Semaphores

Race condition

남편

100$ 출금

잔고: 105 $

아내

70$ 출금

```
int  total = get_total_from_account();          //계좌의 총잔고 확인
int  withrawal = get_withdrawal_amount();   // 사용자의 인출 요구액

total -= withdrawal;                                  // 잔고에서 인출 금액을 공제
update_total_funds(total);
spit_out_money( withdrawal );                    // 이용자에게 현금 지급
```

Race condition

- Concurrent access to shared data
  - may result in data inconsistency.

- Race condition
  - Situation where several processes access and manipulate shared data concurrently.
  - The final value of the shared data
    - depends upon which process finishes last.

- To prevent race conditions,
  - concurrent processes must be synchronized.

Critical-section problem

- General structure of a typical process Pi

```
do {
        entry section
        critical section
        exit section
        remainder section
} while (TRUE);
```

critical section

a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one processes.

Semaphore

- Semaphore S
  - integer variable
- Two standard operations to modify S.
  - wait() and signal()
    - originally called P() and V().

```
P(S) {
        while (S <= 0)
                ;                    // noop
        S--;
}

V(S) {
        S++;
}
```

If positive, decrement & enter.
Otherwise, wait until it gets positive.

Semaphore

- To obtain a shared resource,
  - 1. Test the semaphore that controls the resource.
  - 2. If the value of the semaphore is positive, the process can use the resource. In this case, the process decrements the semaphore value by 1, indicating that it has used one unit of the resource.
  - 3. Otherwise, if the value of the semaphore is 0, the process goes to sleep until the semaphore value is greater than 0. When the process wakes up, it returns to step 1.

Semaphore

- To release the shared resource,
  - 1. When a process is done with a shared resource that is controlled by a semaphore, the semaphore value is incremented by 1.
  - 2. If any other processes are asleep, waiting for the semaphore, they are awakened.

POSIX semaphore

- POSIX semaphore APIs
  - sem_open()
  - sem_wait()
  - sem_post()
  - sem_unlink()

sem_open()

```
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
        Returns: the address of the new semaphore if OK, SEM_FAILED on error
```

- initialize and open a named semaphore.
  - name: identifier of semaphore
  - oflag
    - O_CREAT: the semaphore is created if it does not exist.
  - mode: permissions to be placed on the new semaphore
  - value: the initial value for the new semaphore
- Link with -pthread.

sem_wait()

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);
                                                    Returns:  0 on success; -1 on error
```

- lock a semaphore.
- sem_wait()
  - decrements (locks) the semaphore pointed to by sem.
  - If the semaphore's value is greater than zero, then it decrements, and the function returns.
  - If the value is zero, then the call blocks until it becomes positive.

sem_wait()

- sem_trywait()
  - same as sem_wait(), except that if the decrement cannot be immediately performed, then call returns an error instead of blocking.
- sem_timedwait()
  - same as sem_wait(), except that abs_timeout specifies a limit on the amount of time that the call should block if the decrement cannot be immediately performed.
    - struct timespec {
    - time_t tv_sec; /* seconds */
    - long tv_nsec; /* nanoseconds */
    - };

sem_post()

```
#include <semaphore.h>

int sem_post(sem_t *sem);
                                        Returns:  0 on success; -1 on error
```

- unlock a semaphore.
  - increments (unlocks) the semaphore pointed to by sem.
  - If the semaphore's value consequently becomes greater than zero, then another process blocked in a sem_wait call will be woken up and proceed to lock the semaphore.

sem_unlink()

```
#include <semaphore.h>

int sem_unlink(const char *name);
                                        Returns:  0 on success; -1 on error
```

- remove a named semaphore.
  - removes the named semaphore referred to by name.

POSIX semaphore

Example

```
/* 생략 */
sem_t *mysem;
if((mysem = sem_open("mysem", O_CREAT, 0777, 1)) == NULL) {
        perror("Sem Open Error");
        return 1;
}
while(1) {
        sem_wait(mysem);
        fd = open(countFile,O_RDWR);
        lseek(fd, 0, SEEK_SET);
        read(fd, (void *)&count, sizeof(count));
        printf("Read Data %d\n",count);
        count++;
        lseek(fd, 0, SEEK_SET);
        write(fd, (void *)&count, sizeof(count));
        sleep(1);
        close(fd);
        sem_post(mysem);
}
```

critical section

System V semaphore

- System V semaphore APIs
  - semget()
  - semctl()
  - semop()

semget()

```
#include <sys/sem.h>

int semget(key_t key , int nsems , int flag );
                                        Returns: semaphore ID if OK, -1 on error
```

- return a semaphore set identifier or create a semaphore set.
- key (semaphore key)
  - acts as an external name for an semaphore.
  - Cf. Identifier is an internal name for an semaphore.
- nsems
  - The number of semaphores in the set.

semget()

- semflg
  - If semflg specifies both IPC_CREAT and IPC_EXCL, and a semaphore set already exists for key,
  - ➔ then semget() fails with errno set to EEXIST.
  - is analogous to open(…, O_CREAT|O_EXCL).

semget()

- Example
  - Create a semaphore set
    - key is 100
    - the number of semaphores is 1

```
int semid;

if ((semid = semget((key_t)100, 1, 0600 | IPC_CREAT | IPC_EXCL)) == -1) {
                if (errno == EEXIST) {
                semid  = semget((key_t)100, 1, 0);
                }
} else {
                /* initialize the semaphore value with semctl( ) */
}
```

semctl()

```
#include <sys/sem.h>

int semctl(int semid , int semnum , int  cmd , ... /* union semun arg  */);
                                                    Returns: non-negative if OK, -1 on error
```

- performs the control operation specified by *cmd*
  - on the semaphore set identified by *semid*,
  - or on the *semnum*-th semaphore of that set.
    - The semaphores in a set are numbered starting at 0.

semctl()

- CMD
  - SETVAL: Set the value of semval to arg.val for the *semnum*-th semaphore of the set
  - IPC_RMID: remove the semaphore set.

- union semun

```
union semun {
   int              val;                /* Value for SETVAL */
   struct semid_ds *buf;                /* Buffer for IPC_STAT, IPC_SET */
   unsigned short  *array;              /* Array for GETALL, SETALL */
   struct seminfo  *__buf;              /* Buffer for IPC_INFO
};
```

semctl()

- Initialize semaphore

```
union semun arg;

arg.val = 1;
semctl(semid, 0, SETVAL, arg);
```

- Remove semaphore

```
union semun arg;

semctl(semid, 0, IPC_RMID, arg);
```

semop()

```
#include <sys/sem.h>

int semop(int semid , struct sembuf semoparray[ ], size_t nops );
                                                    Returns: 0 if OK, -1 on error
```

- Performs the semaphore operations.
- semoparray
  - An array of semaphore operations.

```
struct sembuf {
        unsigned short sem_num;     /* semaphore index in array */
        short          sem_op;      /* semaphore operation */
        short          sem_flg;     /* operation flags */
}
```

- nops
  - The number of operations in the array.

semop()

- P() - wait()

```
int p(int semid)
{
        struct sembuf pbuf;

        pbuf.sem_num = 0;                   // first semaphore
        pbuf.sem_op = -1;                   // want to enter critical section.
        pbuf.sem_flg = SEM_UNDO;            // will be automatically undone
                                            // when the process terminates

        if (semop(semid, &pbuf, 1)==-1) {
                printf("p() operation is failed\n");
                return 0;
        } else {
                return 1;
        }
}
```

semop()

- V() - signal()

```
int v(int semid)
{
        struct sembuf vbuf;

        vbuf.sem_num = 0;
        vbuf.sem_op = 1;                // adds this value to the semaphore value(semval).
        vbuf.sem_flg = SEM_UNDO;

        if (semop(semid, &vbuf, 1)==-1) {
                printf("v() operation is failed\n");
                return 0;
        } else {
                return 1;
        }
}
```

Semaphore example

- Example

```
#include <sys/ipc.h>
#include <sys/sem.h>
…

int p(int semid)
{
        struct sembuf pbuf;

        pbuf.sem_num = 0;
        pbuf.sem_op = -1;
        pbuf.sem_flg = SEM_UNDO;

        if (semop(semid, &pbuf, 1)==-1) {
                printf("p() operation is failed\n");
                return 0;
        } else {
                return 1;
        }
}
```

Semaphore example

- Example (cont.)

```
int v(int semid)
{
        struct sembuf vbuf;

        vbuf.sem_num = 0;
        vbuf.sem_op = 1;
        vbuf.sem_flg = SEM_UNDO;

        if (semop(semid, &vbuf, 1)==-1) {
                printf("v() operation is failed\n");
                return 0;
        } else {
                return 1;
        }
}
```

Semaphore example

- Example (cont.)

```
int initsem(key_t skey)
{
        int status = 0, semid;

        if ((semid = semget(skey, 1, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
                if (errno = EEXIST)
                        semid = semget(skey, 1, 0);
        } else
                status = semctl(semid, 0, SETVAL, 1);

        if (semid == -1 || status == -1) {
                printf("initsem is failed\n");
                exit(1);
        } else
                return semid;
}
```

Semaphore example

- Example (cont.)

```
void handlesem(key_t skey)
{
        int semid, pid = getpid();

        if((semid = initsem(skey)) < 0)
                exit(1);

        printf("\nprocess %d before critical section\n", pid);

        p(semid);                                          critical section
        printf("\nprocess %d is in critical section\n", pid);
        sleep(5);
        printf("\nprocess %d is leaving critical section\n", pid);
        v(semid);

        printf("\nprocess %d is exiting\n", pid);
        exit(0);
}
```

Semaphore example

- Example (cont.)

```
int main(int argc, char **argv)
{
        key_t semkey = 0x200;

        if(fork() == 0)
                handlesem(semkey);

        if(fork() == 0)
                handlesem(semkey);

        if(fork() == 0)
                handlesem(semkey);
}
```

Semaphore example

- Results

```
tskim@sslab-server:~/test/semaphore$ ./a.out
process 19496 before critical section
process 19496 is in critical section
process 19498 before critical section
process 19497 before critical section
tskim@sslab-server:~/test/semaphore$
(after 5 seconds)
process 19496 is leaving critical section
process 19496 is exiting
process 19498 is in critical section
(after 5 seconds)
process 19498 is leaving critical section
process 19498 is exiting
process 19497 is in critical section
(after 5 seconds)
process 19497 is leaving critical section
process 19497 is exiting
```