Solution of Exercise Sheet 9

Exercise 1 (Inter-Process Communication)

1. Describe what a critical section is.

   Processes carry out read and write operations on common data. Critical sections may not be processed by multiple processes at the same time.

2. Describe what a race condition is.

   It is an unintended race condition of two processes, which want to modify the value of the same record.

3. Describe why race conditions are hard to locate and fix.

   The result of a process depends on the order or timing of other events. The occurrence of the symptoms depends on different events. The symptoms may be different or disappear with each test run.

4. Describe how to avoid race conditions.

   Race conditions can be avoided with the semaphore concept.

Exercise 2 (Synchronization)

1. Explain the advantage of signal and wait compared to busy waiting.

   When using busy waiting, computing time of the CPU is wasted because it is again and again occupied by the waiting process. Using signal and wait causes lesser CPU workload because the waiting process is blocked and later deblocked.

2. Name two problems that can arise from blocking.

   Starvation and deadlock.

3. Explain the difference between signaling and blocking.

   Signaling specifies the execution order of the critical sections of processes.

   Blocking secures critical sections. The execution order of the critical sections of the processes is not specified. It is just ensured that the execution of critical sections does not overlap.

4. Mark the four precondition that must be fulfilled at the same time that a deadlock can arise.

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

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

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

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

The existing resource vector and the current allocation matrix are used to calculate available resource vector.

$$\textit{Available resource vector} = \begin{pmatrix} 3 & 2 & 6 & 0 \end{pmatrix}$$

Only process 2 can run with this available resource vector. The following available resource vector results when process 2 has finished execution and deallocates its resources.

$$\textit{Available resource vektor} = \begin{pmatrix} 6 & 3 & 6 & 4 \end{pmatrix}$$

Only process 3 can run with this available resource vector. The following available resource vector results when process 3 has finished execution and deallocates its resources.

$$\textit{Available resource vector} = \begin{pmatrix} 6 & 5 & 7 & 5 \end{pmatrix}$$

Process 1 is not blocked.

No deadlock occurs.

Exercise 3 (Communication of Processes)

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

The processes need to coordinate the accesses themselves and to ensure that their memory accesses are mutually exclusive. A receiver process, cannot read data from the shared memory, before the sender process has finished its current write operation. If access operations are not coordinated carefully, inconsistencies occur.

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

   Linux/UNIX operating systems contain a shared memory table, which contains information about the existing shared memory segments. This information includes: Start address in memory, size, owner (username and group) and privileges.

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.

   2

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

   The process that tries to write into the pipe is blocked.

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

   The process that tries to read from the pipe is blocked.

8. Name the two different types of pipes.

   Anonymous pipes and named pipes.

9. Name the two different types of sockets.

Connection-less sockets (also called: datagram sockets) and connection-oriented sockets (also called: stream 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. Name the two operations are used with semaphores and describe how they work.

   A semaphore is a counter lock.

2. Name the two operations are used with semaphores and describe how they work.

   The access operation `P(S)` tries to reduce (decrement) the value of the counter variable `S`.

   The access operation `V(S)` increments the value of the counter variable `S`.

3. Explain the difference between Semaphores versus blocking.

   In contrast to semaphores, can locks only be used to allow a single process entering the critical section at the same time.

4. Explain what a binary semaphore is.

   Binary semaphores are initialized with value 1 and ensure that 2 or more processes cannot simultaneously enter their critical sections.

5. Explain what a mutex is and what its purpose is.

   Semaphores offer the feature of counting. However, if this feature is not required, a simplified semaphore version, the mutex can be used instead. Mutexes are used to protect critical sections, which are allowed to be accessed by only a single process at any given moment.

6. Name the type of semaphores that has the same functionality as the mutex.

   Binary semaphore.

7. Name the states a mutex can have.

   Mutexes can only have 2 states: „occupied“ and „not occupied“.

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

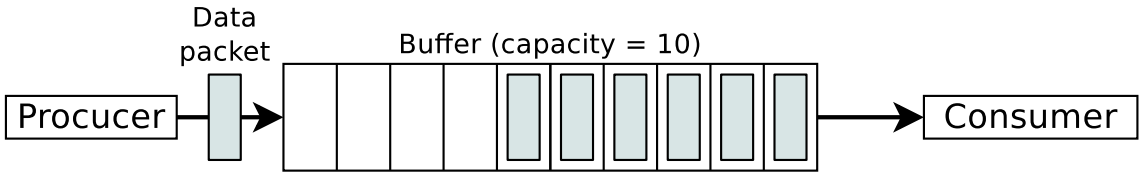
```
ipcs
```

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

   ```
   ipcrm
   ```

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;          // semaphores are of type integer
semaphore filled = 0;           // counts the occupied locations in the buffer
semaphore empty  = 10;          // counts the empty locations in the buffer
semaphore mutex  = 1;           // controls access to the critial sections

void producer (void) {
  int data;

  while (TRUE) {                // infinite loop
    createDatapacket(data);     // create data packet
    P(empty);                   // decrement the empty locations counter
    P(mutex);                   // enter the critical section
    insertDatapacket(data);     // write data packet into the buffer
    V(mutex);                   // leave the critical section
    V(filled);                  // increment the occupied locations counter
  }
}

void consumer (void) {
  int data;

  while (TRUE) {                // infinite loop
    P(filled);                  // decrement the occupied locations counter
    P(mutex);                   // enter the critical section
    removeDatapacket(data);     // pick data packet from the buffer
    V(mutex);                   // leave the critical section
    V(empty);                   // increment the empty locations counter
    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.

```
sema gate     = 1
sema outgoing = 1
sema empty    = 10
sema occupied = 0
```

```
Supplier
{
  while (TRUE)
  {
    P(gate);
    <Pass through gate>;
    V(gate);

    <Enter incoming
    goods section>;

    P(empty);
    P(empty);
    P(empty);
    <Unload 3 packets>;
    V(occupied);
    V(occupied);
    V(occupied);

    <Leave incoming
    goods section>;

    P(gate);
    <Pass through gate>;
    V(gate);
    }
}
```

```
Deliverer_X
{
  while (TRUE)
  {
    P(gate);
    <Pass through gate>;
    V(gate);

    P(outgoing);
    <Enter outgoing
    goods section>;

    P(occupied);
    P(occupied);
    <Pick 2 packets>;
    V(empty);
    V(empty);

    <Leave outgoing
    goods section>;
    V(outgoing);

    P(gate);
    <Pass through gate>;
    V(gate);
    }
}
```

```
Deliverer_Y
{
  while (TRUE)
  {
    P(gate);
    <Pass through gate>;
    V(gate);

    P(outgoing);
    <Enter outgoing
    goods section>;

    P(occupied);
    <Pick 1 packet>;
    V(empty);

    <Leave outgoing
    goods section>;
    V(outgoing);

    P(gate);
    <Pass through gate>;
    V(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`.

```
#!/bin/bash
#
# Skript: testdaten_erzeugen.bat
#
# falls Ordner nicht vorhanden, Ordner erzeugen

VERZEICHNIS=/tmp/testdaten
DATEINAME=testdata.txt

if [ ! -d $VERZEICHNIS ] ; then
  if mkdir $VERZEICHNIS ; then
    echo "Ein Verzeichnis für Testdaten wurde erstellt."
  else
    echo "Es konnte kein Verzeichnis erstellt werden."
  fi
```

```
16 else
17   echo "Ein Verzeichnis für Testdaten existiert schon."
18   exit 1
19 fi
20
21 if touch `echo "$VERZEICHNIS/$DATEINAME"` ; then
22   # Zufallszahl zwischen 128 und 512 erstellen
23   ZUFALLSZAHL=`awk -vmin=128 -vmax=512 'BEGIN{srand(); print
      int(min+rand()*(max-min+1))}'`
24   # Die Datei mit Nullen füllen
25   `dd if=/dev/zero of=$VERZEICHNIS/$DATEINAME bs=$ZUFALLSZAHL
      count=1K`
26   echo "Eine Datei für Testdaten wurde erstellt."
27 else
28   echo "Es konnte keine Datei erstellt werden."
29   exit 1
30 fi
```

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>
```

```
#!/bin/bash
#
# Skript: archivieren.bat
#
# Funktion zum komprimieren einer Datei via ZIP
zip_packen() {
  if zip -r $1.zip $1 ; then
    echo "Die Datei $1 wurde via ZIP komprimiert."
  else
    echo "Die Kompression der Datei $1 via ZIP ist
     fehlgeschlagen."
  fi
}

# Funktion zum komprimieren einer Datei via ARJ
arj_packen() {
  if arj a $1.arj $1 ; then
    echo "Die Datei $1 wurde via ARJ komprimiert."
  else
    echo "Die Kompression der Datei $1 via ARJ ist
     fehlgeschlagen."
  fi
}

# Funktion zum komprimieren einer Datei via RAR
rar_packen() {
  if rar a $1.rar $1 ; then
    echo "Die Datei $1 wurde via RAR komprimiert."
  else
    echo "Die Kompression der Datei $1 via RAR ist
     fehlgeschlagen."
  fi
}

# Funktion zum komprimieren einer Datei via GZ
gz_packen() {
  if gzip -c $1 > $1.gz ; then
    echo "Die Datei $1 wurde via GZ komprimiert."
  else
    echo "Die Kompression der Datei $1 via GZ ist
     fehlgeschlagen."
  fi
}

# Funktion zum komprimieren einer Datei via BZ2
bz2_packen() {
  if bzip2 -zk $1 ; then
    echo "Die Datei $1 wurde via BZ2 komprimiert."
  else
    echo "Die Kompression der Datei $1 via BZ2 ist
     fehlgeschlagen."
  fi
}

```

```
50 # Untersuchen ob die als Kommandozeilenargument übergebene
       Datei existiert
51 if [ ! -e $1 ] ; then
52   # Die Datei existiert nicht.
53   echo "Die Datei $1 existiert nicht."
54   # Das Skript beenden.
55   exit 1
56 fi
57
58 # Untersuchen ob die Datei ein Verzeichnis ist.
59 if [ -d $1 ] ; then
60   echo "Das Kommandozeilenargument ist ein Verzeichnis."
61   exit
62 elif [ -L $1 ] ; then
63   echo "Das Kommandozeilenargument ist ein symbolischer Link."
64   exit
65 elif [ -f $1 ] ; then
66   echo "Das Kommandozeilenargument ist eine reguläre Datei."
67
68   # Auswahlmöglichkeiten ausgeben.
69   select auswahl in ZIP ARJ RAR GZ BZ2 Alle Beenden
70
71   do
72     if [ "$auswahl" = "ZIP" ] ; then
73       zip_packen $1
74       ls -lh $1|awk '{print $9,$5}'
75       ls -lh $1.zip|awk '{print $9,$5}' | column -t
76       exit
77     elif [ "$auswahl" = "ARJ" ] ; then
78       arj_packen $1
79       ls -lh $1|awk '{print $9,$5}'
80       ls -lh $1.arj|awk '{print $9,$5}' | column -t
81       exit
82     elif [ "$auswahl" = "RAR" ] ; then
83       rar_packen $1
84       ls -lh $1|awk '{print $9,$5}'
85       ls -lh $1.rar|awk '{print $9,$5}' | column -t
86       exit
87     elif [ "$auswahl" = "GZ" ] ; then
88       gz_packen $1
89       ls -lh $1|awk '{print $9,$5}'
90       ls -lh $1.gz|awk '{print $9,$5}' | column -t
91       exit
92     elif [ "$auswahl" = "BZ2" ] ; then
93       bz2_packen $1
94       ls -lh $1|awk '{print $9,$5}'
95       ls -lh $1.bz2|awk '{print $9,$5}' | column -t
96       exit
97     elif [ "$auswahl" = "Alle" ] ; then
98       zip_packen $1
99       arj_packen $1
100      rar_packen $1
101      gz_packen $1
102      bz2_packen $1
103      ls -lh $1* | awk '{print $9,$5}' | column -t
104      exit
```

```
105     else [ "$auswahl" = "Beenden" ]
106       echo "Das Skript wird beendet."
107       exit
108     fi
109   done
110 else
111   exit 1
112 fi
```

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 ..`).

```
1  !/bin/bash
2  #
3  # Skript: datei_browser.bat
4  #
5  file=""
6
7  while true
8  do
9    if [ "$file" == ".." ] ; then
10     # In der Verzeichnisstruktur eine Ebene höher gehen
11     cd ..
12   elif [ -d $file ] ; then
```

```
13      cd $file              # In ein Verzeichnis wechseln
14    else
15      break
16    fi
17
18    select file in ".." *  # Dateiauswahlliste ausgeben
19    do
20      break
21    done
22  done
23
24  if [ -f $file ]
25  then
26    echo $file          # Dateinamen mit Endung ausgeben
27    echo "Zeichen: "`wc -m $file | awk '{ print $1 }'`
28    echo "Zeilen:  "`wc -l $file | awk '{ print $1 }'`
29    echo "Wörter:  "`wc -w $file | awk '{ print $1 }'`
30    echo "Inhalt:  "
31    cat $file           # Inhalt der Datei ausgeben
32  fi
```