운영체제 (02)

HW3: Deadlock – Dining Philosopher

소프트웨어학부

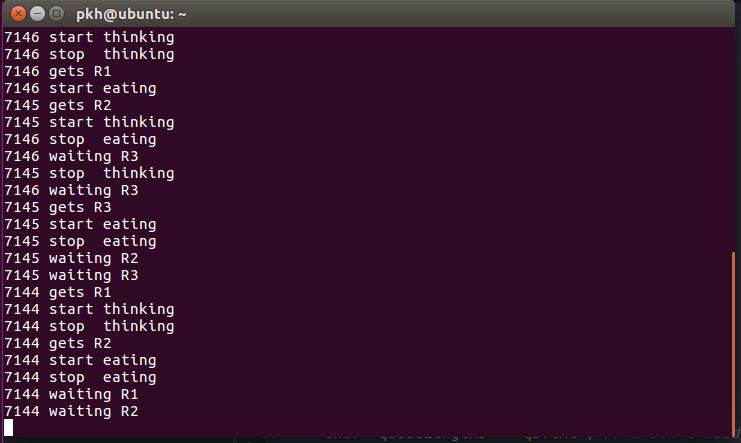
20143060

박기훈

1. Dining Philosopher 문제를 세마포를 이용하여 구현하시오.

- p\_proto.c 코드 첨부

2. Phil\_A, Phil\_B, Phil\_C 를 별도의 프로세스로 실행하여 데드락이 걸리는 상황을 만들고 이 상황의 스냅 샷을 보이고 설명하시오.



  -Deadlock이 걸리는 상황 : phil\_A 는 R1을 집고, phil\_B 는 R2 를 잡고, phil\_C 는 R3 를 잡은 상황.  
   세 철학자 모두 다음 젓가락을 집으려 시도하지만, 모두 사용중이므로 전부 wait 상태를 벗어나지 못하는 DeadLock 상태에 걸리게 된다.  이는 위 코드가 Mutual exclusion , Hold and wait, No preemption, Circular wait - 4가지 Deadlock 의 조건에 부합하기에 일어난다

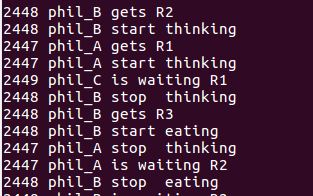
3. Deadlock을 다음 두가지 방법으로 해결하시오.

Prevention(예방) - Circular wait 조건을 깨는 방법으로  
  - 기존 코드에선 Phil A 는 젓가락 R1, R2 순, phil B 는 R2, R3 순, phil C 는 R3,R1 순으로 Take 하여 Circular wait상태가 일어났다.  
     이를 예방하기 위하여 phil C 의 Take 순서만 R1,R3 순으로 바꾸어 Circular wait 상태만 깨 버린다면 Deadlock은 일어나지 않는다.

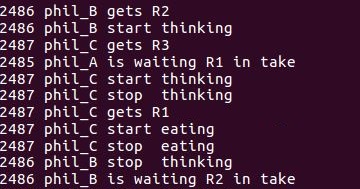
Avoiding(회피) - 강의노트 p21 참조  
  - Take 순서를 바꾸지 않아도, 각 phil 들이 첫번째 Take 하기 이전에 이 젓가락을 집는 것이 안전한지(젓가락을 집음으로서 Deadlock이 일어날 수 있는지) 를 판단한다면 Deadlock 을 회피할 수 있다. 각 phil 은 첫번째 젓가락을 집을 때 이 젓가락이 3개의 젓가락 중 마지막 젓가락인지 판단하여 마지막 젓가락이면 젓가락을 Take 하지 않고 wait한다. 단 누군가 두 개의 젓가락을 사용해 음식을 먹고 있는 상황일 경우, wait 상태에 들어가지 않고 젓가락을 Take 한다(어짜피 먹고있는 phil은 젓가락을 Put할 예정이기 때문에)

4. Deadlock이 걸리는 상태와 이 문제가 해결된 두가지 종류의 상태를 보여주는 실행 결과를 제시하고 설명하시오.

Prevention(예방)

  
   
-phil B는 R2을 집고 phil A 는 R1를 집은 상태 Circular wait 상태였다면 phil C 가 집을 차례에 R3를 잡아 Deadlock 상태에 걸렸겠지만 Circular wait 상태를 깨기 위해 phil C 의 Take 순서를 R1,R3순으로 바꿈. phil C 가 R1을 집으려 시도하지만 phil A 가 R1을 잡고 있으므로 wait상태로 돌입. 이후 phil B 가 R3를 집고 먹기 시작 --반복 -> Deadlock 예방 성공

Avoiding(회피)     
-phil B 는 R2를 집고(조건 만족), phil C 는 R3를 집은 상태(조건 만족). phil A 가 R1을 집으려 시도하지만, 각 phil의 첫번째 Take 조건(아무도 먹고 있지 않지만 쓰고 있는 젓가락은 3개 중 2개이다) 를 만족하지 못하므로 R1을 집지 못하고 wait 상태로 돌입. 이후 phil\_C가 R1을 Take 해 먹음 -반복 -> Banker's Algo 를 이용해 Deadlock Avoiding 성공



**<p\_proto.c>**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <errno.h>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define SEMPERM 0600

#define TRUE 1

#define FALSE 0

typedef union \_semun {

int val;

struct semid\_ds \*buf;

ushort \*array;

} semun;

int initsem (key\_t semkey, int n) {

int status = 0, semid;

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;

status = semctl(semid, 0, SETVAL, arg);

}

if (semid == -1 || status == -1)

{

perror("initsem failed");

return (-1);

}

return (semid);

}

int p (int semid) {

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) {

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 isfile = access(fileVar,0);

if(isfile == -1){

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

fprintf(fp,"0\n");

fclose(fp);

}

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

fclose(fp);

}

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

int n;

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

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",i);

fclose(fp);

}

int Load(char \*fileVar) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp);

return n;

}

void add(char \*fileVar,int i) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp); //store n

n = n + i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",n);

fclose(fp);

}

void sub(char \*fileVar,int i) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp); //store n

n = n - i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",n);

fclose(fp);

}

// Class Lock

typedef struct \_lock {

int semid;

} Lock;

void initLock(Lock \*l, key\_t semkey) {

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

// 세마포를 연결한다.(없으면 초기값을 1로 주면서 새로 만들어서 연결한다.)

exit(1);

}

void Acquire(Lock \*l) {

p(l->semid);

}

void Release(Lock \*l) {

v(l->semid);

}

// Class CondVar

typedef struct \_cond {

int semid;

char \*queueLength;

} CondVar;

void initCondVar(CondVar \*c, key\_t semkey, char \*queueLength) {

c->queueLength = queueLength;

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

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

// 세마포를 연결한다.(없으면 초기값을 0로 주면서 새로 만들어서 연결한다.)

exit(1);

}

void Wait(CondVar \*c, Lock \*lock) {

add(c->queueLength,1);

Release(lock);

p(c->semid);

Acquire(lock);

}

void Signal(CondVar \*c) {

//printf("Signal\n");

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

v(c->semid);

sub(c->queueLength,1);

}

}

void Broadcast(CondVar \*c) {

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

v(c->semid);

sub(c->queueLength,1);

}

}

void Take\_R1(CondVar \*c, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

//if(Load("safe.txt")==1)

while(Load(r) == 0){

printf("%d %s is waiting R1\n",getpid(),name);

Wait(c,lock);

printf("%d %s is wakes up waiting for R1\n",getpid(),name);

}

Store(r,0);

printf("%d %s gets R1\n",getpid(),name);

Release(lock);

}

void Take\_R2(CondVar \*c, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

while(Load(r) == 0){

printf("%d %s is waiting R2\n",getpid(),name);

Wait(c,lock);

printf("%d %s is wakes up waiting for R2\n",getpid(),name);

}

Store(r,0);

printf("%d %s gets R2\n",getpid(),name);

Release(lock);

}

void Take\_R3(CondVar \*c, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

while(Load(r) == 0){

printf("%d %s is waiting R3\n",getpid(),name);

Wait(c,lock);

printf("%d %s is wakes up waiting for R3\n",getpid(),name);

}

Store(r,0);

printf("%d %s gets R3\n",getpid(),name);

Release(lock);

}

void Put\_R1(CondVar \*c, Lock \*lock,char \*r, char\* name){

//printf("%d %d %s\n",c,lock,r);

Acquire(lock);

Store(r,1);

Signal(c);

printf("%d %s is waiting R1\n", getpid(), name);

Release(lock);

}

void Put\_R2(CondVar \*c, Lock \*lock,char \*r, char\* name){

//printf("%d %d %s\n",c,lock,r);

Acquire(lock);

Store(r,1);

Signal(c);

printf("%d %s is waiting R2\n", getpid(), name);

Release(lock);

}

void Put\_R3(CondVar \*c, Lock \*lock,char \*r, char\* name){

Acquire(lock);

Store(r,1);

Signal(c);

printf("%d %s is waiting R3\n", getpid(), name);

Release(lock);

}

void Phil\_A(CondVar\* ca, CondVar\* cb, Lock\* la, Lock\* lb, char\* ra, char\* rb, char\* name) {

Take\_R1(ca, la, ra, name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R2(cb, lb, rb, name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R1(ca, la, ra, name);

Put\_R2(cb, lb, rb, name);

}

void Phil\_B(CondVar\* ca, CondVar\* cb, Lock\* la, Lock\* lb, char\* ra, char\* rb, char\* name) {

Take\_R2(ca, la, ra, name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R3(cb, lb, rb, name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R2(ca, la, ra, name);

Put\_R3(cb, lb, rb, name);

}

void Phil\_C(CondVar\* ca, CondVar\* cb, Lock\* la, Lock\* lb, char\* ra, char\* rb, char\* name) {

Take\_R3(ca, la, ra, name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R1(cb, lb, rb, name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R3(ca, la, ra, name);

Put\_R1(cb, lb, rb, name);

}

int main (int argc, char\* argv[]) {

Lock L1, L2, L3;

CondVar C1, C2, C3;

char\* pA = "phil\_A";

char\* pB = "phil\_B";

char\* pC = "phil\_C";

char\* R1 = "R1file.txt"; //파일 변수 선언

char\* R2 = "R2file.txt";

char\* R3 = "R3file.txt";

char\* queuelength1 = "Q1file.txt"; //waiting queue 선언

char\* queuelength2 = "Q2file.txt";

char\* queuelength3 = "Q3file.txt";

char \*safe = "safe.txt";

reset(R1);

reset(R2); //2개의 파일 변수 초기화

reset(R3);

reset(safe);

Store(R1,1);

Store(R2,1);

Store(R3,1);

Store(safe,3);

key\_t semkey1 = 0x200;

key\_t semkey2 = 0x201;

key\_t semkey3 = 0x202;

key\_t semkey1\_1 = 0x300;

key\_t semkey2\_1 = 0x301;

key\_t semkey3\_1 = 0x302;

initCondVar(&C1, semkey1, queuelength1);

initCondVar(&C2, semkey2, queuelength2);

initCondVar(&C3, semkey3, queuelength3);

initLock(&L1, semkey1\_1);

initLock(&L2, semkey2\_1);

initLock(&L3, semkey3\_1);

if(strcmp(argv[1],"A") == 0) {

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

printf("Count A %d\n",i);

Phil\_A(&C1, &C2, &L1, &L2, R1, R2, pA);

}

printf("Philosopher A done\n");

}

else if(strcmp(argv[1],"B") == 0) {

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

printf("Count B %d\n",i);

Phil\_B(&C2, &C3, &L2, &L3, R2, R3, pB);

}

printf("Philosopher B done\n");

}

else if(strcmp(argv[1],"C") == 0) {

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

printf("Count C %d\n",i);

Phil\_C(&C3, &C1, &L3, &L1, R3, R1, pC);

}

printf("Philosopher C done\n");

}

else

printf("Wrong Parameter");

exit(0);

}

**<p\_cycle.c>**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <errno.h>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define SEMPERM 0600

#define TRUE 1

#define FALSE 0

typedef union \_semun {

int val;

struct semid\_ds \*buf;

ushort \*array;

} semun;

int initsem (key\_t semkey, int n) {

int status = 0, semid;

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;

status = semctl(semid, 0, SETVAL, arg);

}

if (semid == -1 || status == -1)

{

perror("initsem failed");

return (-1);

}

return (semid);

}

int p (int semid) {

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) {

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 isfile = access(fileVar,0);

if(isfile == -1){

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

fprintf(fp,"0\n");

fclose(fp);

}

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

fclose(fp);

}

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

int n;

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

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",i);

fclose(fp);

}

int Load(char \*fileVar) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp);

return n;

}

void add(char \*fileVar,int i) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp); //store n

n = n + i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",n);

fclose(fp);

}

void sub(char \*fileVar,int i) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp); //store n

n = n - i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",n);

fclose(fp);

}

// Class Lock

typedef struct \_lock {

int semid;

} Lock;

void initLock(Lock \*l, key\_t semkey) {

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

// 세마포를 연결한다.(없으면 초기값을 1로 주면서 새로 만들어서 연결한다.)

exit(1);

}

void Acquire(Lock \*l) {

p(l->semid);

}

void Release(Lock \*l) {

v(l->semid);

}

// Class CondVar

typedef struct \_cond {

int semid;

char \*queueLength;

} CondVar;

void initCondVar(CondVar \*c, key\_t semkey, char \*queueLength) {

c->queueLength = queueLength;

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

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

// 세마포를 연결한다.(없으면 초기값을 0로 주면서 새로 만들어서 연결한다.)

exit(1);

}

void Wait(CondVar \*c, Lock \*lock) {

//printf("Wait");

//printf("%d %d \n",c,lock);

add(c->queueLength,1);

Release(lock);

p(c->semid);

Acquire(lock);

}

void Signal(CondVar \*c) {

//printf("Signal\n");

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

v(c->semid);

sub(c->queueLength,1);

}

}

void Broadcast(CondVar \*c) {

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

v(c->semid);

sub(c->queueLength,1);

}

}

void Take\_R1(CondVar \*c, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

//if(Load("safe.txt")==1)

while(Load(r) == 0){

printf("%d %s is waiting R1\n",getpid(),name);

Wait(c,lock);

printf("%d %s is wakes up waiting for R1\n",getpid(),name);

}

Store(r,0);

printf("%d %s gets R1\n",getpid(),name);

Release(lock);

}

void Take\_R2(CondVar \*c, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

while(Load(r) == 0){

printf("%d %s is waiting R2\n",getpid(),name);

Wait(c,lock);

printf("%d %s is wakes up waiting for R2\n",getpid(),name);

}

Store(r,0);

printf("%d %s gets R2\n",getpid(),name);

Release(lock);

}

void Take\_R3(CondVar \*c, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

while(Load(r) == 0){

printf("%d %s is waiting R3\n",getpid(),name);

Wait(c,lock);

printf("%d %s is wakes up waiting for R3\n",getpid(),name);

}

Store(r,0);

printf("%d %s gets R3\n",getpid(),name);

Release(lock);

}

void Put\_R1(CondVar \*c, Lock \*lock,char \*r, char\* name){

Acquire(lock);

Store(r,1);

Signal(c);

printf("%d %s is waiting R1\n", getpid(), name);

Release(lock);

}

void Put\_R2(CondVar \*c, Lock \*lock,char \*r, char\* name){

Acquire(lock);

Store(r,1);

Signal(c);

printf("%d %s is waiting R2\n", getpid(), name);

Release(lock);

}

void Put\_R3(CondVar \*c, Lock \*lock,char \*r, char\* name){

Acquire(lock);

Store(r,1);

Signal(c);

printf("%d %s is waiting R3\n", getpid(), name);

Release(lock);

}

void Phil\_A(CondVar\* ca, CondVar\* cb, Lock\* la, Lock\* lb, char\* ra, char\* rb, char\* name) {

Take\_R1(ca, la, ra, name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R2(cb, lb, rb, name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R1(ca, la, ra, name);

Put\_R2(cb, lb, rb, name);

}

void Phil\_B(CondVar\* ca, CondVar\* cb, Lock\* la, Lock\* lb, char\* ra, char\* rb, char\* name) {

Take\_R2(ca, la, ra, name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R3(cb, lb, rb, name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R2(ca, la, ra, name);

Put\_R3(cb, lb, rb, name);

}

void Phil\_C(CondVar\* ca, CondVar\* cb, Lock\* la, Lock\* lb, char\* ra, char\* rb, char\* name) {

Take\_R1(ca, la, ra, name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R3(cb, lb, rb, name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R3(ca, la, ra, name);

Put\_R1(cb, lb, rb, name);

}

int main (int argc, char\* argv[]) {

Lock L1, L2, L3;

CondVar C1, C2, C3;

char\* pA = "phil\_A";

char\* pB = "phil\_B";

char\* pC = "phil\_C";

char\* R1 = "R1file.txt"; //파일 변수 선언

char\* R2 = "R2file.txt";

char\* R3 = "R3file.txt";

char\* queuelength1 = "Q1file.txt"; //waiting queue 선언

char\* queuelength2 = "Q2file.txt";

char\* queuelength3 = "Q3file.txt";

char \*safe = "safe.txt";

reset(R1);

reset(R2); //2개의 파일 변수 초기화

reset(R3);

reset(safe);

Store(R1,1);

Store(R2,1);

Store(R3,1);

Store(safe,3);

key\_t semkey1 = 0x200;

key\_t semkey2 = 0x201;

key\_t semkey3 = 0x202;

key\_t semkey1\_1 = 0x300;

key\_t semkey2\_1 = 0x301;

key\_t semkey3\_1 = 0x302;

initCondVar(&C1, semkey1, queuelength1);

initCondVar(&C2, semkey2, queuelength2);

initCondVar(&C3, semkey3, queuelength3);

initLock(&L1, semkey1\_1);

initLock(&L2, semkey2\_1);

initLock(&L3, semkey3\_1);

if(strcmp(argv[1],"A") == 0) {

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

printf("Count A %d\n",i);

Phil\_A(&C1, &C2, &L1, &L2, R1, R2, pA);

}

printf("Philosopher A done\n");

}

else if(strcmp(argv[1],"B") == 0) {

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

printf("Count B %d\n",i);

Phil\_B(&C2, &C3, &L2, &L3, R2, R3, pB);

}

printf("Philosopher B done\n");

}

else if(strcmp(argv[1],"C") == 0) {

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

printf("Count C %d\n",i);

Phil\_C(&C1, &C3, &L1, &L3, R1, R3, pC);

}

printf("Philosopher C done\n");

}

else

printf("Wrong Parameter");

exit(0);

}

**<p\_bankers.c>**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <errno.h>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define SEMPERM 0600

#define TRUE 1

#define FALSE 0

typedef union \_semun {

int val;

struct semid\_ds \*buf;

ushort \*array;

} semun;

int initsem (key\_t semkey, int n) {

int status = 0, semid;

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;

status = semctl(semid, 0, SETVAL, arg);

}

if (semid == -1 || status == -1)

{

perror("initsem failed");

return (-1);

}

return (semid);

}

int p (int semid) {

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) {

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 isfile = access(fileVar,0);

if(isfile == -1){

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

fprintf(fp,"0\n");

fclose(fp);

}

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

fclose(fp);

}

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

int n;

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

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",i);

fclose(fp);

}

int Load(char \*fileVar) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp);

return n;

}

void add(char \*fileVar,int i) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp); //store n

n = n + i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",n);

fclose(fp);

}

void sub(char \*fileVar,int i) {

int tmp,id;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %d", &tmp,&id,&n);

}

fclose(fp); //store n

n = n - i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",n);

fclose(fp);

}

// Class Lock

typedef struct \_lock {

int semid;

} Lock;

void initLock(Lock \*l, key\_t semkey) {

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

// 세마포를 연결한다.(없으면 초기값을 1로 주면서 새로 만들어서 연결한다.)

exit(1);

}

void Acquire(Lock \*l) {

p(l->semid);

}

void Release(Lock \*l) {

v(l->semid);

}

// Class CondVar

typedef struct \_cond {

int semid;

char \*queueLength;

} CondVar;

void initCondVar(CondVar \*c, key\_t semkey, char \*queueLength) {

c->queueLength = queueLength;

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

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

// 세마포를 연결한다.(없으면 초기값을 0로 주면서 새로 만들어서 연결한다.)

exit(1);

}

void Wait(CondVar \*c, Lock \*lock) {

//printf("Wait");

//printf("%d %d \n",c,lock);

add(c->queueLength,1);

Release(lock);

p(c->semid);

Acquire(lock);

}

void Signal(CondVar \*c) {

//printf("Signal\n");

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

v(c->semid);

sub(c->queueLength,1);

}

}

void Broadcast(CondVar \*c) {

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

v(c->semid);

sub(c->queueLength,1);

}

}

void Take\_R1(CondVar \*c1,CondVar \*c2,CondVar \*c3, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

//printf("%d\n",Load("safe.txt"));

//while(Load(r) == 0 &&((i==2) || ((i==1)&& (Load("safe.txt")==1)&& (Load("iseat.txt")==1)))) {

if(i==1) {

//printf("%d %d %d\n",Load(r),Load("safe.txt"),Load("iseat.txt"));

while(Load(r)==0 || (Load("safe.txt")==1)) {

if(Load("iseat.txt")==0)

break;

printf("%d %s is waiting R1 in take\n",getpid(),name);

Wait(c1,lock);

printf("%d %s is wakes up waiting for R1\n",getpid(),name);

}

}

else if(i==2){

while(Load(r)==0) {

printf("%d %s is waiting R1 in take %d,\n",getpid(),name,i);

Wait(c1,lock);

printf("%d %s is wakes up waiting for R1\n",getpid(),name);

}

}

Store(r,0);

sub("safe.txt",1);

if(i==2)

sub("iseat.txt",1);

printf("%d %s gets R1\n",getpid(),name);

Release(lock);

}

void Take\_R2(CondVar \*c1,CondVar \*c2,CondVar \*c3 , Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

if(i==1) {

//printf("%d %d %d\n",Load(r),Load("safe.txt"),Load("iseat.txt"));

while(Load(r)==0 || (Load("safe.txt")==1)) {

if(Load("iseat.txt")==0)

break;

printf("%d %s is waiting R2 in take\n",getpid(),name);

Wait(c2,lock);

printf("%d %s is wakes up waiting for R2\n",getpid(),name);

}

}

else if(i==2){

while(Load(r)==0) {

printf("%d %s is waiting R2 in take\n",getpid(),name);

Wait(c2,lock);

printf("%d %s is wakes up waiting for R2\n",getpid(),name);

}

}

Store(r,0);

sub("safe.txt",1);

if(i==2)

sub("iseat.txt",1);

printf("%d %s gets R2\n",getpid(),name);

Release(lock);

}

void Take\_R3(CondVar \*c1,CondVar \*c2,CondVar \*c3, Lock \*lock,char \*r, char\* name,int \*i){

Acquire(lock);

if(i==1){

//printf("%d %d %d\n",Load(r),Load("safe.txt"),Load("iseat.txt"));

while(Load(r)==0 || (Load("safe.txt")==1)) {

if(Load("iseat.txt")==0)

break;

printf("%d %s is waiting R3 in take\n",getpid(),name);

Wait(c3,lock);

printf("%d %s is wakes up waiting for R3\n",getpid(),name);

}

}

else if(i==2){

while(Load(r)==0) {

printf("%d %s is waiting R2 in take\n",getpid(),name);

Wait(c3,lock);

printf("%d %s is wakes up waiting for R3\n",getpid(),name);

}

}

Store(r,0);

sub("safe.txt",1);

if(i==2)

sub("iseat.txt",1);

printf("%d %s gets R3\n",getpid(),name);

Release(lock);

}

void Put\_R1(CondVar \*c, Lock \*lock,char \*r, char\* name,int\* i){

//printf("%d %d %s\n",c,lock,r);

Acquire(lock);

Store(r,1);

Signal(c);

if(i==2)

add("iseat.txt",1);

add("safe.txt",1);

printf("%d %s is put R1\n", getpid(), name);

Release(lock);

}

void Put\_R2(CondVar \*c, Lock \*lock,char \*r, char\* name,int\* i){

//printf("%d %d %s\n",c,lock,r);

Acquire(lock);

Store(r,1);

if(i==2)

add("iseat.txt",1);

add("safe.txt",1);

Signal(c);

printf("%d %s is put R2\n", getpid(), name);

//add("safe.txt",1);

Release(lock);

}

void Put\_R3(CondVar \*c, Lock \*lock,char \*r, char\* name,int\* i){

Acquire(lock);

Store(r,1);

if(i==2)

add("iseat.txt",1);

add("safe.txt",1);

Signal(c);

printf("%d %s is put R3\n", getpid(), name);

Release(lock);

}

void Phil\_A(CondVar \*c1,CondVar \*c2,CondVar \*c3, Lock\* l, char\* ra, char\* rb, char\* name) {

Take\_R1(c1,c2,c3, l, ra, name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R2(c1,c2,c3, l, rb, name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R1(c1, l, ra, name,1);

Put\_R2(c2, l, rb, name,2);

}

void Phil\_B(CondVar \*c1,CondVar \*c2,CondVar \*c3, Lock\* l, char\* ra, char\* rb, char\* name) {

Take\_R2(c1,c2,c3, l, ra,name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R3(c1,c2,c3, l, rb,name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R2(c2, l, ra, name,1);

Put\_R3(c3, l, rb, name,2);

}

void Phil\_C(CondVar \*c1,CondVar \*c2,CondVar \*c3, Lock\* l, char\* ra, char\* rb, char\* name) {

Take\_R3(c1,c2,c3, l, ra,name,1);

printf("%d %s start thinking\n",getpid(),name);

sleep(0.5);

printf("%d %s stop thinking\n",getpid(),name);

Take\_R1(c1,c2,c3, l, rb,name,2);

printf("%d %s start eating\n",getpid(),name);

sleep(0.5);

printf("%d %s stop eating\n",getpid(),name);

Put\_R3(c3, l, ra, name,1);

Put\_R1(c1, l, rb, name,2);

}

int main (int argc, char\* argv[]) {

Lock L1, L2, L3;

CondVar C1, C2, C3;

char\* pA = "phil\_A";

char\* pB = "phil\_B";

char\* pC = "phil\_C";

char\* R1 = "R1file.txt"; //파일 변수 선언

char\* R2 = "R2file.txt";

char\* R3 = "R3file.txt";

char\* queuelength1 = "Q1file.txt"; //waiting queue 선언

char\* queuelength2 = "Q2file.txt";

char\* queuelength3 = "Q3file.txt";

char \*safe = "safe.txt";

char \*iseat = "iseat.txt";

reset(R1);

reset(R2); //2개의 파일 변수 초기화

reset(R3);

reset(safe);

reset(iseat);

Store(R1,1);

Store(R2,1);

Store(R3,1);

Store(safe,3);

Store(iseat,1);

key\_t semkey1 = 0x200;

key\_t semkey2 = 0x201;

key\_t semkey3 = 0x202;

key\_t semkey1\_1 = 0x300;

initCondVar(&C1, semkey1, queuelength1);

initCondVar(&C2, semkey2, queuelength2);

initCondVar(&C3, semkey3, queuelength3);

initLock(&L1, semkey1\_1);

sleep(3);

if(strcmp(argv[1],"A") == 0) {

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

printf("Count A %d\n",i);

Phil\_A(&C1, &C2, &C3, &L1, R1, R2, pA);

}

printf("Philosopher A done\n");

}

else if(strcmp(argv[1],"B") == 0) {

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

printf("Count B %d\n",i);

Phil\_B(&C1, &C2, &C3, &L1, R2, R3, pB);

}

printf("Philosopher B done\n");

}

else if(strcmp(argv[1],"C") == 0) {

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

printf("Count C %d\n",i);

Phil\_C(&C1, &C2, &C3, &L1, R3, R1, pC);

}

printf("Philosopher C done\n");

}

else

printf("Wrong Parameter");

exit(0);

}