

CSIS0230A Principle of Operating Systems(Class A)

Tutorial 9

Reader-Writer Problem

In the tutorial, you will have to write a program about “Reader and Writer Problem”. Most parts of the program are implemented in the file `/home/guest/ReWr.c`, leaving only the functions `sem_wait` (P) and `sem_signal` (V) are not implemented. The code is shown below.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define READER_COUNT 4
#define WRITER_COUNT 2
#define SEM_COUNT 2 /* Number of semaphores */
#define PERM 0600
#define MUTEX 0 /* Quick semaphore to make sure sh_var access is exclusive */
#define WRT 1

int sem_id, shm_id;
struct memstruct{
    int rcount; /* Number of readers */
    int quit; /* Prepare for quit! */
};
struct memstruct* sh_var;

void sem_wait(int sem_no) { /* to be implemented */ }
void sem_signal(int sem_no) { /* to be implemented */ }
void randsleep(int max) { sleep(rand() % max); }

void create_reader(int reader_no) {
    int i = 0;
    if (fork()) return;
    printf ("Reader %d started\n", reader_no);
    srand(54321 * reader_no);
    while (!sh_var->quit){
        randsleep(5);
        i++;
        sem_wait(MUTEX);
        sh_var->rcount++;
        if (sh_var->rcount == 1)
            sem_wait(WRT); /* Should never block here */
        sem_signal(MUTEX);
        printf("Reader %d reading\n", reader_no); randsleep(3);
        printf("Reader %d done reading\n", reader_no);
        sem_wait(MUTEX);
        sh_var->rcount--;
        if(sh_var->rcount == 0) /* No more reader, so give up lock */
```

```
        sem_signal(WRT);
        sem_signal(MUTEX);
    }
    printf ("Reader %d done\n", reader_no);
    exit(0);
}

void create_writer(int writer_no) {
    int i;
    if (fork()) return;
    printf("Writer %d started\n", writer_no);
    srand(12345 * writer_no);
    for (i=0; i<5; i++) {
        randsleep(10);
        printf("Writer %d waiting\n", writer_no);
        sem_wait(WRT);
        printf("Writer %d writing\n", writer_no); randsleep(3);
        printf("Writer %d done writing\n", writer_no);
        sem_signal(WRT);
    }
    sh_var->quit = 1;
    exit(0);
}

int main() { /* Code to create semaphore, shared memory and threads */ }
```

The program uses the algorithm described in the tutorial notes. The algorithm favours readers over writers: writers have to wait until all readers are done. On the other hand, even if some writers are waiting, readers can start reading if another reader is already reading.

Your tasks:

1. Implement the remaining part of the program. (**Hint**: use *semop()*).
2. (**Optional**) Modify the program, so that it prefers writers: whenever a writer is waiting, all new reader must wait until the waiting writer completes its write.

Hint: To do this, you need the followings. (1) Add a new semaphore for readers to wait to cope with the new restriction, i.e., when some writers are waiting. (2) The readers should maintain a shared variable *nrcount*, to count the number of readers waiting in this semaphore. When the writer completes and find that no other writers are waiting, it calls *signal* this many times. (3) The writers should maintain a new shared variable, *wcount*, to count the number of waiting writers. With this information, a reader or writer can know whether some writers are waiting, by checking whether this value is zero.

One trick to use is for the last writer not to release the write lock when it finds that there is only a some readers waiting and no writer. This way the waiting readers are guaranteed to get serviced next, and thus do not need to wait for the write lock.