Lab 5 Ex 2 - SDP - Davide Gallitelli S241521

The aim of this exercise is to implement semaphore in xv6, which is a kernel without threads. To realize that some files has been modified because we want to implement:

- sem_alloc;
- sem_init;
- sem_destroy;
- sem_wait;
- sem_post.

The struct semaphore is implemented in file.c which has:

- spinlock: allow to implement the mutual exclusion;
- value: counter of the semaphore (in case it would be negative it means that the absolute value are the number of processes that are locked on that semaphore;
- valuelocked: necessary for waking up a single process and so allow us to know how many processes are waiting for a sem_post.

file.c

In *file.c* there is another struct, *sem_table*, which has a spinlock and a vector of semaphore where the maximum number that can be instantiated is defined in another file, param.h, which is 1024. Moreover in this file there are the definitions of the functions to be implemented:

- *void semaphoreInit (void)*: initialize the lock on the semaphore table.
- *int sem_alloc(void)*: acquires the lock on the semaphore table, looks for the first available semaphore (not yet initialized) and returns an integer identifier *k*, then releases the lock.
- *void sem_init(int sem, int count)*: initializes the semaphore identified by *sem* to the value *count*. The variable *valuelocked* is set to 0.
- *void sem_destroy(int sem)*: resets the *count*, as well as *valuelocked*, of the semaphore *sem* to 0.
- *void sem_wait(int sem)*: it acquires the lock on the *sem* semaphore; it decrements its count, and decrements *valuelocked* only if *value* is less than 0; then it loops infinitely, sleeping until a *wakeup* is performed by some other process. The process which performs the *sleep* releases temporary the lock until a *wakeup* is called. A *wakeup* function awakens all processes which performed a *sleep*. A process woken up while value is equal to valuelocked, which means that it cannot access its critical section, cannot leave the loop and sleeps again. Only then it releases the lock on the semaphore.
- *void sem_post(int sem)*: increments the *value* of the *sem* semaphore, then performs a *wakeup* if there is at least one process sleeping, which means that *valuelocked* is negative.

sysfile.c

The *sysfile.c* file contains the definition of the system calls, which are called when the appropriate function is called by the user. The main concept behind these functions is to check for correctness of the arguments, if any, and call the appropriate function defined in *file.c*.

The following example is for the *sys_sem_post* function:

```
int sys_sem_post(void){
        int sem;
        if (argint(0, &sem) < 0)
                return -1;
        sem_post(sem);
        return 0;
}
```

main.c

In order to initialize at boot the semaphore table, the *semaphoreInit()* function is called in the main procedure of the *main.c* file.

Other files

The other files that have been modified are:

- defs.h
- Makefile
- param.h
- syscall.c
- syscall.h
- user.h
- usys.S