INF 208 LAB PRÜFUNG

Informatik – 2

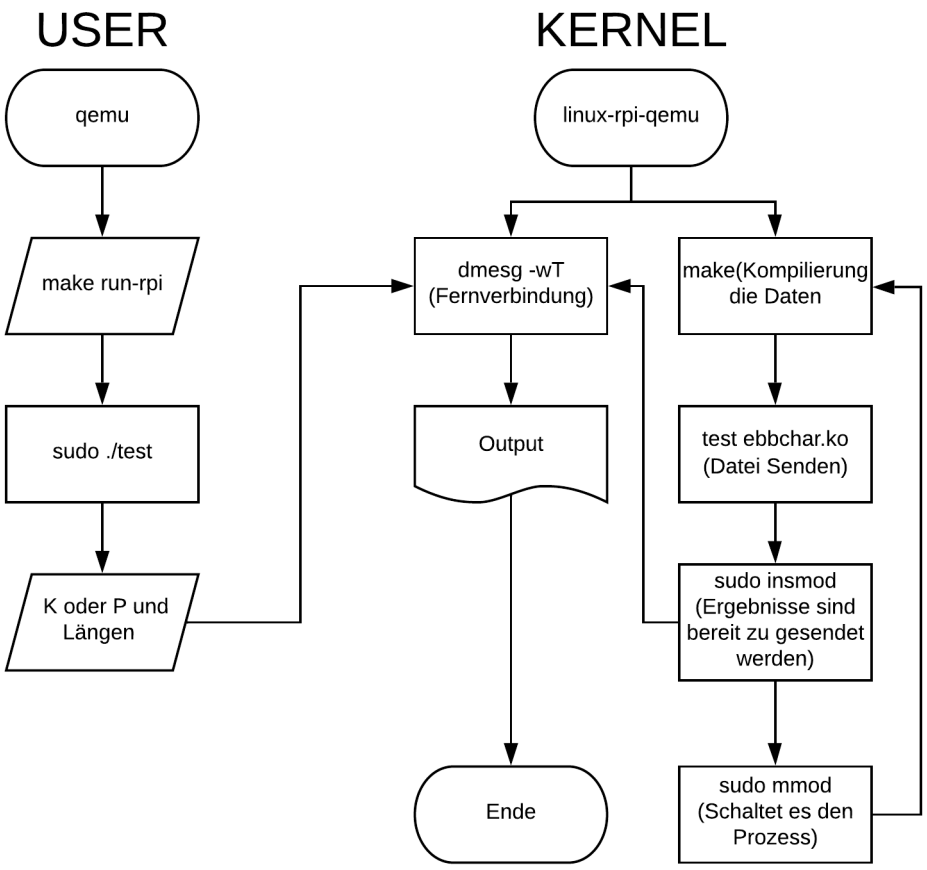
Kamil Uğur Sarp

170503033

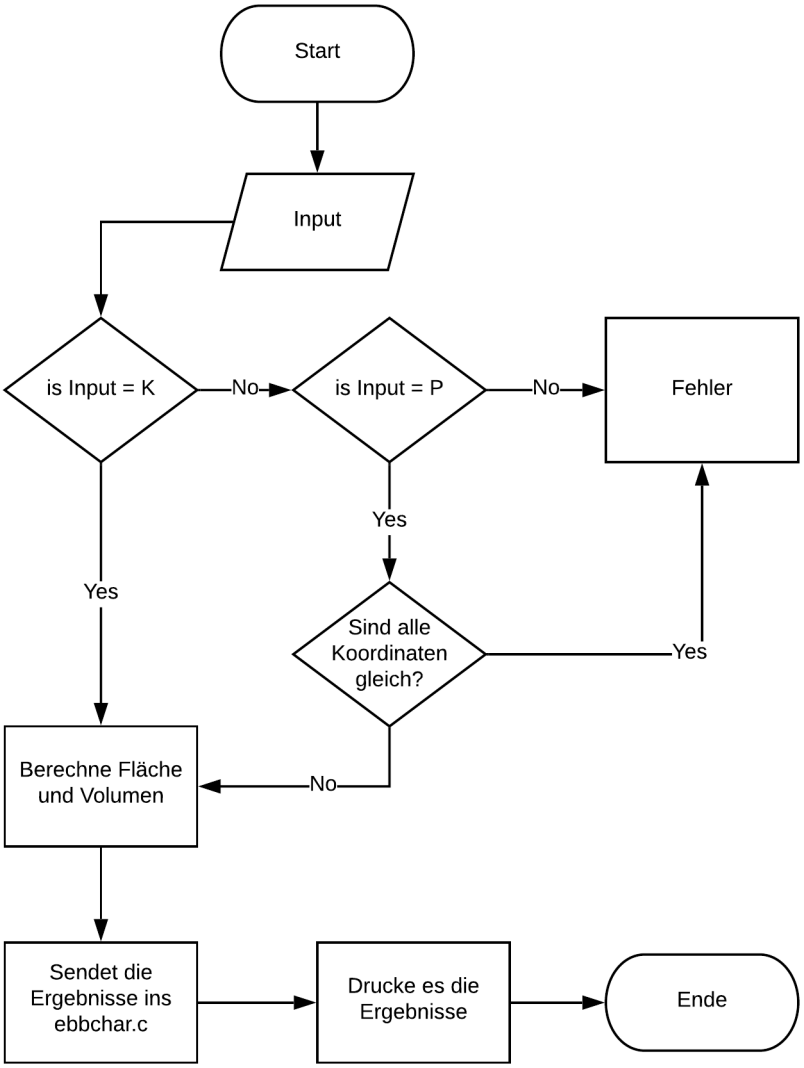
16.06.2020

2019/2020

1)



Das erste, was wir tun, um das Programm auszuführen make run-rpi. Auf diese Weise stellen wir die Verbindung unseres Programms mit der RPI-Umgebung her. Es gibt einige Befehle, die wir ausführen müssen, um eine Verbindung zur Kernelumgebung herzustellen. Wenn wir dmesg -wT Befehl schreiben, machen wir die Fernverbindung zwischen qemu und linux-rpi-qemu. Danach müssen wir die Daten kompilieren. Wir schreiben die beiden anderen Befehle, um eine Fernverbindung herzustellen. Nachdem wir alle diese Verbindungen bereitgestellt haben, können wir das Hauptprogramm ausführen. Nachdem die Prozesse in unserem Hauptprogramm abgeschlossen sind, werden die Ergebnisse von der qemu-Umgebung in die Kernel-Umgebung übertragen. Alle Ergebnisse werden in dieser Umgebung gedruckt.



Was von uns verlangt wird, ist ein Programm, das die Fläche und das Volumen des Prismas oder Kegels berechnet. Nach dem Ausführen des Programms wird erwartet, dass wir k oder p als Eingabewert eingeben. Wenn wir keine dieser beiden Optionen eingeben, wird unser Programm mit der Fehlermeldung geschlossen. Wenn unsere Eingaben korrekt sind, werden wir aufgefordert, Werte entsprechend dem von uns ausgewählten Objekt einzugeben. Wenn unsere ausgewählte Option Prisma ist und die 3 eingegebenen Werte denselben Koordinatenpunkt angeben, wird ein diesbezüglicher Fehler angegeben. Im letzten Schritt werden alle Vorgänge ausgeführt und die Bereichs- und Volumeninformationen werden dem Benutzer gegeben.
Diese Informationen werden dann an das String-Format gesendet und an ebbchar übertragen.

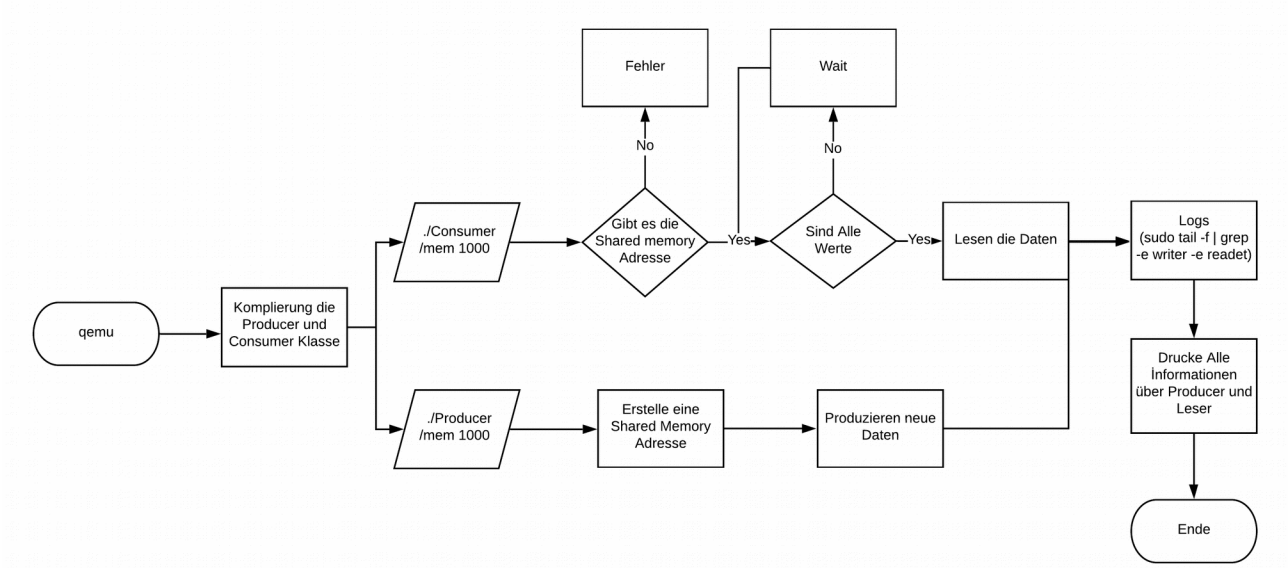
Qemu-Umgebung

```
pi@raspberrypi:~$ sudo ./test
Starting device test code example...
Bitte wahlen Sie die Form aus,die Sie mit den angegebenen Buchstaben berechnen möchten
:Dreieckiges Prisma (P,p) / Kegel (K,k)=P
Bitte geben Sie die Punkte X0,Y0,X1,Y1,X2,Y2 und h als Ganzezahlen ein
1.Koordinate X0:10
1.Koordinate Y0:20
2.Koordinate X1:30
2.Koordinate Y1:40
3.Koordinate X2:50
3.Koordinate Y2:60
4.hohe h:20
Grundflache= 1000
Volumen=20000
Press ENTER to read back from the device...
The received message is: [Prozess ist erfolgreich abgeschlossen!]
End of the program
pi@raspberrypi:~$ sudo ./test
Starting device test code example...
Bitte wahlen Sie die Form aus,die Sie mit den angegebenen Buchstaben berechnen möchten
:Dreieckiges Prisma (P,p) / Kegel (K,k)=K
Bitte geben Sie die Radius und h als Ganzzahlen ein
Radius:10
Hohe h:15
Grundflache=300
Volumen=1500
Press ENTER to read back from the device...
The received message is: [Prozess ist erfolgreich abgeschlossen!]
End of the program
pi@raspberrypi:~$ sudo ./test
Starting device test code example...
Bitte wahlen Sie die Form aus,die Sie mit den angegebenen Buchstaben berechnen möchten
:Dreieckiges Prisma (P,p) / Kegel (K,k)=S
 ihre wahl ist falch
Press ENTER to read back from the device...
The received message is: [Prozess ist erfolgreich abgeschlossen!]
End of the program
pi@raspberrypi:~$
```

Kernel Umgebung

```
[Mon Jun 15 13:11:33 2020] Die gewahlte Form ist [ Dreieckiges Prisma ]- (P)
[Mon Jun 15 13:11:33 2020] Grundflache und Volumen werden in Kurze berechnet
[Mon Jun 15 13:11:44 2020] EBBChar: 1000
[Mon Jun 15 13:11:44 2020] EBBChar: 20000
[Mon Jun 15 13:11:44 2020] EBBChar: Device successfully closed
[Mon Jun 15 13:11:59 2020] EBBChar: Device has been opened 2 time(s)
[Mon Jun 15 13:12:02 2020] EBBChar: K\xcd\xf9\xb6(G\xf9\xb6xY\xfc\xb6
[Mon Jun 15 13:12:02 2020] Die gewahlte Form ist [ Kegel ]- (K)
[Mon Jun 15 13:12:02 2020] Grundflache und Volumen werden in Kurze berechnet
[Mon Jun 15 13:12:11 2020] EBBChar: 300
[Mon Jun 15 13:12:11 2020] EBBChar: 1500
[Mon Jun 15 13:12:11 2020] EBBChar: Device successfully closed
[Mon Jun 15 13:12:18 2020] EBBChar: Device has been opened 3 time(s)
[Mon Jun 15 13:12:22 2020] EBBChar: SM\xf4\xb6(\xc7\xf3\xb6x\xd9\xf6\xb6
[Mon Jun 15 13:12:22 2020] EBBChar: ihre wahl ist falch
[Mon Jun 15 13:12:22 2020] EBBChar: Device successfully closed
```

3-a)



Zunächst müssen wir unseren Code kompilieren. Machen wir es mithilfe gcc intConsumer.c -o intConsumer -lrt -lpthread. Nach diesen Prozessen müssen wir das Sender- und Empfängerprogramm in zwei separaten Terminals ausführen (./intConsumer /mem 1000).
Zu diesem Zeitpunkt sind drei Parameter erforderlich, Producer oder Consumer, Shared Memory Adresse und Delay. Producer Klasse erstellt eine neue Shared Memory Adresse für sich selbst und für Consumer Klasse zu benutzen. Wenn es ergestellt ist, produziere sie die Daten, die wir brauchen. Wenn der Shared-Memory-Bereich nicht erstellt werden kann, endet das Programm mit einem Fehler. Wenn Producer fehlerfrei arbeitet, beginnt Consumer, nachdem Producer alle Werte erstellt hat, diese Werte nacheinander zu lesen. Da wir die Semaphorstruktur verwenden, beginnt der Lesestatus erst, wenn alle Werte erzeugt wurden. Wir verwenden auch Log, um diese Situationen zu beobachten(sudo tail -f | grep -e writer -e reader). Während das Programm ausgeführt wird, können wir auch die Änderung in Log gleichzeitig sehen.

Writer:

```
    sem_init(&shmp->full, 1, 0);
    sem_init(&shmp->empty, 1, 0);
    pthread_mutex_lock(&shmp->mutex);

    openlog("writer:" , LOG_NOWAIT , LOG_USER);

    char *array="Greetings from TAU";
    int size=strlen(array);
    for(int i=0;i<size;i++){
        char item = array[i];
        shmp->buffer[i]=item;
        printf("writer:Added character %c by %d\n",item,i);
        syslog(LOG_INFO,"writer: Ein Element , %c , wird ins Shared Memory geschrieben",shmp->buffer[i]);
        delay(milliseconds);

    }

    closelog();
    pthread_mutex_unlock(&shmp->mutex);
    sem_post(&shmp->full);
    exit(EXIT_SUCCESS);
}
```

Kurz gesagt verhindert der Prozess die Eingabe des gemeinsam genutzten Speichers, indem er mit der Semaphorstruktur gesperrt wird, bis das Programm abgeschlossen ist. Es teilt die Werte in unserem Array mit dem Verbraucher, indem es sie einzeln drückt.

Reader:

```
    sem_wait(&shmp->full);
    openlog("reader:" , LOG_NOWAIT , LOG_USER);
    while(1){
        pthread_mutex_lock(&shmp->mutex);
        printf("Text:%s\n",shmp->buffer);
        int size=strlen(shmp->buffer);
        for(int i=0;i<size;i++){
        char item = shmp->buffer[i];
        printf("reader:Read character %c\n",item);
        syslog(LOG_INFO,"reader : Ein Element , %c , wird ins Shared Memory gelesen",item);
        delay(1000);
        shmp->buffer[i]=shmp->buffer[i+1];

        }
        pthread_mutex_unlock(&shmp->mutex);

    }
    sem_post(&shmp->full);
    closelog();
    return 0;
```

```
dev@tau:~/Desktop/int$ ./intProducer /mem 1000
writer:Added character G by 0
writer:Added character r by 1
writer:Added character e by 2
writer:Added character e by 3
writer:Added character t by 4
writer:Added character i by 5
writer:Added character n by 6
writer:Added character g by 7
writer:Added character s by 8
writer:Added character   by 9
writer:Added character f by 10
writer:Added character r by 11
writer:Added character o by 12
writer:Added character m by 13
writer:Added character   by 14
writer:Added character T by 15
writer:Added character A by 16
writer:Added character U by 17
dev@tau:~/Desktop/int$

dev@tau:~/Desktop/int$ ./intConsumer /mem 1000
Text:Greetings from TAU
reader:Read character G
reader:Read character r
reader:Read character e
reader:Read character e
reader:Read character t
reader:Read character i
reader:Read character n
reader:Read character g
reader:Read character s
reader:Read character
reader:Read character f
reader:Read character r
reader:Read character o
reader:Read character m
reader:Read character
reader:Read character T
reader:Read character A
reader:Read character U
^Z
[2]+  Stopped                 ./intConsumer /mem 1000
dev@tau:~/Desktop/int$
```

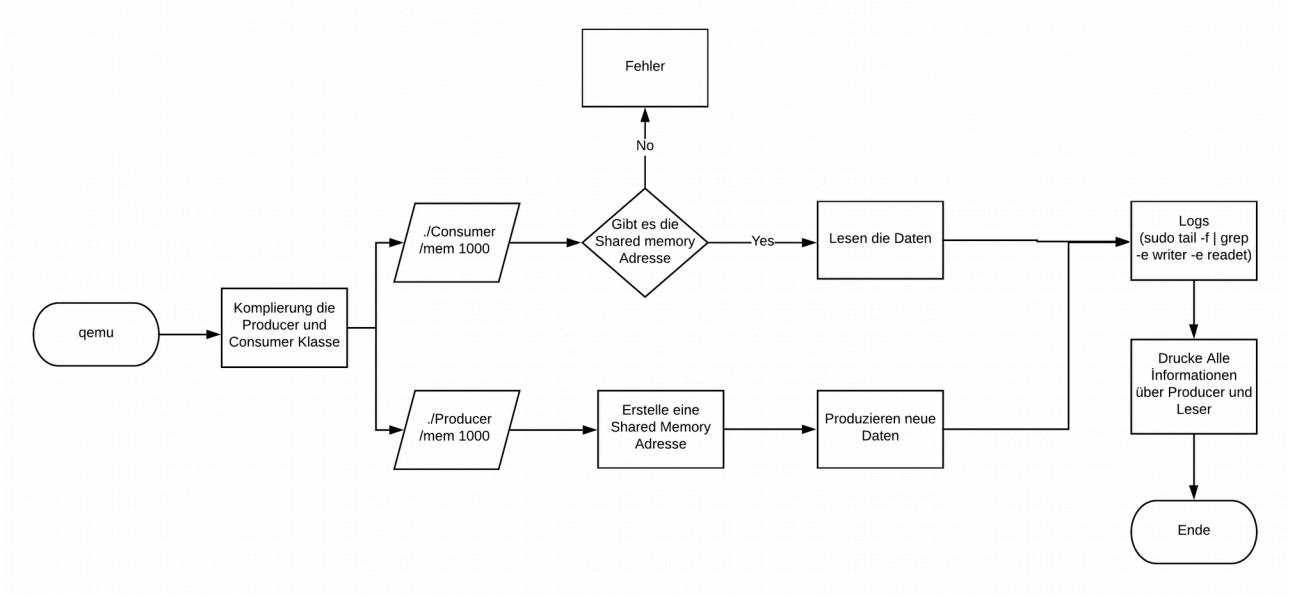
```
dev@tau:~/Desktop/int$ sudo tail -f /var/log/syslog | grep -e writer -e reader
Jun 15 15:39:38 tau writer:: message repeated 2 times: [ writer: Ein Element , G , wird ins Shared M
y geschrieben]
Jun 15 15:44:03 tau writer:: writer: Ein Element , G , wird ins Shared Memory geschrieben
Jun 15 15:44:04 tau writer:: writer: Ein Element , r , wird ins Shared Memory geschrieben
Jun 15 15:44:05 tau writer:: writer: Ein Element , e , wird ins Shared Memory geschrieben
Jun 15 15:44:06 tau writer:: writer: Ein Element , e , wird ins Shared Memory geschrieben
Jun 15 15:44:07 tau writer:: writer: Ein Element , t , wird ins Shared Memory geschrieben
Jun 15 15:44:08 tau writer:: writer: Ein Element , i , wird ins Shared Memory geschrieben
Jun 15 15:44:09 tau writer:: writer: Ein Element , n , wird ins Shared Memory geschrieben
Jun 15 15:44:10 tau writer:: writer: Ein Element , g , wird ins Shared Memory geschrieben
Jun 15 15:44:11 tau writer:: writer: Ein Element , s , wird ins Shared Memory geschrieben
Jun 15 15:44:12 tau writer:: writer: Ein Element ,  , wird ins Shared Memory geschrieben
Jun 15 15:44:13 tau writer:: writer: Ein Element , f , wird ins Shared Memory geschrieben
Jun 15 15:44:14 tau writer:: writer: Ein Element , r , wird ins Shared Memory geschrieben
Jun 15 15:44:15 tau writer:: writer: Ein Element , o , wird ins Shared Memory geschrieben
Jun 15 15:44:16 tau writer:: writer: Ein Element , m , wird ins Shared Memory geschrieben
Jun 15 15:44:17 tau writer:: writer: Ein Element ,  , wird ins Shared Memory geschrieben
Jun 15 15:44:18 tau writer:: writer: Ein Element , T , wird ins Shared Memory geschrieben
Jun 15 15:44:19 tau writer:: writer: Ein Element , A , wird ins Shared Memory geschrieben
Jun 15 15:44:20 tau writer:: writer: Ein Element , U , wird ins Shared Memory geschrieben
Jun 15 15:44:21 tau reader:: reader : Ein Element , G , wird ins Shared Memory gelesen
Jun 15 15:44:22 tau reader:: reader : Ein Element , r , wird ins Shared Memory gelesen
Jun 15 15:44:23 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:44:24 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:44:25 tau reader:: reader : Ein Element , t , wird ins Shared Memory gelesen
Jun 15 15:44:26 tau reader:: reader : Ein Element , i , wird ins Shared Memory gelesen
Jun 15 15:44:27 tau reader:: reader : Ein Element , n , wird ins Shared Memory gelesen
Jun 15 15:44:28 tau reader:: reader : Ein Element , g , wird ins Shared Memory gelesen
Jun 15 15:44:29 tau reader:: reader : Ein Element , s , wird ins Shared Memory gelesen
Jun 15 15:44:30 tau reader:: reader : Ein Element ,  , wird ins Shared Memory gelesen
Jun 15 15:44:31 tau reader:: reader : Ein Element , f , wird ins Shared Memory gelesen
Jun 15 15:44:32 tau reader:: reader : Ein Element , r , wird ins Shared Memory gelesen
Jun 15 15:44:33 tau reader:: reader : Ein Element , o , wird ins Shared Memory gelesen
Jun 15 15:44:34 tau reader:: reader : Ein Element , m , wird ins Shared Memory gelesen
Jun 15 15:44:35 tau reader:: reader : Ein Element ,  , wird ins Shared Memory gelesen
Jun 15 15:44:36 tau reader:: reader : Ein Element , T , wird ins Shared Memory gelesen
Jun 15 15:44:37 tau reader:: reader : Ein Element , A , wird ins Shared Memory gelesen
Jun 15 15:44:38 tau reader:: reader : Ein Element , U , wird ins Shared Memory gelesen
```

CPU

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3266 dev       20   0    8780    860    772 R  65.1  0.0   0:01.96 intProducer
 1312 root      20   0  688644  96644  36552 S  14.0  1.2   0:19.12 Xorg
 2029 dev       20   0  693532  59200  33936 S   3.7  0.7   0:06.08 x-terminal+
```

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3267 dev       20   0    8776    860    772 R  99.7  0.0   0:04.98 intConsumer
 1312 root      20   0  688644  96644  36552 S   0.7  1.2   0:19.44 Xorg
 1460 dev       20   0  185384   2820   2440 S   0.3  0.0   0:00.59 VBoxClient
```

3-b)



Der Algorithmus in dieser Frage ist fast der gleiche wie in a. Die einzige Änderung ist, dass wir Mutex anstelle der Semaphorstruktur verwenden. Da wir Mutex verwenden, wird der Consumer-Teil weiterhin ausgegeben, indem er sich mit jedem eingehenden Wert ändert, anstatt auf die Fertigstellung der gesamten Struktur zu warten.

Bsp:

Producer:	Consumer:
G	G
r	G r
e	G r e
e	G r e e
t	G r e e t
i	G r e e t i
n	G r e e t i n
g	G r e e t i n g
s	G r e e t i n g s

```
dev@tau:~/Desktop/pol$ ./pollProducer /mem 1000
Neue Charge wird produziert..
writer:Added Charackter G by 0
writer:Added Charackter r by 1
writer:Added Charackter e by 2
writer:Added Charackter e by 3
writer:Added Charackter t by 4
writer:Added Charackter i by 5
writer:Added Charackter n by 6
writer:Added Charackter g by 7
writer:Added Charackter s by 8
writer:Added Charackter   by 9
writer:Added Charackter f by 10
writer:Added Charackter r by 11
writer:Added Charackter o by 12
writer:Added Charackter m by 13
writer:Added Charackter   by 14
writer:Added Charackter T by 15
writer:Added Charackter A by 16
writer:Added Charackter U by 17
dev@tau:~/Desktop/pol$
```

```
Neue Charge wird gelesen..
reader:Read Element G
reader:Read Element r
reader:Read Element e
reader:Read Element e
reader:Read Element t
reader:Read Element i
reader:Read Element n
reader:Read Element g
reader:Read Element s
reader:Read Element
reader:Read Element f
reader:Read Element r
reader:Read Element o
reader:Read Element m
reader:Read Element
reader:Read Element T
reader:Read Element A
reader:Read Element U
^Z
[1]+  Stopped                 ./pollConsumer /mem 1000
dev@tau:~/Desktop/pol$
```

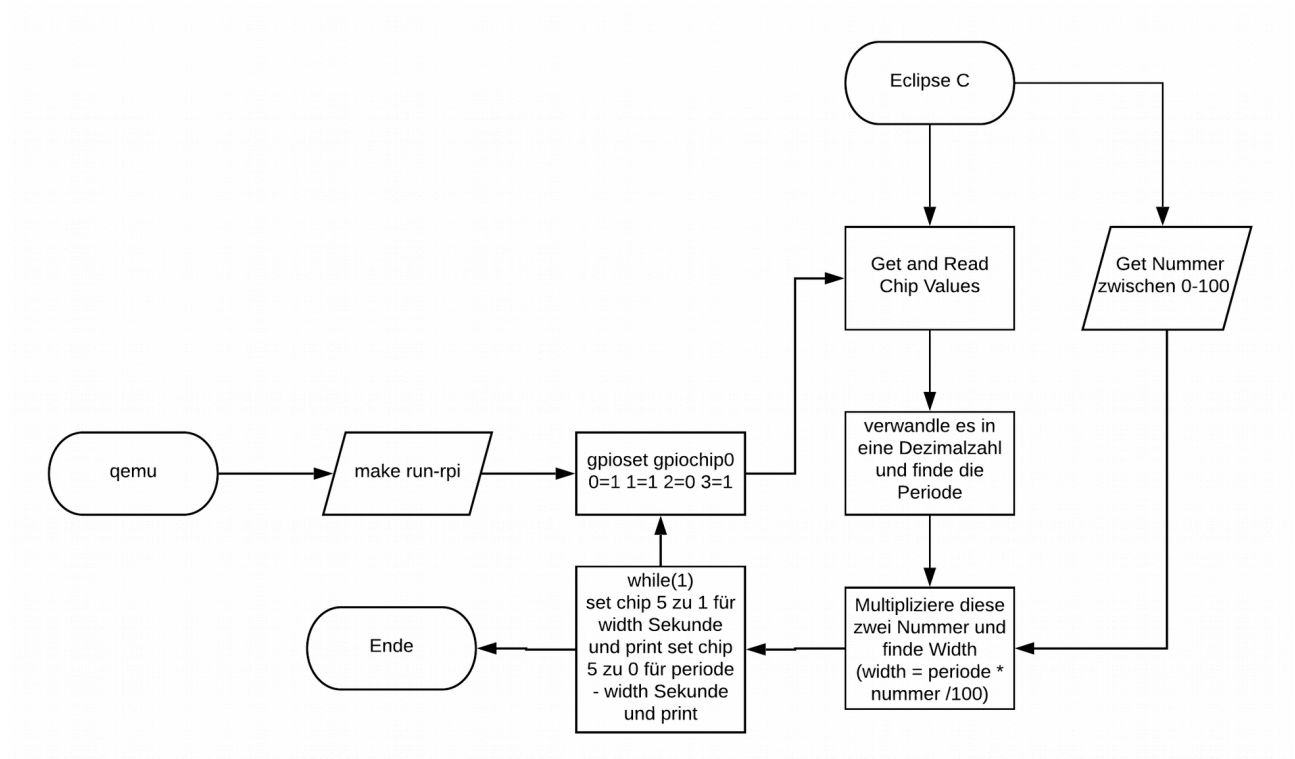
```
Jun 15 15:22:27 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:27 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:28 tau writer:: writer: Ein Element , G , wird ins Shared Memory geschrieben
Jun 15 15:22:28 tau reader:: reader : Ein Element , G , wird ins Shared Memory gelesen
Jun 15 15:22:28 tau reader:: reader : Ein Element , r , wird ins Shared Memory gelesen
Jun 15 15:22:28 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:28 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:28 tau reader:: reader : Ein Element , t , wird ins Shared Memory gelesen
Jun 15 15:22:29 tau writer:: writer: Ein Element , G , wird ins Shared Memory geschrieben
Jun 15 15:22:29 tau reader:: reader : Ein Element , G , wird ins Shared Memory gelesen
Jun 15 15:22:29 tau reader:: reader : Ein Element , r , wird ins Shared Memory gelesen
Jun 15 15:22:29 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:29 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:29 tau reader:: reader : Ein Element , t , wird ins Shared Memory gelesen
Jun 15 15:22:29 tau reader:: reader : Ein Element , i , wird ins Shared Memory gelesen
Jun 15 15:22:30 tau writer:: writer: Ein Element , G , wird ins Shared Memory geschrieben
Jun 15 15:22:30 tau reader:: reader : Ein Element , G , wird ins Shared Memory gelesen
Jun 15 15:22:30 tau reader:: reader : Ein Element , r , wird ins Shared Memory gelesen
Jun 15 15:22:30 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:30 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:30 tau reader:: reader : Ein Element , t , wird ins Shared Memory gelesen
Jun 15 15:22:30 tau reader:: reader : Ein Element , i , wird ins Shared Memory gelesen
Jun 15 15:22:30 tau reader:: reader : Ein Element , n , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau writer:: writer: Ein Element , G , wird ins Shared Memory geschrieben
Jun 15 15:22:31 tau reader:: reader : Ein Element , G , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau reader:: reader : Ein Element , r , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau reader:: reader : Ein Element , t , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau reader:: reader : Ein Element , i , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau reader:: reader : Ein Element , n , wird ins Shared Memory gelesen
Jun 15 15:22:31 tau reader:: reader : Ein Element , g , wird ins Shared Memory gelesen
Jun 15 15:22:32 tau writer:: writer: Ein Element , G , wird ins Shared Memory geschrieben
Jun 15 15:22:32 tau reader:: reader : Ein Element , G , wird ins Shared Memory gelesen
Jun 15 15:22:32 tau reader:: reader : Ein Element , r , wird ins Shared Memory gelesen
Jun 15 15:22:32 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
Jun 15 15:22:32 tau reader:: reader : Ein Element , e , wird ins Shared Memory gelesen
```

Wie wir im Protokol sehen können, wird es mit den Werten gelesen, die für jeden neuen Buchstaben im Speicher verbleiben.

CPU

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3017	dev	20	0	8780	856	768	R	100.0	0.0	0:11.44	pollProduc+
3018	dev	20	0	8776	836	748	R	100.0	0.0	0:10.73	pollConsum+
1312	root	20	0	688644	96592	36500	S	2.3	1.2	0:15.42	Xorg
435	root	19	-1	94804	14188	13456	S	0.7	0.2	0:00.27	systemd-jo+

4)



Zuerst müssen wir das rpi ausführen. Wir geben dann die ersten 4 Pins 1 oder 0, um den Periodenwert zu erstellen (gpioset gpiochip0 0=1 1=1 2=0 3=1). Nach diesem Schritt müssen wir die Pin-Werte in der RPI-Umgebung mit unserem Code in Eclipse lesen. Um den Punkt zu finden, konvertieren wir die Werte in den von uns gelesenen Pins in Dezimalzahlen. Um die Impulsbreite zu berechnen, müssen wir vom Benutzer eine Zahl zwischen 0 und 100 erhalten. Wir multiplizieren diese beiden Werte und teilen sie durch 100. Dieser erste Wert, den wir gefunden haben, gibt Pin5 an, wie viele Sekunden 1 wert sind. Periode - width ist, wie viele Sekunden 0 sein werden. Wir brauchen mindestens 10 Impulse, also wiederholen wir diesen Vorgang in der while-Schleife.
Bsp:
gpioset gpiochip0 0=1 1=1 2=0 3=1 => 1101 machen wir diese Wert Decimal
(2^3 * 1) + (2^2 * 1) + (2^1 * 0) + (2^0 * 1) = 13 Unsere Periode ist 13
eingegebene Nummer ist 60
13 * 60 /100 = 7,8 Sekunde Pin5 ist 1
13 – 7,8 = 5,2 Sekunde Pin5 ist 0

```c
#include <gpiod.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#define GPIO_DEVICE "/dev/gpiochip0"
int main(void) {

    struct gpiod_chip *chip = NULL;
    struct gpiod_line *line = NULL;
    int req = -1;
    int value = 0;
    int toplam = 0;
    int k = 8;
    float girdi = 0;
    float pwm;

    chip = gpiod_chip_open(GPIO_DEVICE);
    if(!chip){
        return 1;
    }

    printf("0 ile 100 arasında bir sayı giriniz: ");
    scanf("%f",&girdi);
    if(girdi<0 || girdi>100){
        printf("Hatalı format");
        return -1;
    }

    for(int i=0; i<4; i++){
        line = gpiod_chip_get_line(chip, i);
        req = gpiod_line_request_input(line, "gpio_state");
        value = gpiod_line_get_value(line);
```

```c
        printf("Pin %d:%d ", i, value);
        toplam += k * value;
        k = k/2;
    }

    pwm = (toplam * girdi)/100;
    printf("\nPeriyot: %d \n", toplam);
    while(1){
        gpiod_ctxless_set_value(GPIO_DEVICE, 4, 1, false, "gpio_toggle", NULL, NULL);
        printf("Pin 4 ist %d %.2f sek. lang\n", 1, pwm);
        sleep(pwm);
        gpiod_ctxless_set_value(GPIO_DEVICE, 4, 0, false, "gpio_toggle", NULL, NULL);
        printf("Pin 4 ist %d %.2f sek. lang\n", 0, toplam-pwm);
        sleep(toplam-pwm);

    }
    gpiod_chip_close(chip);
    return 0;

    return EXIT_SUCCESS;
}
```

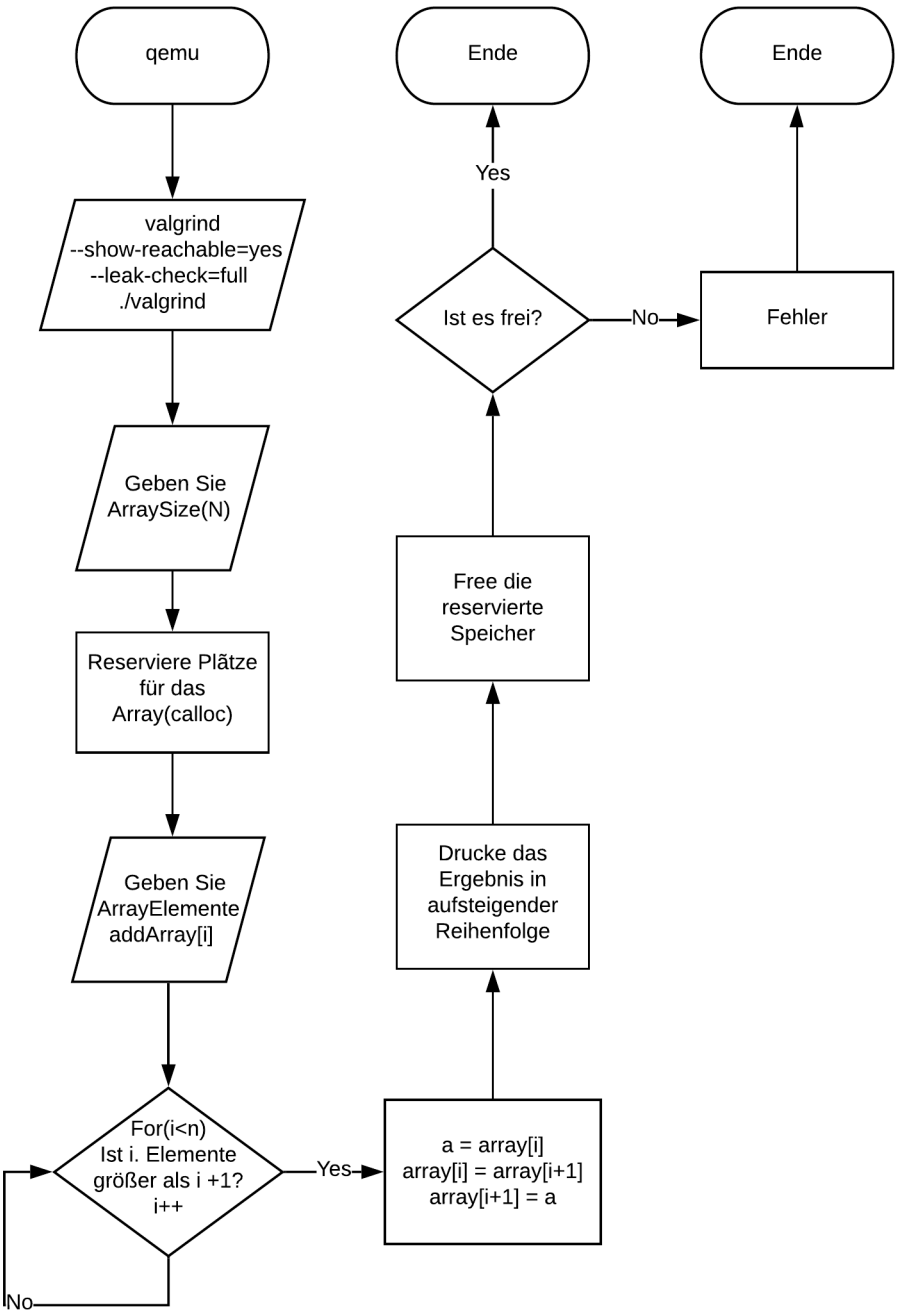
In der for-Schleife lesen wir die Werte von Pin0 bis Pin4. In diesem Schritt werden auch eingehende Pin-Werte in Dezimalzahlen konvertiert. Da der Pin0-Wert 8 in Dezimalzahl entspricht, müssen wir mit 8 multiplizieren, wenn dieser Pin-Wert 1 ist. Pin1 ist gleich 4, wenn 1. Der als k ermittelte Wert wurde also von 8 aus gestartet. Und wenn der Pin-Wert zunimmt, teilen wir ihn durch 2, um den aktuellen Wert zu berechnen. Wir haben diesen Wert in “total” gespeichert. Dann berechneten wir die Pulsbreite und ergriffen die notwendigen Maßnahmen.

```
pi@raspberrypi:~$ gpioset gpiochip0 0=1 1=1 2=0 3=1
pi@raspberrypi:~$ gpioget gpiochip0 0 1 2 3
1 1 0 1
pi@raspberrypi:~$
```

```
pi@raspberrypi:~$ /home/pi/inf208;exit
0 ile 100 arasında bir sayı giriniz: 75
75
Pin 0:1 Pin 1:1 Pin 2:0 Pin 3:1
Periyot: 13
Pin 4 ist 1 9.75 sek. lang
Pin 4 ist 0 3.25 sek. lang
Pin 4 ist 1 9.75 sek. lang
Pin 4 ist 0 3.25 sek. lang
Pin 4 ist 1 9.75 sek. lang
Pin 4 ist 0 3.25 sek. lang
Pin 4 ist 1 9.75 sek. lang
Pin 4 ist 0 3.25 sek. lang
Pin 4 ist 1 9.75 sek. lang
Pin 4 ist 0 3.25 sek. lang
Pin 4 ist 1 9.75 sek. lang
Pin 4 ist 0 3.25 sek. lang
Pin 4 ist 1 9.75 sek. lang
```

```
pi@raspberrypi:~$ gpioget gpiochip0 4
0
pi@raspberrypi:~$ gpioget gpiochip0 4
1
pi@raspberrypi:~$ gpioget gpiochip0 4
1
pi@raspberrypi:~$ gpioget gpiochip0 4
1
pi@raspberrypi:~$ gpioget gpiochip0 4
0
pi@raspberrypi:~$ gpioget gpiochip0 4
1
pi@raspberrypi:~$
```

5)



Valgrind ist ein Programmierwerkzeug für das Debuggen von Speicher, die Erkennung von Speicherlecks und die Profilerstellung. Wenn wir normal ausgeführt werden, schlägt unser Programm möglicherweise nicht fehl, aber aufgrund des begrenzten Speichers in eingebetteten Systemen können im Hintergrund Speicherprobleme auftreten. Das Programm fordert den Benutzer zunächst auf, zuerst die Arraygröße anzugeben. Die Calloc Function weist für jede Werte in Array int ** Speicherplatz zu. Dann gibt der Benutzer den ganzzahligen Wert ebenso ein wie die Arraygröße. Nachdem der Benutzer alle Nummern eingegeben hat, unser Programm zeichnet nach diesem Schritt jedes Element im Array in aufsteigender Reihenfolge auf. Die Werte werden in aufsteigender Reihenfolge an den Benutzer zurückgegeben. Der von calloc zugewiesene Platz endet mit dem Befehl free.

```
dev@tau:~/Desktop$ gcc valgrind.c -o valgrind
valgrind.c: In function 'main':
valgrind.c:15:21: error: 'n' undeclared (first use in this function)
     for(int i=0; i< n; i++){
                     ^
valgrind.c:15:21: note: each undeclared identifier is reported only once for each function it appears in
valgrind.c:28:33: error: subscripted value is neither array nor pointer nor vector
         add_array[i] = &(*array)[i];
                                 ^
valgrind.c:34:19: warning: assignment makes integer from pointer without a cast [-Wint-conversion]
                 a = add_array[i];
                   ^
valgrind.c:36:30: warning: assignment makes pointer from integer without a cast [-Wint-conversion]
                 add_array[j] = a;
                              ^
valgrind.c:42:18: warning: format '%d' expects argument of type 'int', but argument 2 has type 'int *' [-Wformat=]
         printf("%d\n", add_array[i]);
                 ~^     ~~~~~~~~~~~
                 %ls
dev@tau:~/Desktop$
```

Wenn wir versuchen, das Programm in seiner ursprünglichen Form zu starten, treten einige Fehler auf. Wir müssen klein n groß N machen, weil kleines n in unserem Programm nicht definiert ist. Wir erhalten die Fehlermeldung "Der Wert ist weder Array noch Zeiger oder Vektor". Wir müssen das * Zeichen vor dem &array[i] löschen. Wir erhalten den Fehler "Zeiger von Ganzzahl ohne Umwandlung" Um diesen Fehler zu vermeiden, fügen wir * vor add_array hinzu.
In diesem Fall scheint unser Programm ordnungsgemäß zu funktionieren.
Aber wenn wir Valgrind Analysis machen, bekommen wir ein paar Fehler. Wir erhalten Definitely Lost und Still reachable Fehler. Durch Löschen der for-Schleife außerhalb von calloc und durch Hinzufügen der freien Funktion wird diese Situation ebenfalls verhindert.

```
dev@tau:~/Desktop$ valgrind --show-reachable=yes --leak-check=full ./valgrind
==18368== Memcheck, a memory error detector
==18368== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18368== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==18368== Command: ./valgrind
==18368==
Geben Sie die Groesse des Arrays ein, N=6
Geben Sie die Nummern des Arrays ein
Element[0]=12
Element[1]=2
Element[2]=-3
Element[3]=27
Element[4]=-7
Element[5]=11
Die in aufsteigender Reihenflde angeordneten Nummern sind unten angegeben
-7
-3
2
11
12
27
==18368==
==18368== HEAP SUMMARY:
==18368==     in use at exit: 0 bytes in 0 blocks
==18368==   total heap usage: 3 allocs, 3 frees, 2,096 bytes allocated
==18368==
==18368== All heap blocks were freed -- no leaks are possible
==18368==
==18368== For counts of detected and suppressed errors, rerun with: -v
==18368== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
dev@tau:~/Desktop$
```