Exercise Sheet 10

Aufgabe 1 (Communication of Processes)

1. What must be considered, when inter-process communication via shared memory segments is used?
2. What is the function of the shared memory table in the Linux kernel?
3. What is the impact of a restart (reboot) of the operating system on the existing shared memory segments?
   (Only a single answer is correct!)

   ☐ The shared memory segments are created new during boot and the contents are restored.
   ☐ The shared memory segments are created new during boot, but they remain empty. This means, only the contents are lost.
   ☐ The shared memory segments and their contents are lost.
   ☐ Only the shared memory segments are lost. The operating system stores the contents in temporary files inside the folder `\tmp`.
4. According to which principle operate message queues?
   (Only a single answer is correct!)

   ☐ Round Robin ☐ LIFO ☐ FIFO ☐ SJF ☐ LJF
5. How many processes can communicate with each other via a pipe?
6. What is the effect, when a process tries to write data into a pipe without free capacity?
7. What is the effect, when a process tries to read data from an empty pipe?
8. Which two different types of pipes exist?
9. Which two different types of sockets exist?
10. Communication via pipes works...
    (Only a single answer is correct!)

    ☐ memory-based ☐ stream-based
    ☐ object-based ☐ message-based
11. Communication via message queues works...
    (Only a single answer is correct!)

    ☐ memory-based ☐ stream-based
    ☐ object-based ☐ message-based

12. Communication via shared memory segments works...
    (Only a single answer is correct!)

    ☐ memory-based ☐ stream-based
    ☐ object-based ☐ message-based

13. Communication via sockets works...
    (Only a single answer is correct!)

    ☐ memory-based ☐ stream-based
    ☐ object-based ☐ message-based

14. Which two sorts of inter-process communication operate bidirectional?

    ☐ Shared memory segments ☐ Message queues
    ☐ Anonymous pipes ☐ Named pipes
    ☐ Sockets

15. Name a sort of inter-process communication, which can only be used for processes, which are closely related to each other.

    ☐ Shared memory segments ☐ Message queues
    ☐ Anonymous pipes ☐ Named pipes
    ☐ Sockets

16. Which sort of inter-process communication allows communication over computer boundaries?

    ☐ Shared memory segments ☐ Message queues
    ☐ Anonymous pipes ☐ Named pipes
    ☐ Sockets

17. With which sorts of inter-process communication remains the data intact without a bound process?

    ☐ Shared memory segments ☐ Message queues
    ☐ Anonymous pipes ☐ Named pipes
    ☐ Sockets

18. For which sort of inter-process communication guarantees the operating system not the synchronization?

    ☐ Shared memory segments ☐ Message queues
    ☐ Anonymous pipes ☐ Named pipes
    ☐ Sockets

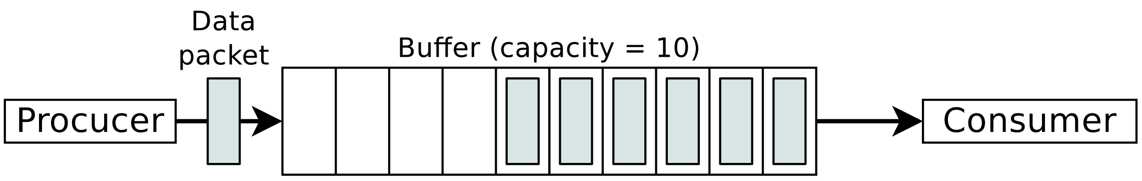
Aufgabe 2 (Cooperation of Processes)

1. What is a semaphore and what is its intended purpose?

2. Which two operations are used with semaphores?
   Provide the names of the operations and for each operation a short description of the functioning.
3. What is the difference between Semaphores versus blocking?
4. What is a binary semaphore?
5. What is a strong semaphore?
6. What is a weak semaphore?
7. What is a mutex and what is its intended purpose?
8. Which type of semaphores has the same functionality as the mutex?
9. Which states can a mutex have?
10. Which Linux/UNIX command returns information about existing shared memory segments, message queues and semaphores?
11. Which Linux/UNIX command allows to erase existing shared memory segments, message queues and semaphores?

Aufgabe 3 (Producer/Consumer Scenario)

A producer should send data to a consumer. A buffer with limited capacity should be used to minimize the waiting times of the consumer. Data is placed into the buffer by the producer and the consumer removes data from the buffer. Mutual exclusion is necessary in order to avoid inconsistencies. If the buffer has no more free capacity, the producer must block itself. If the buffer is empty, the consumer must block itself.



For synchronizing the two processes, create the required semaphores, assign them initial values and insert semaphore operations.

```
typedef int semaphore;



void producer (void) {
  int data;
  while (TRUE) {                // infinite loop
    createDatapacket(data);     // create data packet



    insertDatapacket(data);     // write data packet into buffer



  }
}

void consumer (void) {
  int data;
  while (TRUE) {                // infinite loop



    removeDatapacket(data);     // remove data packet from buffer



    consumeDatapacket(data);    // consume data packet
  }
}
```

Aufgabe 4 (Semaphores)

In a warehouse, packages are delivered constantly by a supplier and picked up by two deliverers. The supplier and the deliverers need to pass through a gate. The gate can always be passed only by a single person. The supplier brings three packages with every shipment to the incoming goods section. One of the deliverers can pick two packages with every pickup from the outgoing goods section. The other deliverer can pick only a single package per pickup from the outgoing goods section.

```
Supplier                  Deliverer_X               Deliverer_Y
{                         {                         {
  while (TRUE)              while (TRUE)              while (TRUE)
  {                         {                         {

    <Pass through gate>;      <Pass through gate>;      <Pass through gate>;



    <Enter incoming           <Enter outgoing           <Enter outgoing
    goods section>;           goods section>;           goods section>;



    <Unload 3 packets>;       <Pick 2 packets>;         <Pick 1 packet>;



    <Leave incoming           <Leave outgoing           <Leave outgoing
    goods section>;           goods section>;           goods section>;



    <Pass through gate>;      <Pass through gate>;      <Pass through gate>;

    }                         }                         }
}                         }                         }
```

Exactly one process `Supplier`, one process `Deliverer_X` and one process `Deliverer_Y` exist.

For synchronizing the three processes, create the required semaphores, assign them values and insert semaphore operations.

These conditions must be met:

- Only a single process can pass through the gate.
  It is impossible that multiple processes pass though the gate simultaneously.
- Only one of both existing deliverers can access the outgoing goods section.
  It is impossible that both deliverers access the outgoing goods section simultaneously.
- It should be possible that the supplier and one of the deliverers can simultaneously unload and pick goods.
- The capacity of the warehouse is 10 packages.
- No deadlocks are allowed.

- At the beginning, the warehouse contains no packets and the gate, as well as the incoming goods section and the outgoing goods section are free.

Source: TU-München, Übungen zur Einführung in die Informatik III, WS01/02

Aufgabe 5 (Inter-Process Communication)

Develop a part of a real-time system, which consists of four processes:

1. **Conv**. This process reads the measured values of A/D converters (analog/digital). It checks the measured values for plausibility and converts them if this is necessary. Because we have no physical A/D converter, the process Conv must generate random numbers. These numbers must be in a certain range of values to simulate an A/D converter.
2. **Log**. This process reads the measured values from the A/D converter (Conv) and writes them into a local file.
3. **Stat**. This process reads the measured values from the A/D converter (Conv) and calculates statistical data, including the average value and the sum.
4. **Report**. This process reads the results of Stat and prints out the statistical data in the shell.

These synchronization conditions must be met:

- **Conv** must first write measured values before **Log** and **Stat** can read the measured values.
- **Stat** must first write statistical data before **Report** can read the statistical data.

Develop and implement the real-time system in C with the appropriate system calls and implement the exchange of data between the four processes once with **pipes**, **message queues** and **shared memory segments with semaphores**. This implies that you program three implementation variants of the real-time system. The source code should be clear to understand because of intensive use of comments.

Approach

The processes Conv, History, Stats and Reports are parallel processes, which are implemented via infinite loops. Implement a framework for the start of the infinite processes with the system call `fork`. Monitor your parallel processes with appropriate commands like `top`, `ps` and `pstree` and determine the parent-child relations.

The program can be terminated with the key combination `Ctrl-C`. To realize this, you need to implement a signal handler for the signal `SIGINT`. Please make sure that when the program is terminated, all occupied resources (message queues, shared memory segments, semaphores) are released.

Develop and implement the following three variants where the exchange of data between the four processes works once with **pipes**, **message queues** and **shared memory segments with semaphores**.

Monitor the message queues, shared memory segments and semaphores with the command `ipcs`. With `ipcrm` it is possible to erase message queues, shared memory segments and semaphores if your program incorrectly missed to free these occupied resources.