R2에사용  
하는condvar 함수초기화이때queueR2의파일명을지정한다.  
initCondVar(&C3, semkey\_chopsticks3, "queueR3.txt"); //R3에사용  
하는condvar 함수초기화이때queueR3의파일명을지정한다.

```
for(int i=0;i<100;i++)  
    Phil_C(&lock1,&C1, &C3);  
  
return 0;  
}
```

철학자 세명이 있다 이들은 세개의 젓가락을 공유하여 밥을 먹는다 이때  
굶어죽지 않고 사이좋게 밥(자원)을 나누어 먹을 수 있을까?

이 문제를 해결 하면서 겪은것은 한 프로세스가 자원을 독점하기도 하고  
두개의 프로세스가 서로 자원을 나눠 갖기도 하며 결국 모든 상황들은  
골고루 자원을 나눠 갖지 못하는 현생을 야기한다. 이런 현상들을 해결  
하는 것이 이번 과제의 목표라고 할 수 있었다.

프로세스는 한번에 하나의 일만 가능하다. 그러므로 젓가락도 한번에 하  
나씩만 들 수 있다. 만약 내가 하나의 젓가락을 집었으나 잡은 젓가락  
이외의 다른 젓가락을 옆에 철학자가 집고 있다면 나는 상대방이 젓가락  
을 줄때까지 대기 할 수 밖에 없다. 물론 철학자라면 젓가락을 달라고  
요구할테지만 프로세스에는 그런 똑똑한 기능이 존재 하지 않는다. 그러  
므로 이것을 해결하기 위해서는 프로세스에 몇가지의 규칙을 적용 하여  
야 한다. 왼쪽의 젓가락, 오른쪽의 젓가락 중 지금 사용하지 않는 젓가

락(자원)이어야 하며, 누군가 젓가락(자원)을 사용중 이라면 다음에 내가 쓰고 싶다고 표시를 해두는 것이다. 이를통해 대기 하던 철학자가 먼저 젓가락을 얻게 되는 것 이다.

철학자들이 젓가락 하나씩 확보해놓고 식사를 하기위해 다른 철학자에게 젓가락을 요구하듯이 프로세스도 서로 필요한 자원의 일부분을 확보한 채 상대방이 가진 자원을 요구하는 경우가 발생하게 된다. 이현상을 교착상태(deadlock)라고 한다. 프로세스들이 해결될 수 없는 작업(자원을 확보할 수 있는가 등)들을 반복하므로, 컴퓨터 속도는 매우 느려지게 된다. 심하면 다운된다.

아래와 같은 상태가 dead lock 이라고 할 수 있다 . A philosopher 가 R1을 보유하고 R2를 얻기위해 대기하고 B philosopher가 R2를 보유하고 R3를 얻기 위해 대기하고 C philosopher가 R3를 얻고 R1을 얻기위해 대기하면서 circle이 형성 되서 dead lock이 걸리게 된다.

```

----- Shared Memory Segments -----
key      shmid  owner  perms  bytes  nattch status
0x00000000 720896  knucs  600    524288  2      dest
0x00000000 884737  knucs  600    524288  2      dest
0x00000000 2981890  knucs  600    2097152 2      dest
0x00000000 1572867  knucs  600    524288  2      dest
0x00000000 1212420  knucs  600    524288  2      dest
0x00000000 1310725  knucs  600    524288  2      dest
0x00000000 1490930  knucs  600    524288  2      dest
0x00000000 1702791  knucs  600    524288  2      dest
0x00000000 1474568  knucs  600    67108864 2      dest
0x00000000 1769481  knucs  600    524288  2      dest
0x00000000 3178511  knucs  600    524288  2      dest

----- Semaphore Arrays -----
key      shmid  owner  perms  nsens
[1] 7617  gcc -o phil_A phil_A.c
[2] 7618  gcc -o phil_B phil_B.c
[3] 7619  gcc -o phil_C phil_C.c
knucs@knucs-virtual-machine:~/바탕화면/이름 없는 폴더$ ipcs

----- Message Queues -----
key      msqid  owner  perms  used-bytes  messages
----- Shared Memory Segments -----
key      shmid  owner  perms  bytes  nattch status
0x00000000 720896  knucs  600    524288  2      dest
0x00000000 884737  knucs  600    524288  2      dest
0x00000000 2981890  knucs  600    2097152 2      dest
0x00000000 1572867  knucs  600    524288  2      dest
0x00000000 1212420  knucs  600    524288  2      dest
0x00000000 1310725  knucs  600    524288  2      dest
0x00000000 1490930  knucs  600    524288  2      dest
0x00000000 1702791  knucs  600    524288  2      dest
0x00000000 1474568  knucs  600    67108864 2      dest
0x00000000 1769481  knucs  600    524288  2      dest
0x00000000 3178511  knucs  600    524288  2      dest

----- Semaphore Arrays -----
key      shmid  owner  perms  nsens
knucs@knucs-virtual-machine:~/바탕화면/이름 없는 폴더$ ./phil_A & ./phil_B & ./phil_C &
[1] 7617
[2] 7618
[3] 7619
knucs@knucs-virtual-machine:~/바탕화면/이름 없는 폴더$ message: 7619가 R3를 가져옴
message: 7619가 생각할 시간함
message: 7618가 R2를 가져옴
message: 7618가 생각할 시간함
message: 7617가 R1을 가져옴
message: 7617가 생각할 시간함
message: 7619가 R3를 얻음
message: 7618가 R2를 얻음
message: 7618가 R3를 기다림
message: 7617가 R2를 기다림

```

그것을 깨는 방법중 첫번째는 2번 소스와 같은 Circular wait를 깨는 방법이다. A pilosopher은 기존에 1번 젓가락을 먼저 가질려고 했던것

과 다르게 2번을 먼저 갖게 할려고 하고 1번 젓가락을 나중에 갖게 한다. 이때 B philosopher 와 C philosopher는 원래의 젓가락을 획득하려는 순서를 유지한다.

이러한 형태를 갖도록 하면 A philosopher와 B philosopher는 2번 젓가락을 우선적으로 얻으려고 하는데 둘중 하나는 2번 젓가락에서 wait를 하기 때문에 C philosopher는 정상적으로 작동하게 되고 나머지도 그에 따라 돌아가면서 자원을 먹게 된다. 이형태를 좀더 간단하게 나타내면

기준: A philosopher R1 R2

B philosopher R2 R3

C philosopher R3 R1

예방: A philosopher R2 R1

B philosopher R2 R3

C philosopher R3 R1



```

kmcnc@kmcnc-virtual-machine:~/개발원본/program$ message: 18270 R2를 얻음
kmcnc@kmcnc-virtual-machine:~/개발원본/program$
message: 18270 생각을 시작함
message: 18269 R1을 얻음
message: 18269 생각을 시작함
message: 18271 R3을 기다림
message: 18270 생각을 얻음
message: 18270 R3을 얻음
message: 18270 가 먹기 시작함
message: 18269 생각할 필요 없다.
message: 18269 R2를 기다림
message: 18270 다 먹었음
message: 18270 R2를 내려놓는다.
message: 18270 R3을 내려놓는다.
message: 18271 R3을 기다리다가 깨어남
message: 18271 R3을 얻음
message: 18271 생각을 시작함
message: 18270 R2를 기다림
message: 18269 R2를 기다리다가 깨어남
message: 18269 R2를 기다림
message: 18269 가 먹기 시작함
message: 18269 다 먹었음
message: 18271 생각을 시작함
message: 18271 R1을 기다림
message: 18269 R1을 내려놓는다.
message: 18269 R2를 내려놓는다.
message: 18270 R2를 기다리다가 깨어남
message: 18270 R2를 얻음
message: 18270 생각을 시작함
message: 18269 R1을 기다림
message: 18271 R1을 기다리다가 깨어남
message: 18271 R1을 얻음
message: 18271 가 먹기 시작함
message: 18270 R3을 기다림
message: 18270 R3을 기다림
message: 18271 다 먹었음
message: 18271 R3을 내려놓는다.
message: 18270 R3을 기다리다가 깨어남
message: 18270 R3을 얻음
message: 18270 가 먹기 시작함
message: 18271 R1을 내려놓는다.
message: 18269 R1을 기다리다가 깨어남
message: 18269 R1을 얻음
message: 18269 생각을 시작함
message: 18271 R3을 기다림
message: 18270 다 먹었음
message: 18270 다 먹었음
message: 18270 R2를 내려놓는다.
message: 18270 R3을 내려놓는다.
message: 18271 R3을 기다리다가 깨어남
message: 18271 R3을 얻음
message: 18271 R3을 시작함
message: 18270 R2를 기다림
message: 18269 생각을 얻음
message: 18269 R2를 얻음
message: 18269 가 먹기 시작함
message: 18269 R2를 기다림
message: 18271 R1을 기다림
message: 18269 다 먹었음
message: 18269 R1을 내려놓는다.
message: 18269 R1을 내려놓는다.
message: 18271 R1을 기다리다가 깨어남
message: 18271 R1을 얻음

```