Exercise sheet 7 – Process synchronisation 2

Goals:

- Understand and solve advanced synchronisation issues

Exercise 7.1: Reader-Writer problem 1 (theoretical, pseudo C code)

Given is the following reader-writer pseudo C code. There exists one instance of the `writer` and 3 instances of the `reader`.

```
//initialise semaphores



//writer and readers
```

```
writer() {
  while(1) {

    data = collect_data();





    write_data("data_file", data);









  }
}
```

```
reader() {
  while(1) {








    data = read_data("data_file");








    work_with_data(data);
  }
}
```

(a) What is the problem here if nothing is synchronised?

Proposal for solution: The readers can read inconsistent data.

(b) Solve the race condition with appropriate semaphores and a `num_readers` variable that counts the readers that are currently reading from the file.

Proposal for solution:

```
//initialise semaphores
int num_active_readers = 0;
seminit(s_file, 1);
seminit(s_readers, 1);
//writer and readers
```

```c
writer() {
  while(1) {

    data = collect_data();




    P(s_file);
    write_data("data_file", data);
    V(s_file);








  }
}
```

```c
reader() {
  while(1) {
    P(s_readers);
    ++num_active_readers;
    if(num_active_readers == 1) {
      P(s_file);
    }
    V(s_readers);

    data = read_data("data_file");

    P(s_readers);
    --num_active_readers;
    if(num_active_readers == 0) {
      V(s_file);
    }
    V(s_readers);

    work_with_data(data);
  }
}
```

(c) Who is preferred here, the readers or the writer?

Proposal for solution: The readers are preferred.

Exercise 7.2: Reader-Writer problem 2 (theoretical, pseudo C code)

There are three processes P1, P2, P3 which are periodically accessing the same file. There is an order for accessing the file: At first P1 accesses the file. When it has finished, P2 and P3 should access the file at the same time. When both processes are finished with this, a new cycle starts and again P1 accesses the file, and so on ... . Write pseudo C code which uses semaphores for accessing the file.

Proposal for solution:

```c
//initialise semaphores
seminit(s_p1, 2);
seminit(s_p2, 0);
seminit(s_p3, 0);
//processes
```

```c
P1() {
  while(1) {
    P(s_p1);
    P(s_p1);
    work_with_file();
    V(s_p2);
    V(s_p3);
  }
}
```

```c
P2() {
  while(1) {

    P(s_p2);
    work_with_file();
    V(s_p1);

  }
}
```

```c
P3() {
  while(1) {

    P(s_p3);
    work_with_file();
    V(s_p1);

  }
}
```

Exercise 7.3: Producer-Consumer problem (theoretical, pseudo C code)

Consider a hot day in summer: You have a butler, a mini refrigerator and an endless supply of beer. The problem is that the beer is warm and there is only space for one bottle in the refrigerator. If you take the beer from the refrigerator, the butler puts in a new bottle. The beer is cold enough when it was in the refrigerator for at least one hour. Only one person can access the refrigerator at the same time. Describe this procedure with pseudo C code and solve the synchronisation with semaphores! Tip: You need only 2 semaphores to solve this.

Proposal for solution:

```
//initialise semaphores
seminit(s_refrigerator_empty, 1);
seminit(s_beer_ready, 0);
//butler and drinker

butler() {
  while(1) {
    P(s_refrigerator_empty);
    put_beer_into_refrigerator();
    cool_beer_for_an_hour();
    V(s_beer_ready);

  }
}

drinker() {
  while(1) {
    P(s_beer_ready);
    take_beer_from_refrigerator();

    V(s_refrigerator_empty);
    drink_beer();
  }
}
```

Exercise 7.4: Mutual exclusion/synchronisation problem (theoretical, pseudo C code)

There are three processes `P1`, `P2`, and `P3` which are periodicly accessing the same file. There is an order for accessing the file: `P1`, then `P2`, and at last `P3`. When `P3` has finished, a new cycle starts and again `P1` accesses the file, and so on ... . Write pseudo C code which uses semaphores for accessing the file.

Proposal for solution:

```
//initialise semaphores
seminit(s_p1_ready, 1);
seminit(s_p2_ready, 0);
seminit(s_p3_ready, 0);
//processes

P1() {
  while(1) {
    P(s_p1_ready);
    work_with_file();
    V(s_p2_ready);
  }
}

P2() {
  while(1) {
    P(s_p2_ready);
    work_with_file();
    V(s_p3_ready);
  }
}

P3() {
  while(1) {
    P(s_p3_ready);
    work_with_file();
    V(s_p1_ready);
  }
}
```