운영체제

HW2 : Reader & Writer Problem (재제출)

소프트웨어학과
20153180 박정은

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

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

//semid : semget() 함수에 의해 얻어진 세마포

//semnum : 세마포 집합 중 제어하고자하는 세마포의 순번 (0부터 시작)

//semnu == argument

//semctl() : 1 arg = semget호출로 반환된 key값, 2 arg = 동작의 목표가 되는 세마포 번호

```
typedef union   _semun {
        int val; //세마포에 지정될 값
        struct semid_ds *buf; //커널 안에서 사용되는 내부 세마포 자료 구조의 복사본
        ushort *array; //집합 안 모든 세마포 값들을 조회하거나 지정하는데 사용되는
정수값들의 배열을 가리키고 있어야한다.
        } semun;

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

  //binary sempahore
  if ((semid = semget (semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == -1)
  {
     if (errno == EEXIST)
           semid = semget (semkey, 1, 0);
  }

  //scheduling constraints
  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);
}



// Class Lock
typedef struct _lock {
//세마포 1개로 열림,잠김 및 waiting queue의 상태까지 모두 표현된다
  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);
}
```

```
// Shared variable by file
void initVar(char *fileVar) {
// fileVar라는 이름의 텍스트 화일이 없으면 새로 만들고 0값을 기록한다.
  if(fopen(fileVar, "r") == NULL) {
    FILE *fp = fopen(fileVar, "w");
    fprintf(fp, "%d\n", 0);
    fflush(fp);
    fclose(fp);
  }
}

void Store(char *fileVar,int i) {
// fileVar 화일 끝에 i 값을 append한다.
  FILE *fp = fopen(fileVar, "a");
  fprintf(fp, "%d\n", i);
  fflush(fp);
  fclose(fp);
}

int Load(char *fileVar) {
// fileVar 화일의 마지막 값을 읽어 온다.
  int num = 0;
  FILE *fp = fopen(fileVar, "r");
  while(fscanf(fp, "%d", &num) != EOF);
  fflush(fp);
  fclose(fp);

  return num;
}

void add(char *fileVar,int i) {
// fileVar 화일의 마지막 값을 읽어서 i를 더한 후에 이를 끝에 append 한다.
  int num = 0;
  num = Load(fileVar);
  num += i;
  Store(fileVar, num);
}

void sub(char *fileVar,int i) {
// fileVar 화일의 마지막 값을 읽어서 i를 뺀 후에 이를 끝에 append 한다.
  int num = 0;
  num = Load(fileVar);
  num -= i;
  Store(fileVar, num);
}
```

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

void initCondVar(CondVar *c, key_t semkey, char *queueLength) {
   c->queueLength = queueLength;
   initVar(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) {
  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 read_wirte(char *fileVar, int type) {

   //pid = getpid();
   //initLock(&lock,semkey);
   //prinff("\nprocess %d before critical section\n", pid);
   //Acquire(&lock);   // lock.Acquire()
   //printf("process %d in critical section\n",pid);
    /* 화일에서 읽어서 1 더하기 */
   //printf("process %d leaving critical section\n", pid);
   //Release(&lock);   // lock.Release()
   //printf("process %d exiting\n",pid);
   //exit(0);
```

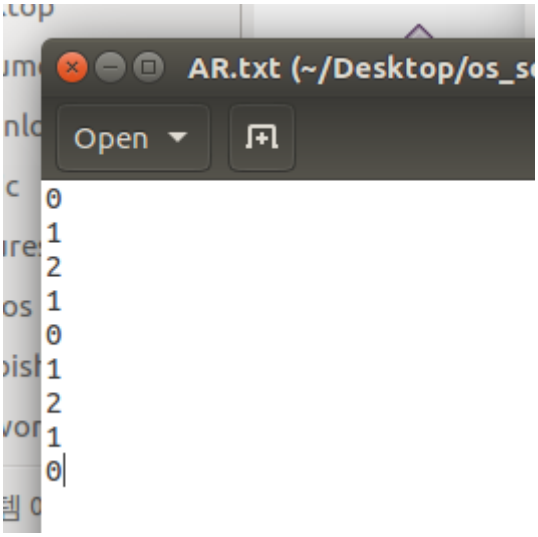
```
  time_t timer;
  struct tm *t;
  timer = time(NULL);
  t = localtime(&timer);

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

  switch(type)
  {
    case 1:
      fprintf(fp, "%d:%d:%d, process %d before critical section\n",
              t->tm_hour, t->tm_min, t->tm_sec, getpid());
      break;
    case 2:
      fprintf(fp, "%d:%d:%d, process %d in critical section\n",
              t->tm_hour, t->tm_min, t->tm_sec, getpid());
      break;
    case 3:
      fprintf(fp, "%d:%d:%d, process %d leaving critical section\n",
              t->tm_hour, t->tm_min, t->tm_sec, getpid());
      break;
    case 4:
      fprintf(fp, "%d:%d:%d, process %d exiting\n",
              t->tm_hour, t->tm_min, t->tm_sec, getpid());
      break;
  }

  fflush(fp);
  fclose(fp);
}


void Reader(
  char *fp_AW, char *fp_WW, char *fp_AR, char *fp_WR,
  CondVar *condVar_reader, CondVar *condVar_writer, Lock *lock,
  char *file_name, int start, int do_sth) {

  sleep(start);

  read_wirte(file_name, 1);
  Acquire(lock);
  read_wirte(file_name, 2);

  int AW = Load(fp_AW);
  int WW = Load(fp_WW);
  while((AW+WW) > 0) {

    add(fp_WR, 1);
    Wait(condVar_reader, lock);
    AW = Load(fp_AW);
    WW = Load(fp_WW);
```

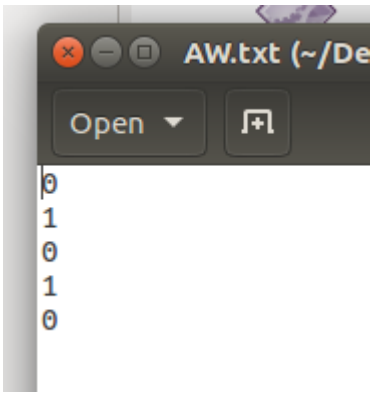
```
    sub(fp_WR, 1);
  }
  add(fp_AR, 1);

  read_wirte(file_name, 3);
  Release(lock);
  read_wirte(file_name, 4);

  sleep(do_sth);

  read_wirte(file_name, 1);
  Acquire(lock);
  read_wirte(file_name, 2);

  sub(fp_AR, 1);
  int AR = Load(fp_AR);
  WW = Load(fp_WW);
  if(AR == 0 && WW > 0) Signal(condVar_writer);

  read_wirte(file_name, 3);
  Release(lock);
  read_wirte(file_name, 4);
}

void Writer(
  char *fp_AW, char *fp_WW, char *fp_AR, char *fp_WR,
  CondVar *condVar_reader, CondVar *condVar_writer, Lock *lock,
  char *file_name, int start, int do_sth) {

  sleep(start);

  read_wirte(file_name, 1);
  Acquire(lock);
  read_wirte(file_name, 2);

  int AW = Load(fp_AW);
  int AR = Load(fp_AR);
  while((AW+AR) > 0) {
    add(fp_WW, 1);
    Wait(condVar_writer, lock);
    AW = Load(fp_AW);
    AR = Load(fp_AR);
    sub(fp_WW, 1);
  }
  add(fp_AW, 1);

  read_wirte(file_name, 3);
  Release(lock);
  read_wirte(file_name, 4);

  sleep(do_sth);
```

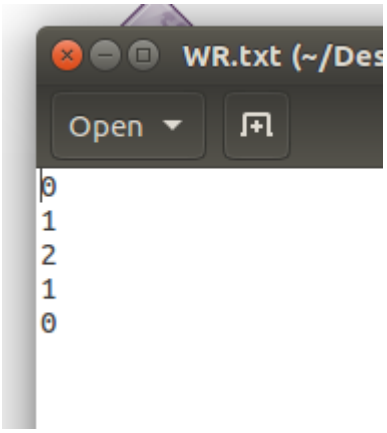
```
  read_wirte(file_name, 1);
  Acquire(lock);
  read_wirte(file_name, 2);

  sub(fp_AW, 1);
  int WW = Load(fp_WW);
  int WR = Load(fp_WR);
  if (WW > 0) Signal(condVar_writer);
  else if (WR > 0) Broadcast(condVar_reader);

  read_wirte(file_name, 3);
  Release(lock);
  read_wirte(file_name, 4);
}

int main(int argc, char* argv[]) {
  key_t semkey_lock = 0x200;
  key_t semkey_reader = 0x201;
  key_t semkey_writer = 0x202;
   // 서버에서 작업할 때는 자기 학번 등을 이용하여 다른 사람의 키와 중복되지 않게 해야
한다.
   // 실행하기 전에 매번 세마포들을 모두 지우거나 아니면 다른 semkey 값을 사용해야 한다.
   // $ ipcs              // 남아 있는 세마포 확인
   // $ ipcrm -s <semid>     // <semid>라는 세마포 제거

  //pid_t pid;
  Lock lock;
  initLock(&lock, semkey_lock);

  CondVar condVar_reader;
  initCondVar(&condVar_reader, semkey_reader, "qlength_r.txt");

  CondVar condVar_writer;
  initCondVar(&condVar_writer, semkey_writer, "qlength_w.txt");

  char *file_name = "fileVar.txt";
  initVar(file_name);

  char *fp_AW = "AW.txt";
  char *fp_WW = "WW.txt";
  char *fp_AR = "AR.txt";
  char *fp_WR = "WR.txt";

  initVar(fp_AW);
  initVar(fp_WW);
  initVar(fp_AR);
  initVar(fp_WR);

//int k = 0;
//for(k = 0; k < argc; k++) {printf("argv[%d] = %s\n", k,argv[k]);}
```

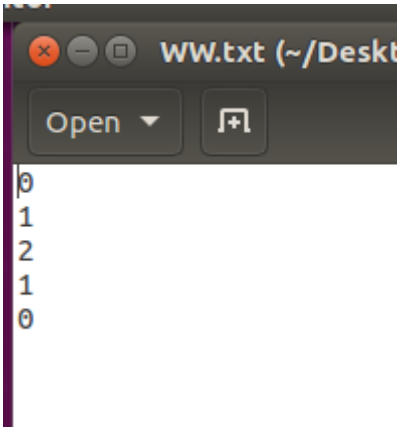
```
 //pid = getpid();

  int start = atoi(argv[2]);
  int do_sth = atoi(argv[3]);

  if(!strcmp(argv[1], "reader")) {
    Reader(fp_AW, fp_WW, fp_AR, fp_WR,
            &condVar_reader, &condVar_writer, &lock, file_name, start, do_sth);
  }
  if(!strcmp(argv[1], "writer")) {
    Writer(fp_AW, fp_WW, fp_AR, fp_WR,
            &condVar_reader, &condVar_writer, &lock, file_name, start, do_sth);
  }


  return 0;
}
```

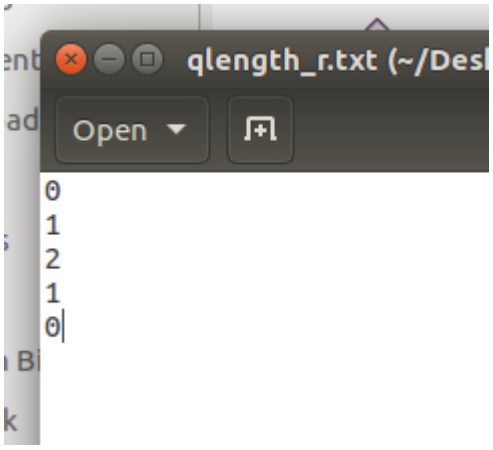
실행 결과 :

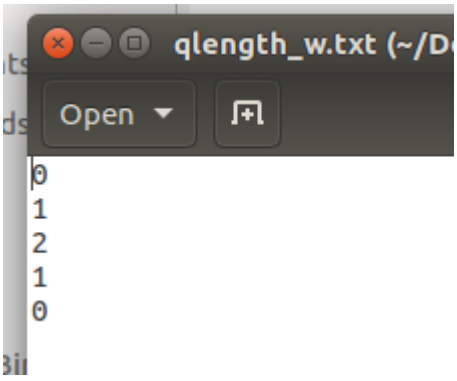

AR.txt (~/Desktop/os_s
Open
0
1
2
1
0
1
2
1
0

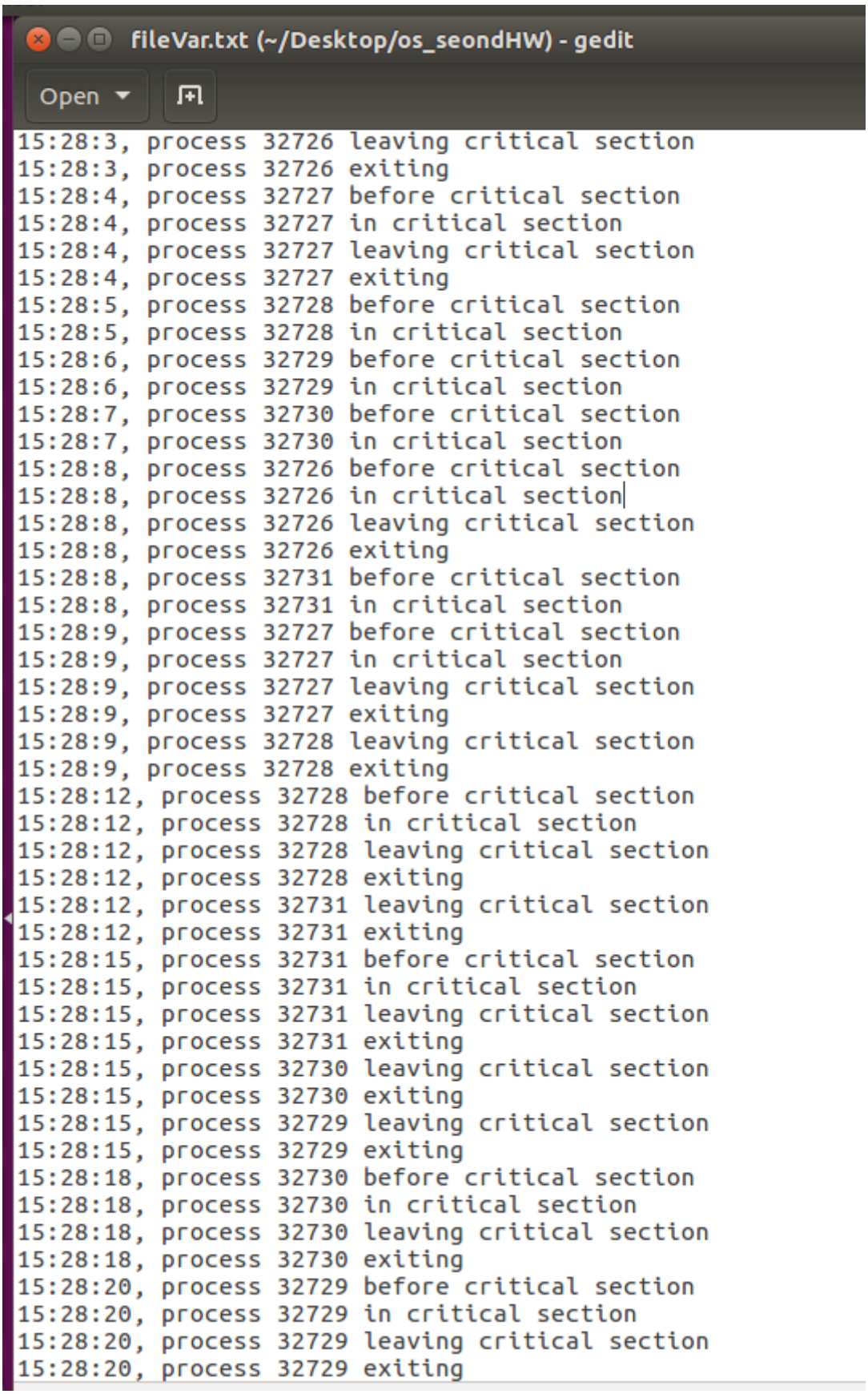

AW.txt (~/De
Open
0
1
0
1
0


WR.txt (~/Des
Open
0
1
2
1
0

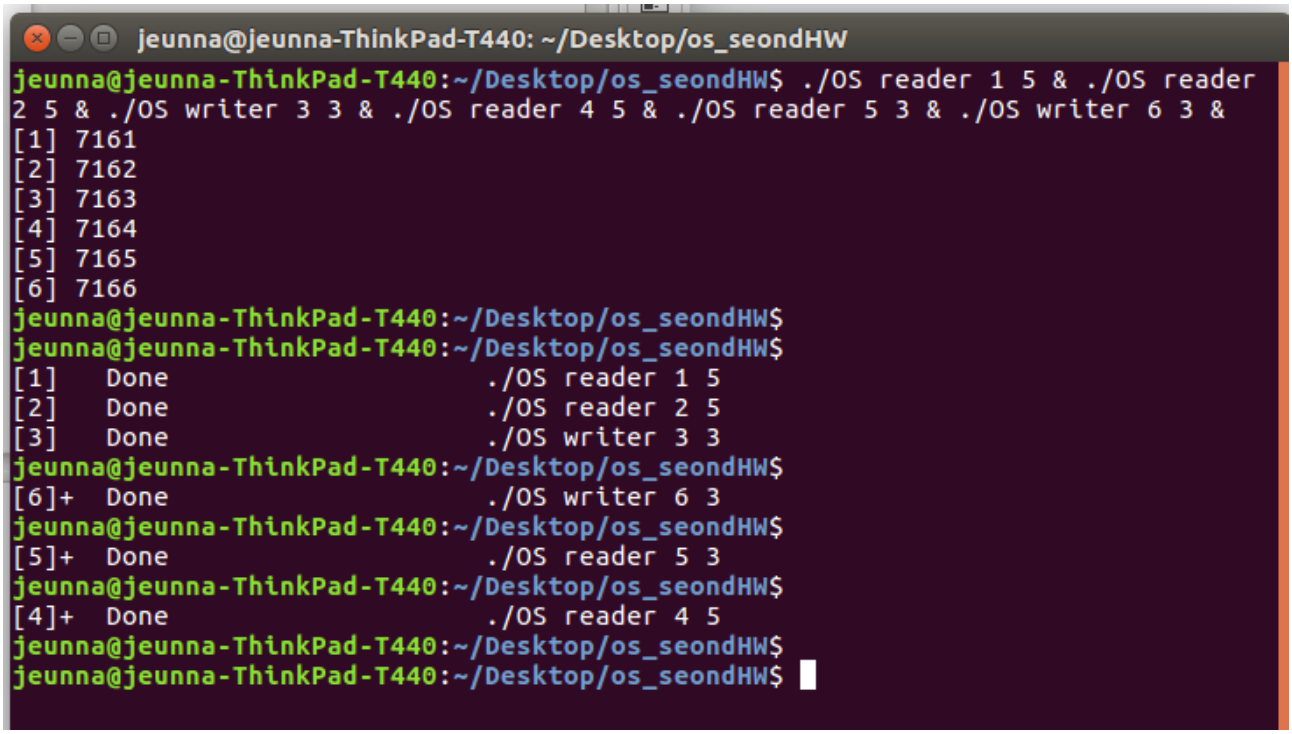

WW.txt (~/Deskt
Open
0
1
2
1
0


qlength_r.txt (~/Des
Open
0
1
2
1
0


qlength_w.txt (~/D
Open
0
1
2
1
0


fileVar.txt (~/Desktop/os_seondHW) - gedit
Open
15:28:3, process 32726 leaving critical section
15:28:3, process 32726 exiting
15:28:4, process 32727 before critical section
15:28:4, process 32727 in critical section
15:28:4, process 32727 leaving critical section
15:28:4, process 32727 exiting
15:28:5, process 32728 before critical section
15:28:5, process 32728 in critical section
15:28:6, process 32729 before critical section
15:28:6, process 32729 in critical section
15:28:7, process 32730 before critical section
15:28:7, process 32730 in critical section
15:28:8, process 32726 before critical section
15:28:8, process 32726 in critical section
15:28:8, process 32726 leaving critical section
15:28:8, process 32726 exiting
15:28:8, process 32731 before critical section
15:28:8, process 32731 in critical section
15:28:9, process 32727 before critical section
15:28:9, process 32727 in critical section
15:28:9, process 32727 leaving critical section
15:28:9, process 32727 exiting
15:28:9, process 32728 leaving critical section
15:28:9, process 32728 exiting
15:28:12, process 32728 before critical section
15:28:12, process 32728 in critical section
15:28:12, process 32728 leaving critical section
15:28:12, process 32728 exiting
15:28:12, process 32731 leaving critical section
15:28:12, process 32731 exiting
15:28:15, process 32731 before critical section
15:28:15, process 32731 in critical section
15:28:15, process 32731 leaving critical section
15:28:15, process 32731 exiting
15:28:15, process 32730 leaving critical section
15:28:15, process 32730 exiting
15:28:15, process 32729 leaving critical section
15:28:15, process 32729 exiting
15:28:18, process 32730 before critical section
15:28:18, process 32730 in critical section
15:28:18, process 32730 leaving critical section
15:28:18, process 32730 exiting
15:28:20, process 32729 before critical section
15:28:20, process 32729 in critical section
15:28:20, process 32729 leaving critical section
15:28:20, process 32729 exiting

전체 실행 시간 : 17초



R1이 먼저 Reader로 들어오고 다음으로 R2가 Reader로 들어옵니다.
W3는 도착해서 R1, R2가 돌아가고 있으니까 waiting하고 R4, R5, W6도 마찬가지로 waiting합니다.
R1이 먼저 끝나고 다음으로 R2가 끝나서 waiting 하고 있는 W3를 깨웁니다.
W3가 무언가하고 다음으로 기다리고 있는 W6를 깨웁니다.
W6는 기다리고 있는 writer가 없으니까 Broadcasting으로 기다리고 있는 reader들을 전체 다 깨웁니다.
그래서 마지막에는 R4랑 R5가 같이 무언가 하지만 실행시간이 더 긴 R4가 제일 마지막에 끝납니다.