Exercise Sheet 9

Exercise 1 (Inter-Process Communication)

1. Describe what a critical section is.
2. Describe what a race condition is.
3. Describe why race conditions are hard to locate and fix.
4. Describe how to avoid race conditions.

Exercise 2 (Synchronization)

1. Explain the advantage of signal and wait compared to busy waiting.
2. Name two problems that can arise from blocking.
3. Explain the difference between signaling and blocking.
4. Mark the four precondition that must be fulfilled at the same time that a deadlock can arise.

   ☐ Recursive function calls
   ☐ Mutual exclusion
   ☐ Frequent function calls
   ☐ Nested `for` loops
   ☐ No preemption
   ☐ Hold and wait
   ☐ > 128 processes in `blocked` state
   ☐ Iterative programming
   ☐ Circular wait
   ☐ Queues

5. Perform a deadlock detection with matrices and check if a deadlock occurs.

$$\text{Existing resource vector} = \begin{pmatrix} 8 & 6 & 7 & 5 \end{pmatrix}$$

$$\text{Current allocation matrix} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 0 & 4 \\ 0 & 2 & 1 & 1 \end{bmatrix} \qquad \text{Request matrix} = \begin{bmatrix} 3 & 2 & 4 & 5 \\ 1 & 1 & 2 & 0 \\ 4 & 3 & 5 & 4 \end{bmatrix}$$

Exercise 3 (Communication of Processes)

1. Explain what must be considered, when using inter-process communication via shared memory segments.

2. Explain the function of the shared memory table in the Linux kernel.

3. Mark 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. Mark the working principle of message queues.
(Only a single answer is correct!)

☐ Round Robin ☐ LIFO ☐ FIFO ☐ SJF ☐ LJF

5. Give the number of processes that can communicate with each other via a pipe.

6. Explain what happens when a process tries to write data into a pipe without free capacity.

7. Explain what happens when a process tries to read data from an empty pipe.

8. Name the two different types of pipes.

9. Name the two different types of sockets.

10. Communication via pipes works...
(Only a single answer is correct!)

☐ memory-based ☐ message-based

11. Communication via message queues works...
(Only a single answer is correct!)

☐ memory-based ☐ message-based

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

☐ memory-based ☐ message-based

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

☐ memory-based ☐ message-based

14. Mark the two sorts of inter-process communication that operate bidirectional.

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

15. Mark the sort of inter-process communication that can only be used for processes that are closely related to each other.

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

16. Mark the sort of inter-process communication that allows communication over computer boundaries.

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

17. Mark the sorts of inter-process communication that remain intact without a bound process.

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

18. Mark the sort of inter-process communication where the operating system does not guarantee the synchronization.

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

Exercise 4 (Cooperation of Processes)

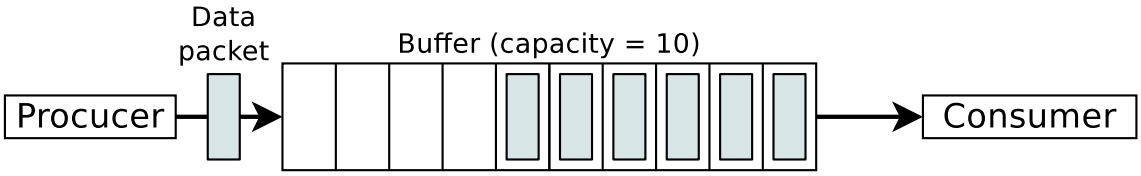
1. Explain what a semaphore is and what its purpose is.
2. Name the two operations are used with semaphores and describe how they work.
3. Explain the difference between Semaphores versus blocking.
4. Explain what a binary semaphore is.
5. Explain what a mutex is and what its purpose is.
6. Name the type of semaphores that has the same functionality as the mutex.
7. Name the states a mutex can have.

8. Name the Linux/UNIX command that returns information about existing shared memory segments, message queues and semaphores.

9. Name the Linux/UNIX command that allows to erase existing shared memory segments, message queues and semaphores.

Exercise 5 (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
  }
}
```

Exercise 6 (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

Exercise 7 (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.

Exercise 8 (Shell Scripts, Data Compression)

1. Program a shell script, which creates a file `testdata.txt`.
   - The file should be filled with zeros.
   - The zeros provides the virtual device file `/dev/zero`.
     (Examples: `dd if=/dev/zero of=/path/to/file bs=512 count=1`
   - The file size should be at least 128 and 512 kB maximum.
   - How large the file becomes, should be specified randomly via `RANDOM`.
2. Program a shell script, which reads a file name as command line argument.
   - The shell script should check the file to find out if it is a file, a link or a directory.
   - If it is a file, the user should have with `select` these options to choose from:

     ```
     1) ZIP
     2) ARJ
     3) RAR
     4) GZ
     5) BZ2
     6) All
     7) Exit
     ```

   - If the user selects a compression algorithm, the file should be compressed with this compression algorithm and the file name should be adjusted accordingly. The file size of the original file and the file size of the compressed file should be printed out both for comparison reasons. e.g.:

```
testdata.txt        <filesize>
testdata.txt.rar    <filesize>
```

- If the user selects the option (`All`), the script should compress the file with all compression algorithms and print out the file size of the original file and the file sizes of the compressed files for comparison reasons.

```
testdata.txt        <filesize>
testdata.txt.zip    <filesize>
testdata.txt.arj    <filesize>
testdata.txt.rar    <filesize>
testdata.txt.gz     <filesize>
testdata.txt.bz2    <filesize>
```

3. Test the shell script with the generated file `testdata.txt`. What is the result?

Exercise 9 (Shell Scripts, File Browser)

Program a shell script, which implements a file browser via `select`.

- The list of files and directories in the current directory should be printed out and the individual entries should be selectable.
- If a file is selected, the file name with the extension, the number of characters, words and lines as well as an information about the file content is printed out. e.g.:

```
<Filename>.<Extension>
Characters: <Number>
Lines:      <Number>
Words:      <Number>
Content:    <Information>
```

Information about the number of characters, words and lines of a file returns the command `wc`. *Information about the contents of a file provides the command* `file`.

- If a directory is selected, the script should navigate into that directory and print out the files and directories in that directory.
- It should also be possible to move up the directory tree into the directory's parent directory (`cd ..`).