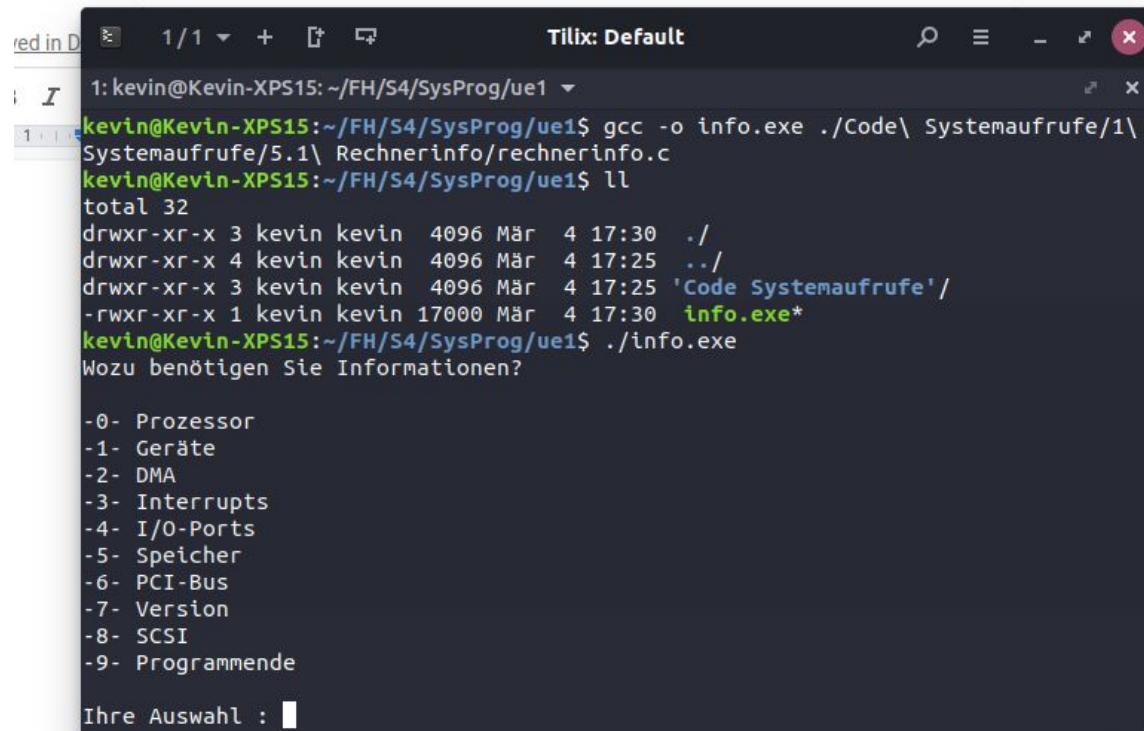


## 5 Aufgabenstellungen Codebeispiele

### 5.1 Hardwareinformationen ermitteln

Zur Ausführung wird zuerst die src Datei kompiliert:



```
1: kevin@Kevin-XPS15: ~/FH/S4/SysProg/ue1
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ gcc -o info.exe ./Code\ Systemaufrufe/1\
Systemaufrufe/5.1\ Rechnerinfo/rechnerinfo.c
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ll
total 32
drwxr-xr-x 3 kevin kevin 4096 Mär  4 17:30 ./
drwxr-xr-x 4 kevin kevin 4096 Mär  4 17:25 ../
drwxr-xr-x 3 kevin kevin 4096 Mär  4 17:25 'Code Systemaufrufe'/
-rwxr-xr-x 1 kevin kevin 17000 Mär  4 17:30 info.exe*
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ./info.exe
Wozu benötigen Sie Informationen?

-0- Prozessor
-1- Geräte
-2- DMA
-3- Interrupts
-4- I/O-Ports
-5- Speicher
-6- PCI-Bus
-7- Version
-8- SCSI
-9- Programmende

Ihre Auswahl : █
```

1. Welche Systeminformationen werden mit dem Programm prinzipiell ausgegeben und in welchem Linux- Verzeichnis sind diese Daten definiert?

Es werden folgende Systeminformationen ausgegeben die unter /proc zu finden sind:

cpuinfo: Informationen über die logischen Prozessoren

devices: Installierte Treiber

dma: Hardware mit direktem Speicherzugriff

interrupts: Liste von I/O Geräte die interrupts erzeugen bzw erzeugt haben

ioports: Liste von registrierten Input/Output Port Regionen

meminfo: Informationen über den aktuellen RAM Status

pci: Liste von allen PCI Geräten

version: Linux Kernel Version und gcc Version welche beim Bau verwendet wurde

scsi/scsi: Liste von SCSI Geräten

## 2. Welche Systeminformationen befinden sich noch in diesem Verzeichnis?

Unter anderem:

apm: Power Management

filesystems: Unterstützte Dateisysteme

kmsg: Kernel Output Messages

locks: Kernel locks

mounts: Mounted Dateisysteme

net: Eine Vielzahl an Netzwerkinformationen

## 3. Mit welchem Linux-Kommandozeilenbefehl kann man alternativ z. B. Informationen zur CPU auslesen?

lscpu, cpuid

## 4. Wie können Informationen anderer devices ausgelesen werden?

/proc/devices oder auch lsblk, lsmem, lsscsi, lsusb, lspci, ...

## 5. Aus dem oben erwähnten Verzeichnis können auch detaillierte Prozessinformationen ausgelesen werden. Geben Sie mit dem Kommando (1) alle Prozesse Ihres Systems aus und ermitteln Sie mit den weiteren Kommandos (2) und (3) anhand der PID weitere Informationen und dokumentieren Sie diese. Welche Prozessinformationen in anderen individuellen PID-Verzeichnissen gibt es noch?

(1) ls /proc

```

kevin@Kevin-XPS15:~$ ls /proc
1      1787    2052    29      422    7254    bus
10     17885   20520   2909    423    728     cgroups
10038  17887   2053    2912    43     73      cmdline
1012   17943   20545   2926    439    730     consoles
102    17977   20649   2930    44     74      cpuinfo
103    17988   2068    2974    443    7443    crypto
104    18      2069    3       450    76      devices
107    18004   20698   30      451    77      diskstats
108    18046   20706   3059    4510   78      dma
1084   18085   2071    31      453    79      driver
1086   182     2074    314     454    8       execdomains
1087   1821    20769   3179    455    80      fb
11     1828    2080    318     456    8066    filesystems
11004  1834    20817   3183    458    81      fs
112    1851    20858   319     459    8137    interrupts
1134   1852    2087    32      46     82      iomem
114    1853    20896   3230    460    8217    ioports
11425  1854    2090    3241    461    828     irq
11427  1855    20941   3330    47     829     kallsyms
1144   1856    20982   3332    48     83      kcore
11452  18567   2113    3345    481    831     keys
11494  1859    21132   34      49     833     key-users
1160   1860    21137   3414    50     834     kmsg
11697  1863    21139   3419    506    835     kpagecgroup
1172   1869    2130    3424    513    836     kpagecount
12     1882    2133    3435    52     8367    kpageflags
12406  1887    2148    3438    53     838     latency_stats
13     1894    2164    3450    54     839     loadavg
1336   1895    2165    3457    55     84      locks
1376   1896    2176    3468    551    843     mdstat
14     1897    2181    35      552    847     meminfo
14992  1898    2188    36      56     848     misc
15378  19      22      3604    5612   85      modules
1561   1901    2212    362     5619   853     mounts
15621  1915    2227    363     5718   8542    mtrr
1568   19187   2240    3689    58     856     net
159    19289   2246    37      5825   858     pagetypeinfo
16     1941    23      3704    5831   86      partitions
160    195     24      3718    5866   867     sched_debug

```

(2) Umgebungsvariablen des Prozesses: `tr '\0' '\n' < /proc/2022/envron`

```

kevin@Kevin-XPS15:~$ tr '\0' '\n' < /proc/2022/environ
USER=kevin
LANGUAGE=en_US
LC_TIME=de_AT.UTF-8
XDG_SEAT=seat0
XDG_SESSION_TYPE=x11
SSH_AGENT_PID=1821
SHLVL=0
QT4_IM_MODULE=xim
HOME=/home/kevin
DESKTOP_SESSION=budgie-desktop
QT_STYLE_OVERRIDE=
GTK_MODULES=gail:atk-bridge
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
LC_MONETARY=de_AT.UTF-8
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
QT_QPA_PLATFORMTHEME=gtk2
IM_CONFIG_PHASE=2
LOGNAME=kevin
GTK_IM_MODULE=ibus
XDG_SESSION_ID=c2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
GDM_LANG=en_US
LC_ADDRESS=de_AT.UTF-8
XDG_RUNTIME_DIR=/run/user/1000
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
DISPLAY=:0
LANG=en_US.UTF-8
XDG_CURRENT_DESKTOP=Budgie:GNOME
LC_TELEPHONE=de_AT.UTF-8
XDG_SESSION_DESKTOP=budgie-desktop
XMODIFIERS=@im=ibus
XAUTHORITY=/home/kevin/.Xauthority
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/kevin
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
LC_NAME=de_AT.UTF-8
SHELL=/bin/bash
QT_ACCESSIBILITY=1
GDMSESSION=budgie-desktop
LC_MEASUREMENT=de_AT.UTF-8
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
LC_IDENTIFICATION=de_AT.UTF-8
XDG_VTNR=7
QT_IM_MODULE=ibus
PWD=/home/kevin
CLUTTER_IM_MODULE=xim
XDG_DATA_DIRS=/usr/share/budgie-desktop:/usr/local/share:/usr/share:/var/lib/napd/desktop
XDG_CONFIG_DIRS=/etc/xdg/xdg-budgie-desktop:/etc/xdg
LC_NUMERIC=de_AT.UTF-8
LC_PAPER=de_AT.UTF-8

```

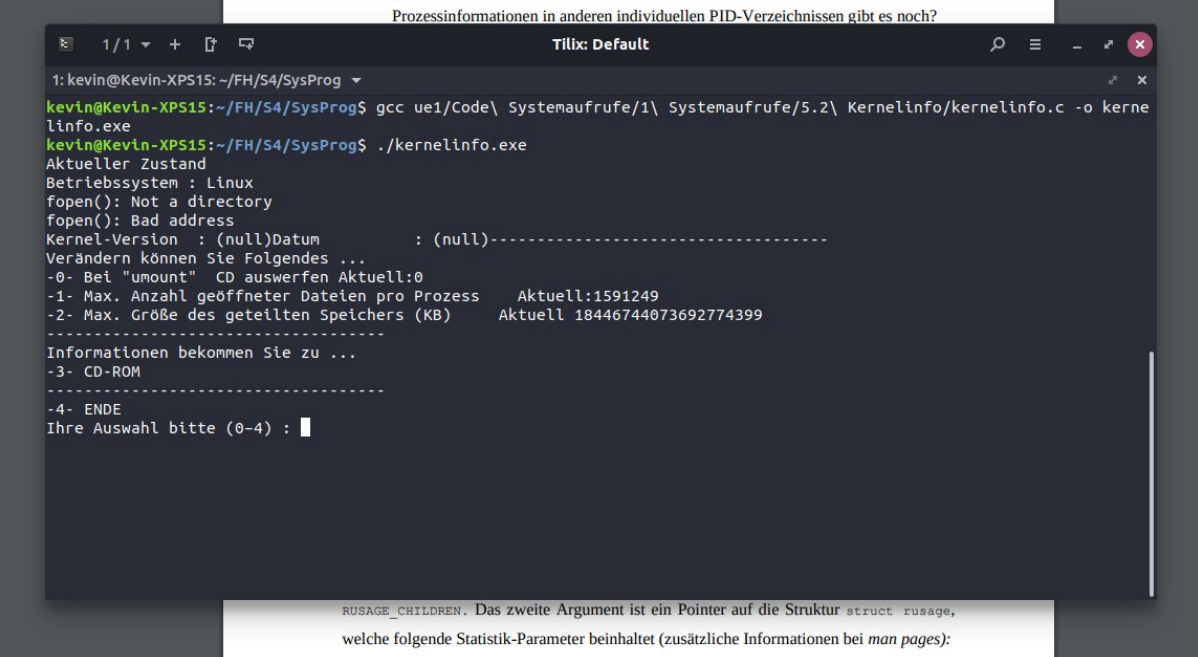
(3) Aktuelles Arbeitsverzeichnis des Prozesses: ls -l /proc/2022/cwd

```

kevin@Kevin-XPS15:~$ ls -l /proc/2022/cwd
lrwxrwxrwx 1 kevin kevin 0 Mär  7 19:57 /proc/2022/cwd -> /home/kevin

```

## 5.2 Kernel-Informationen auslesen und konfigurieren



Prozessinformationen in anderen individuellen PID-Verzeichnissen gibt es noch?

```

1: kevin@Kevin-XPS15: ~/FH/S4/SysProg
kevin@Kevin-XPS15:~/FH/S4/SysProg$ gcc ue1/Code\ Systemaufrufe/1\ Systemaufrufe/5.2\ kernelinfo/kernelinfo.c -o kernelinfo.exe
kevin@Kevin-XPS15:~/FH/S4/SysProg$ ./kernelinfo.exe
Aktueller Zustand
Betriebssystem : Linux
fopen(): Not a directory
fopen(): Bad address
Kernel-Version : (null)Datum : (null)-----
Verändern können Sie Folgendes ...
-0- Bei "umount" CD auswerfen Aktuell:0
-1- Max. Anzahl geöffneter Dateien pro Prozess Aktuell:1591249
-2- Max. Größe des geteilten Speichers (KB) Aktuell 18446744073692774399
-----
Informationen bekommen Sie zu ...
-3- CD-ROM
-----
-4- ENDE
Ihre Auswahl bitte (0-4) : 
  
```

RUSAGE\_CHILDREN. Das zweite Argument ist ein Pointer auf die Struktur struct rusage, welche folgende Statistik-Parameter beinhaltet (zusätzliche Informationen bei *man pages*):

1. Testen Sie, ob damit die vorhandenen Parameter im Kernel geändert werden können. Wenn nicht, wie könnte das konfiguriert werden?

Mit root Rechten ausführen

2. Welche Datei im Verzeichnis /etc müsste verändert werden, um eine dauerhafte Änderung auch nach Systemstart zu erreichen?

/etc/sysctl.conf

3. Mit welchem Kommando könnte aus dem obigen Verzeichnis z.B. der Typ und das aktuelle Release des Betriebssystems ausgelesen werden?

cat /etc/\*-release



## 5.3 Prozess-Statistiken auslesen

```
1  #include <stdio.h>
2  #include <sys/resource.h>
3  #include <sys/time.h>
4  #include <unistd.h>
5
6  int main (void){
7      struct rusage usage;
8      getrusage (RUSAGE_SELF, &usage);
9      printf ("CPU time: %ld.%06ld sec user, %ld.%06ld sec system\n",
10             usage.ru_utime.tv_sec,
11             usage.ru_utime.tv_usec,
12             usage.ru_stime.tv_sec,
13             usage.ru_stime.tv_usec);
14 }
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ gcc getusage53.c -o gr.exe
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ./gr.exe
CPU time: 0.000000 sec user, 0.000560 sec system
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$
```

## 5.4 Benutzerverwaltung des Systems

a)

```
kevin@Kevin-XPS15:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Benutzername:Passwort (zu finden in /etc/shadow):User ID:Group  
ID:Kommentarfeld:Heimatverzeichnis:Shell

```

C getusage53.c  C benutzer54.c  C rechnerinfoc
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <pwd.h>
5  #include <string.h>
6
7  int main(int argc, char *argv[]) {
8      struct passwd *pas_ptr;
9      if( argc != 2 ) {
10         fprintf(stderr, "Usage: %s Name\n", argv[0]);
11         exit (EXIT_FAILURE);
12     }
13     /* Alternativ mit getpwnam() */
14     /* pas_ptr=getpwnam(argv[1], NULL, 10); */
15
16     pas_ptr = getpwnam(argv[1]);
17     if(pas_ptr == NULL) {
18         printf("Konnte nichts über %s ermitteln\n", argv[1]);
19         exit (EXIT_FAILURE);
20     }
21
22     printf("Folgende Angaben wurden ermittelt: \n");
23     printf("Benutzername      : %s\n", pas_ptr->pw_name);
24     printf("Benutzernummer      : %d\n", pas_ptr->pw_uid);
25     printf("Gruppennummer       : %d\n", pas_ptr->pw_gid);
26     printf("Kommentar           : %s\n", pas_ptr->pw_gecos);
27     printf("Loginverzeichnis    : %s\n", pas_ptr->pw_dir);
28     printf("Loginshell          : %s\n", pas_ptr->pw_shell);
29     return EXIT_SUCCESS;
30 }

```

```

1: kevin@Kevin-XPS15: ~/FH/S4/SysProg/ue1
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ gcc -Wall ./benutzer54.c -o user.exe
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ./user.exe kevin
Folgende Angaben wurden ermittelt:
Benutzername      : kevin
Benutzernummer     : 1000
Gruppennummer      : 1000
Kommentar          : Kevin,,
Loginverzeichnis   : /home/kevin
Loginshell         : /bin/bash
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$

```

## 5.5 Koordination von Dateizugriffen

Aufruf von lockfile.exe mit Terminal 1. Aufruf von lockfile.exe mit Terminal 2

```

kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ gcc -Wall -o lockfile.exe ./Code\ Systemaufrufe\1\ Systemaufrufe\5.5\ Koordina
tion\ File\lockfile.c
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ touch test
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ./lockfile.exe test
opening test
locking
locked; hit Enter to unlock...

```

```

1: kevin@Kevin-XPS15: ~/FH/S4/SysProg/ue1
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ./lockfile.exe test
opening test
locking

```

Terminal 1 bekommt lock, Terminal 2 wartet bis lock erhalten wird

```

kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ gcc -Wall -o lockfile.exe ./Code\ Systemaufrufe\1\ Systemaufrufe\5.5\ Koordina
tion\ File\lockfile.c
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ touch test
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ./lockfile.exe test
opening test
locking
locked; hit Enter to unlock...
unlocking
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$

```

```

1: kevin@Kevin-XPS15: ~/FH/S4/SysProg/ue1
kevin@Kevin-XPS15:~/FH/S4/SysProg/ue1$ ./lockfile.exe test
opening test
locking
locked; hit Enter to unlock...

```

Terminal 1 löst lock, Terminal 2 erhält lock

```

1  #include <fcntl.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <unistd.h>
5  int main (int argc, char* argv[]){
6  char* file = argv[1];
7  int fd;
8  struct flock lock;
9  printf ("opening %s\n", file);
10
11  /* opening a file in write only */
12  fd = open (file, O_WRONLY);
13  printf ("locking\n");
14
15  /* fill lock with 0, set lock type exclusive */
16  memset (&lock, 0, sizeof(lock));
17  lock.l_type = F_WRLCK;
18
19  /* record lock with wait is set */
20  fcntl (fd, F_SETLKW, &lock);
21  printf ("locked; hit Enter to unlock... ");
22
23  /* wait for input */
24  getchar ();
25  printf ("unlocking\n");
26
27  /* set lock type to unlock, record unlock, close file*/
28  lock.l_type = F_UNLCK;
29  fcntl (fd, F_SETLKW, &lock);
30  close (fd);
31  return 0;
32  }

```

## 6 Kontrollfragen Systemschnittstellen

1. Was ist bei Kommandozeilenprogrammen der Unterschied zwischen Argumenten und Optionen?

Ein command ist aufgeteilt in Argumente. Optionen sind definierte und dokumentierte Argumente

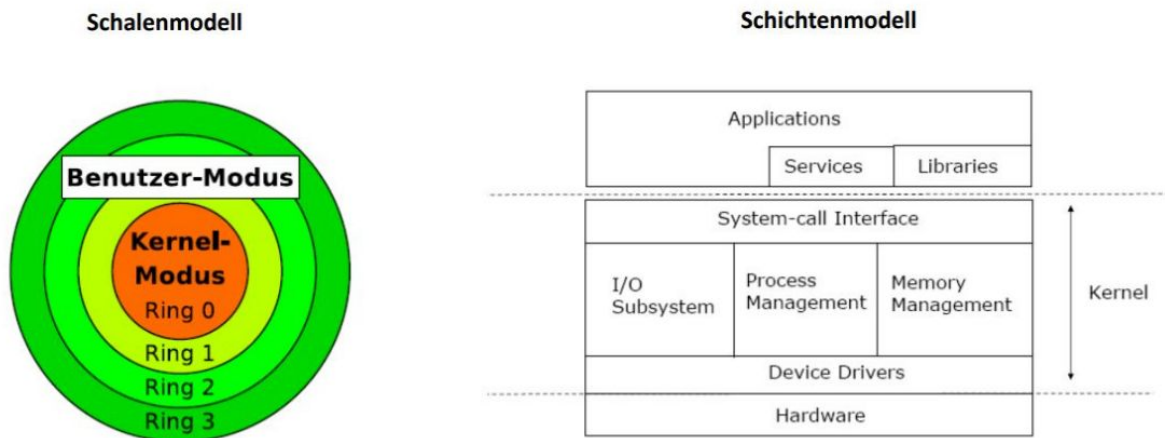
2. Wie können externe Bibliotheken mit dem gcc-Compiler eingebunden werden? Zeigen Sie die Lösung anhand eines Beispiels.

-L (Verzeichnis der Bibliotheken) -l (Name der Bibliothek)

gcc -o prog src.o -L./extlib/ -llibname



3. Beschreiben Sie die grundlegende Architektur eines Betriebssystems mit den entsprechenden Schnittstellen. In welchem Bereich befindet sich die Systemnahe Programmierung?



4. Wodurch unterscheiden sich monolithische und Microkernel-Betriebssystemarchitekturen?

Microkernel Architektur: Der Kernel beinhaltet nur das Minimum an Funktions-Sets zur Speicher-, Prozessverwaltung und zur Kommunikation zwischen Prozessen.

Monolithische Architektur: Der Kernel beinhaltet neben zusätzlich zum Funktionsumfang eines Microkernels noch weitere Funktionen, zB Treiber.

5. What do traps and interrupts have in common? How are they different? Give an example of each.

Trap: User generierte Exception. Zum Beispiel: Invalid Memory Access

Interrupt: Von der Hardware generiert.

Beide sind unvorhersehbar und veranlassen die CPU in einen Interrupt Handler zu springen.

6. Give examples of semantic gaps that must be bridged by software, including the Oss at the following levels: CPU (instruction) level, main memory level, secondary storage level, I/O device level.

Kompilierung eines Programms aber CPU kennt nur seine Instruktionssets.

Programm hält eine Liste von Strings aber Main Memory Level kennt nur bits und bytes.

Ein Datei soll gespeichert werden, der Secondary Storage Level kann nur Blöcke in Speichermedien schreiben.

Das Programm erwartet eine Bestätigung durch Enter die Tastatur liefert aber nur Press-Events

## 7. Warum soll ein Prozess im Benutzermodus keinen Aufruf direkt in den Kernel durchführen?

Um die Sicherheitsmechanismen nicht zu umgehen. Direkte Kernelaufufe können zum Crash des Systems führen, deswegen sollte man Sys Calls verwenden

## 8. Erklären Sie die Begriffe Kernel Mode und User Mode a) Was ist ein Moduswechsel? b) Nennen Sie Aktionen, die nur im Kernel Mode ausgeführt werden dürfen und geben Sie an, wie diese Einschränkung umgesetzt werden können. c) Welche Systemaufrufe gibt es unter Linux und wo sind diese z. B. in der Linux Cross-Reference definiert? Was sind die wichtigsten kernel call Klassen?

Kernel Mode: Programme in diesem Modus haben vollen, uneingeschränkten Zugriff auf das ganze System. Fehler können zu fatalen Resultaten führen.

User Mode: Eingeschränkter und kontrollierter Zugriff auf Kernel-Ressourcen.

a) Alle Prozesse beginnen die Ausführung im Benutzermodus, und sie wechseln nur dann in den Kernelmodus, wenn sie einen vom Kernel bereitgestellten Dienst erhalten.

b) Lesen und Schreiben auf das Dateisystem. stdio library sollte verwendet werden auf Datei zuzugreifen.

c) `sys_restart_syscall` (kernel/signal.c:2058), `sys_exit` (kernel/exit.c:1046), `sys_read` (fs/read\_write.c:391), `sys_write` (fs/read\_write.c:408)

Process Management / Scheduling, Memory Management, File management, I/O Management, Time Management, Signal Management, Interprocess Communication

## 9. Wie kann in einem Betriebssystem mit der HW kommuniziert werden? Beschreiben Sie mögliche Techniken und deren Funktionsweise.

Über HAL (Hardware Abstraction Layer). Hardwareereignisse werden abgefangen und zum Betriebssystem übertragen und umgekehrt. HAL schützt die Computerkomponenten auch die Software vor schlechtem Verhalten. Außerdem haben wir Treiber, die dem Betriebssystem helfen, die Hardware der Maschine genau zu identifizieren.

## 10. Was ist der wesentliche Unterschied zwischen einem Systemaufruf mit oder ohne Library?

Systemaufrufe können zu Komplikationen führen. Libraries vereinfachen den Zugriff.

## 11. Beschreiben Sie einen System Call anhand eines Codebeispiels

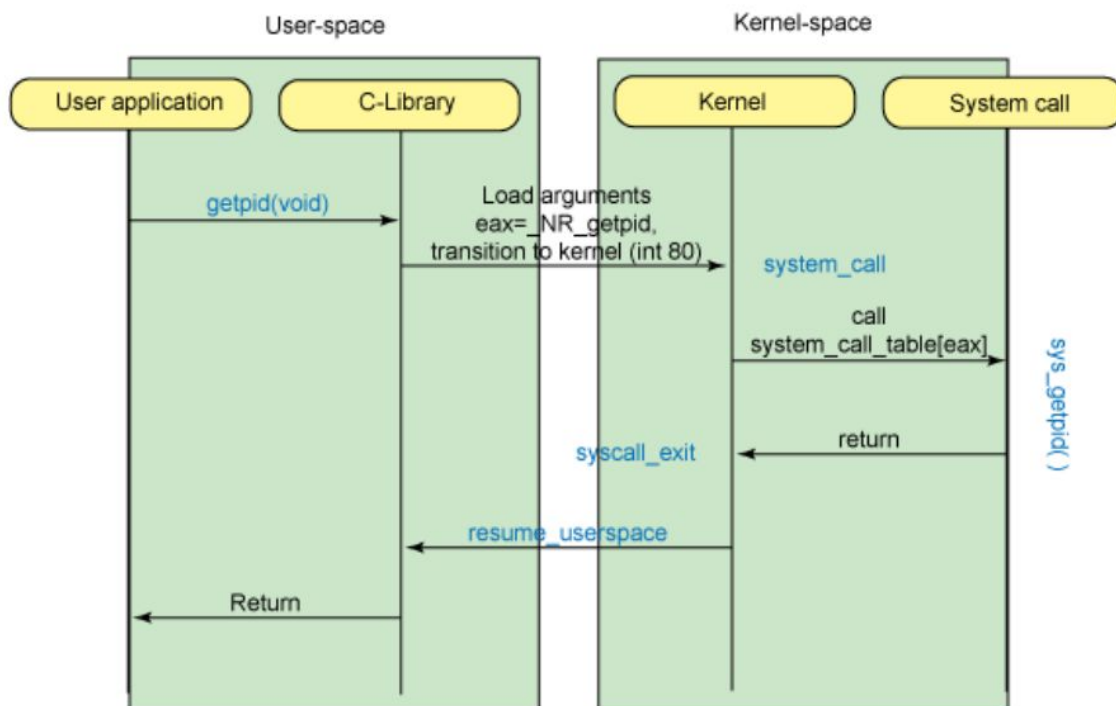
```
1  #define _GNU_SOURCE
2  #include <unistd.h>
3  #include <sys/syscall.h>
4  #include <sys/types.h>
5  int
6  main(int argc, char *argv[])
7  {
8      pid_t tid;
9      tid = syscall(SYS_gettid);
10     tid = syscall(SYS_tgkill, getpid(), tid);
11 }
```

## 12. Zeigen Sie in der Linux Cross-Reference (<https://elixir.bootlin.com/linux/latest/source>) Systemaufrufe des Linux-Kernels

/ kernel / signal.c

```
2705  /**
2706   * sys_restart_syscall - restart a system call
2707   */
2708  SYSCALL_DEFINE0(restart_syscall)
2709  {
2710     struct restart_block *restart = &current->restart_block;
2711     return restart->fn(restart);
2712 }
```

### 13. Beschreiben Sie die Bedeutung und Funktionsweise folgender Abbildung



Eine "User application" ruft die Funktion "getpid" einer C-Library auf. Diese Funktion vollzieht einen Moduswechsel in den Kernel-space. Im Kernel-space wird der System call `sys_getpid()` ausgeführt und das Resultat bis zur User application retourniert.