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) {

  //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){
    //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){
  //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(&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;
  //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);
  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);
}
```