

Notes on the Solution to Tutorial 6

1 Sample user.c file (point 9 on tutorial sheet)

```
1  /*
2  ** ICOS (C) Kevin Twidle 1997
3  **
4  ** Operating Systems Concepts (JMC2 / MSc Conv) 2005/'06
5  **
6  ** Tutorial 6
7  **
8  ** user.c: User's process for using semaphore to control access to a
9  **         shared variable.
10 **
11 ** Olav Beckmann 2003-11-24
12 **
13 */
14
15 #include <icos/user.h>
16
17 semaphore s;
18
19 int global_shared = 'x';
20
21 /*
22  * set_to_y
23  * This function sets the shared variable to 'y' and then makes
24  * the semaphore available.
25  */
26 void set_to_y( void ) {
27     for( ;; ) {
28         con_outc( '1' );
29         global_shared = 'y';
30         V( &s );
31         delay( 1 );
32     }
33 }
34
35 /*
36  * print_value
37  * This function prints the value of the shared variable, but only
38  * when the semaphore indicates that it has been correctly set to
39  * y. We release the semaphore afterwards to make sure that it can be
40  * accessed by other processes.
41  */
42 void print_value( void ) {
43     for( ;; ) {
44         con_outc( '2' );
45         P(&s);
46         con_outc(global_shared);
47         V(&s);
48         delay( 1 );
49     }
50 }
51
52 /*
53  * set_to_n
54  *
55  * This is a function that spoils the shared variable by setting it
56  * to 'n'. The function takes but does not release the semaphore, thus
57  * guaranteeing that the value of the shared variable cannot be
58  * printed until it has been reset.
59  */
60 void set_to_n( void ) {
61     for( ;; ) {
62         con_outc( '3' );
63         P(&s);
64         global_shared = 'n';
```

```
65     delay( 1 );
66   }
67 }
68
69 void user_init(void) {
70
71     initSema( &s, 0 );
72     create( set_to_y, 8192, 3 );
73     create( print_value, 8192, 3 );
74     create( set_to_n, 8192, 3 );
75 }
```

2 Notes and Comments

This exercise is meant to give you some simple experience of programming with semaphores in Simple Kernel.

- The basic idea is that the semaphore gets set by the function `set_to_n`, blocking access to the shared variable by `print_value`, and released by `set_to_y`, freeing access.
- Note that there are equal numbers of P and V operations in the program.
- This solution would not ensure that only `y` is printed without the `delay()` calls in each process: with no `delay()`, process `set_to_y` could iterate more than once, raising the value of the semaphore to a value higher than 1, which would mean that multiple P operations could take place before the value of the semaphore becomes zero and blocks a process.
- As an alternative to calling `delay()` in each process, we could have modified the semaphore mechanism in `sem.c` to prevent semaphores from reaching a value higher than 1.
- The scheduling priorities of the processes are equal here, if they were not, we would also have to consider the possibility of processes that free the semaphore being starved of access, thus deadlocking the entire system.