Esercizi su OS161 (tratti da compiti di esame) - SOLUZIONI

1. Si spieghi, in relazione alla funzione `syscall()`, che cosa rappresenta la sua variabile `callno`, a cui si assegna il valore `tf->tf_v0`. Si dica poi che cosa significano le seguenti istruzioni alla fine della funzione `syscall()` (si dica, in particolare, cosa sono i campi `tf_v0` e `tf_a3` del trapframe):

```
if (err) {
     tf->tf_v0= err;
     tf->tf_a3= 1;
}
else {
     tf->tf_v0= retval;
     tf->tf_a3= 0;
}
```

`callno` rappresenta il selettore della system call da effettuare, utilizzato all'interno di `syscall` per selezionare il relativo `case`. Viene assegnato al campo `tf->tf_a0` dalla routine che gestisce la trap e che chiama `mips_trap->syscall`.
Le istruzioni in figura gestiscono lo stato e il valore di ritorno della syscall. In `v0` viene posto il valore di ritorno della system call o il valore di errore, mentre in `a3` viene ritornato lo stato di successo/errore (rispettivamente 0 e 1)

2. Si consideri la realizzazione dei `lock` in un sistema OS161. Quale thread deve essere considerato owner (proprietario) di un lock? Motivare la scelta.

1. Il thread che ha creato il lock.
2. L'ultimo thread chiamante la funzione `lock_acquire`.
3. Altro (completare)

1. No, aver creato un lock non significa essere owner. In altri termini, chiamare `lock_create` NON implica l'ownership.
2. No, bisogna verificare che il thread abbia effetivamente acquisito il lock e non sia ancora in attesa. Inoltre, la chiamata a `lock_acquire` può essere fatta anche su un lock diverso da quello in questione.
3. L'owner di un lock è il thread chiamante `lock_acquire` sul lock in questione e che abbia effettivamente acquisito il lock (il lock era libero fino a quel momento oppure ha superato l'eventuale attesa).

3. Sono date le funzioni `lock_release` e `lock_do_i_hold` proposte in figura. Nelle funzioni potrebbero essere presenti errori: in caso affermativo, li si identifichi e li si corregga, motivando.
N.B.: si considerino solamente errori logici/funzionali, non di mistyping o sintassi.

```
void lock_release(struct lock *lock) {
     KASSERT(lock != NULL);
     spinlock_acquire(&lock->lk_lock);
     KASSERT(lock_do_i_hold(lock));
     lock->lk_owner=NULL;
     wchan_wakeone(lock->lk_wchan, &lock->lk_lock);
     spinlock_release(&lock->lk_lock);
}
```

```
bool lock_do_i_hold(struct lock *lock) {
     spinlock_acquire(&lock->lk_lock);
     if (lock->lk_owner==curthread)
         return true;
     spinlock_release(&lock->lk_lock);
     return false;
}
```

L'errore nella `lock_do_i_hold` è l'istruzione `return` senza rilascio dello spinlock. Una possibile correzione è la seguente:

```
bool lock_do_i_hold(struct lock *lock) {
   bool ret;
   spinlock_acquire(&lock->lk_lock);
   ret = lock->lk_owner==curthread;
   spinlock_release(&lock->lk_lock);
   return ret;
}
```

L'errore nella `lock_release` è dato dall'acquisizione dello spinlock prima della `lock_do_i_hold`, che tenterà di acquisire lo stesso spinlock, causando deadlock. Una possibile correzione consiste nello spostare lo `spinlock_acquire` dopo la chiamata a `lock_do_i_hold`.

4. Dato il codice delle funzioni di semaforo P e V riportate di seguito:

```
void P(struct semaphore *sem) {
  spinlock_acquire(&sem->sem_lock);
  while (sem->sem_count == 0) {
     wchan_sleep(sem->sem_wchan,
                 &sem->sem_lock);
  }
  sem->sem_count--;
  spinlock_release(&sem->sem_lock);
}
```

```
void V(struct semaphore *sem) {
  spinlock_acquire(&sem->sem_lock);
  sem->sem_count++;
  KASSERT(sem->sem_count > 0);
  wchan_wakeone(sem->sem_wchan,
                &sem->sem_lock);
  spinlock_release(&sem->sem_lock);
}
```

Si risponda alle seguenti domande:

1. A cosa serve lo `spinlock` all'interno di P e V?
2. Perchè la P contiene un ciclo `while` invece di un `if(sem->sem_count == 0)`?
3. Perchè il ciclo non è presente della V?
4. Perché la `wchan_sleep` riceve come parametro lo `spinlock`? Vale lo stesso motivo per la `wchan_wakeone`?
5. È possibile che la chiamata alla `wchan_wakeone` svegli più di un thread in attesa su `wchan_sleep`? Se no, perché? Se sì, come si fa in modo di rilasciare un solo thread in attesa su P?

1. Lo spinlock è necessario per poter utilizzare un wait channel, quindi per chiamare correttamente `wchan_sleep` e `wchan_wakeone`. Serve a garantire la gestione in mutua esclusione della condizione (`sem->sem_count`)
2. Perché la sincronizzazione tra `wchan_wakeone` e `wchan_sleep` non garantisce che la condizione sul `sem_count`, vera alla chiamata di `wchan_wakeone`, lo sia ancora al ritorno da `wchan_sleep`. Altri thread potrebbero essere risvegliati nel frattempo e quindi modificarla
3. Il ciclo di attesa non è presente nella V perchè il signal (cioè l'uscita da una sezione critica) può essere fatto in qualsiasi momento senza attendere spinlock.
4. La `wchan_sleep` deve rilasciare lo spinlock, prima di mettere il thread in stato “wait”, per poi riprenderlo al risveglio (prima di ritornare al chiamante). La `wchan_wakeone` non deve fare nulla sullo spinlock.
5. No, c’è la garanzia che la `wchan_wakeone` svegli un solo thread in attesa su `wchan_sleep`. La funzione che sveglia più thread in attesa (tutti) è la `wchan_wakeall`.