

DM14 - 1. Obligatorisk opgave F.06

# System Call

Jacob Aae Mikkelsen - 191076

Ingen andre gruppe medlemmer

6. marts 2005

## Indhold

<b>1 Opgave beskrivelse</b>	<b>2</b>
<b>2 Analyse</b>	<b>2</b>
2.1 Hukommelses allokering . . . . .	2
2.2 Adresser i kernel/user-space . . . . .	2
2.3 Fejlhåndtering . . . . .	2
2.4 Samtidige kald . . . . .	3
<b>3 Implementation</b>	<b>3</b>
3.1 Indarbejdning af nyt systemkald . . . . .	3
3.2 Systemkaldet . . . . .	3
<b>4 Afprøvning</b>	<b>6</b>
4.1 Positive test . . . . .	6
4.2 Negative test . . . . .	7
<b>5 Konklusion</b>	<b>8</b>
<b>6 bilag</b>	<b>9</b>
6.1 Testresultater . . . . .	9
6.2 kokkenMSbox.h . . . . .	10
6.3 kokkenMSbox.c . . . . .	10
6.4 Test 1 . . . . .	12
6.5 Test 2 . . . . .	13
6.6 Test 3 . . . . .	13
6.7 Test 4 . . . . .	14
6.8 Test 5 . . . . .	15
6.9 Test 6 . . . . .	15
6.10 Test 7 . . . . .	16

## 1 Opgave beskrivelse

Dette er den første obligatoriske opgave i faget DM14 - operativsystemer, på IMADA - Syddansk Universitet, Odense i foråret 2006.

Opgaven består af at implementere et dobbelt systemkald i en 'User Mode Linux' kerne. Systemkaldet kan benyttes til 'message passing' mellem processer, og derfor skal man kunne både sende og hente beskeder til en 'message box'. 'Message boxen' laves som en „First In First Out kø“ og kan derfor overføre en eller flere beskeder mellem forskellige processer.

Systemkaldet skal være robust, det vil sige returnere de korrekte fejlkoder hvis benyttet forkert, og ikke fejle hvis det bliver stoppet midtvejs af en interrupt. Systemkaldene skal derfor testes og dokumenteres.

## 2 Analyse

Der er i opgaven vedlagt en FIFO kø implementeret i userspace linux. Den skal justeres i forhold til systemkaldet, hvor den skal benyttes i kernelspace. De følgende emner skal der tages højde for i implementationen.

### 2.1 Hukommelses allokering

Hvor i userspace funktionerne `malloc` og `free` benyttes, skal man i kernelspace bruge `kmalloc` og `kfree`. Funktionen `kmalloc` skal have en prioritets parameter[3], hvor jeg har valgt `GFP_DMA` som er til brug udelukkende med `kmalloc`. Ellers er det den eneste forskel, og disse kald er blot udskiftet i den udleverede kode.

### 2.2 Adresser i kernel/user-space

I opgaven er beskrevet hvorledes, funktionerne `access_ok`, `copy_to_user` og `copy_from_user` benyttes til selve kopieringsprocessen, da de fysiske adresser er forskellige i userspace og kernelspace.

### 2.3 Fejlhåndtering

Et systemkald skal være robust, så styresystemet ikke bliver ustabilt. Jeg har valgt at implementere det, så det muligvis er mere kantet, men til gengæld meget robust. For at virke, skal den streng man vil gemme have en afslutning, ellers modtages den ikke, og hvis ikke bufferen er stor nok til at få hele beskeden med, accepteres kaldet heller ikke, da der muligvis ville gå information tabt, og der ville under alle omstændigheder mangle en strengafslutning.

## 2.4 Samtidige kald

For at interrupts ikke skal gøre systemet ustabilt i f.eks. et tilfælde hvor systemet er ved at skrive, men bliver stoppet midtvejs, og der så læses og dermed slettes af en anden proces, vil der gå kuk i køen. Dette løses ved at sætte et flag der gør midlertidigt forhindrer interrupts. Dermed kan kaldet implementeres således at det kan færdiggøre den vitale del af arbejdet i én proces, og dermed være stabilt. Selv om to processer samtidigt skulle tilgå køen ved hjælp af de to systemkald, skulle det dermed kunne lade sig gøre at undgå problemer, eftersom den vitale del af kaldet ikke kan interruptes.

# 3 Implementation

## 3.1 Indarbejdning af nyt systemkald

For at få linuxkernen til at acceptere et nyt systemkald, er indsats følgende linier i filen `unistd.h`, som i guiden til at implementere et systemkald[2]:

```
#define __NR_kokkenSend 223  
#define __NR_kokkenHent 251
```

Da der var ledige pladser mellem de allerede tilgængelige systemkald, var det ikke nødvendigt at ændre på andre parametre.

I filen `sys_call_table.c` er indsats følgende linier:

```
extern syscall_handler_t sys_kokkenSend;  
extern syscall_handler_t sys_kokkenHent;  
  
samt  
  
[ __NR_kokkenSend ] (syscall_handler_t *) sys_kokkenSend,  
[ __NR_kokkenHent ] (syscall_handler_t *) sys_kokkenHent,
```

Det er har alle styresystemets systemkald er defineret, således at kernen er klar over hvilke kald der eksisterer.

For at systemet kompilerer implementationen af systemkaldet med, skal `kokkenMSbox.o` tilføjes til filen `Makefile`. Nu er det kun selve implementationen af systemkaldet der mangler.

## 3.2 Systemkaldet

Først oprettes filen `kokkenMSbox.h`, hvor de to kald defineres, og parameterne beskrives. Denne del er triviel, og vedlagt som bilag 6.2

Det er i filen kokkenMSbox.c, hvor den reelle implementation foregår. I denne fil er der to metoder, én til at sende en besked og én til at hente en besked.

Kildekoden i sin helhed er vedlagt som bilag 6.3, men her gennemgåes den i brudstykker

### **kokkenSend**

```
if (length < 1) {
    return -22;
}
```

Først kontrolleres om længde parameteren er længere en bare et afslutningsstegn, ellers er det formålsløst at fortsætte.

```
if(access_ok(VERIFY_READ , buffer , length)) {
```

Vi undersøger om vi har læserettigheder til at kunne kopiere beskeden til kernelspace.

```
local_irq_save(flags);
```

Hvis vi har fået grønt lys med rettighederne, bliver vi nødt til at få kopieret hele strengen uden afbrydelse, ellers ville systemet fejle, hvis systemkaldet blev kaldt midtvejs, og enten overskrive eller hente en halv besked. Derfor forhindres interrupts til kaldet er færdigt med den sårbar del.

```
int i;
for( i=0 ; i < length ; i++) {
    if(buffer[i] == '\0') {
        break;
    }
    if(i == length-1 && buffer[i] != '\0') {
        return -22;
    }
}
```

For at være sikker på at det rent faktisk er en streng der kopieres, undersøges om den indeholder et strengafslutningstegn. Hvis ikke der er et afslutningstegn, opfattes det som forkerte parametre at funktionen er kaldt med (forkert længde/bufferstørrelse), og derved returneres fejlkoden for 'invalid argument'.

```
msg_t* msg = kmalloc(sizeof(msg_t) , GFP_DMA);
if(msg == NULL) {
    return -12;
}
```

Her allokeres pladsen i kernel space til kopieringen

```
msg->next = NULL;
msg->length = length;
msg->message = kmalloc(length,GFP_DMA);
if(msg->message == NULL) {
    return -12;
}
copy_from_user(msg->message, buffer, length);

if (first == NULL) {
    first = msg;
    last = msg;
}
else {
    last->next = msg;
    last = msg;
}
```

Denne del er stort set identisk med koden fra opgaven, med de ændringer der er beskrevet i analyse afsnittet, hvor beskeden kopieres.

```
local_irq_save(flags);
return 0;
}
else {
    return -12;
}
```

I den sidste del frigives låsen på interrupts og der returneres.

### **kokkenHent**

```
int i;
for( i=0 ; i < length ; i++) {
    if(buffer[i] == '\0') {
        break;
}
if(i == length-1 && buffer[i] != '\0') {
    return -22;
}
```

Her testes om vi vil få hele strengen med ud. Hvis ikke den indeholder en strengafslutning gør vi ikke, og noget ville gå tabt.

```
if (first != NULL) {
```

```

local_irq_save(flags);
msg_t* msg = first;
int mlength = msg->length;
first = msg->next;

/* copy message */
if(access_ok(VERIFY_WRITE , buffer , mlength)) {
    copy_to_user(buffer, msg->message, mlength);
    /* free memory */
    kfree(msg->message);
    kfree(msg);
    local_irq_save(flags);
    return mlength;
}
else {
    return -12;
}
}
return -1;
}

```

Igen låses for interrupts når kopieringen sættes igang, beskeden kopiereres til userspace, og hukommelsen frigives.

## 4 Afpøvning

For at teste om systemkaldet fungerer efter hensigten, testes både funktionaliteten som den gerne skulle være (positive test) og test, der forsøger at få systemet til at fejle (negative test).

### 4.1 Positive test

1.
  - Én besked sendes („Test message: Hello World“) med korrekt længde (strlen()+1)
  - Én besked hentes med en buffer der er stor nok til hele beskeden
  - Denne test returnerede som forventet strengen „Test message: Hello World“ og den korrekte retur værdi „26“<sup>1</sup>
2.
  - To beskeder sendes („Test message: Hello World“ og „Another test sample“) begge med korrekt længde (strlen()+1)
  - To beskeder hentes, begge med stor nok buffer til hele beskeden
  - Denne test returnerede korrekt begge strenge, med korrekt længde

---

<sup>1</sup>Test resultaterne er vedlagt, se bilag 6.1

3.
  - Tre beskeder sendes.
  - Tre beskeder hentes, med stor nok buffer.
  - Her returneredes ligeledes alle strenge korrekt.

Da alle tre test forløber efter hensigten, konkluderes det at funktionaliteten er i orden, med hensyn til den korrekte brug. Da operativsystemet ikke må fejle, ved forkert brug af et systemkald, men derimod returnere den korrekte fejlkode, er det vigtigt at teste om dette også sker.

## 4.2 Negative test

1.
  - Her prøves at gemme en streng, men med en negativ længde angivet
  - Testen returnerer `-1`.
  - Ved brug af `perror()` fåes „Invalid argument“, hvilket var forventet.
  - Testen er altså bestået
2.
  - Her testes ved at sende en streng med kortere længde end hele strengen. Herved kopieres slut tegnet `\0` ikke med, og det er ikke en streng længere.
  - Dette kan ikke tillades, og testen fejler korrekt med `'-1'` som er `'Invalid argument'`
3.
  - Her testes ved at sende en streng korrekt, og prøve at hente den igen, men med kortere længde end hele strengen  
Herved mistes noget af strengen og dermed information.
  - Dette kan ikke tillades, og testen fejler korrekt med `'-1'` som er `'Invalid argument'`
4.
  - Sidste test prøver at hente en besked uden at der er nogen i køen.  
Dette skal naturligvis fejle korrekt.
  - Da testen køres lige efter test6, der gemmer en streng korrekt, uden at hente den, returnerer testen i første omgang strengen fra den tidligere test (det er ok).
  - Anden gang testen køres returneres korrekt og returnerer `'-1'` og med `perror()` fåes `'Operation not permitted'`
  - Dette er som forventet, testen er altså bestået

## 5 Konklusion

Der er implementeret to systemkald i en usermode linux kerne, der gør det muligt for message passing mellem to forskellige processer.

Disse systemkald er testet for funktionalitet og robusthed, både i korrekt brug og ved forkerte parametre. Systemkaldene bestod alle tests.

Alle spørgsmål i opgaven er besvaret, og da testene af implementationen ikke afslørede fejl, betragtes opgaven derfor som løst tilfredsstillende.

## Litteratur

- [1] *Silberschatz, Galvin, Gagne, Operating System Concepts, 7th ed.*
- [2] [http://www.imada.sdu.dk/Courses/DM14/add\\_syscall.html](http://www.imada.sdu.dk/Courses/DM14/add_syscall.html)
- [3] <http://hegel.ittc.ku.edu/topics/linux/man-pages/man9/kmalloc.9.html>

## 6 bilag

### 6.1 Testresultater

```
dm14:/home/DM14/kok04/linux-2.6.10# ./test1
Return value of send: 0
First message retrieved: Test message: Hello World
Return value: 26

dm14:/home/DM14/kok04/linux-2.6.10# ./test2
First message retrieved: Test message: Hello World
Return value: 26
Second message retrieved: Another test sample
Return value: 20

dm14:/home/DM14/kok04/linux-2.6.10# ./test3
First message retrieved: Test message: Hello World
Return value: 26
Second message retrieved: Another test sample
Return value: 20
Third message retrieved: This is also a test sample
Return value: 27

dm14:/home/DM14/kok04/linux-2.6.10# ./test4
Return value: -1
Fejlen er: Invalid argument

dm14:/home/DM14/kok04/linux-2.6.10# ./test5
Return value: -1
The error is: Invalid argument

dm14:/home/DM14/kok04/linux-2.6.10# ./test6
Return value: 0
Return value: -1
The error is: Invalid argument

dm14:/home/DM14/kok04/linux-2.6.10# ./test7
Return value: 22
Fejlen er: Success

dm14:/home/DM14/kok04/linux-2.6.10# ./test7
Return value: -1
Fejlen er: Operation not permitted
```

## 6.2 kokkenMSbox.h

```

1  /*
2  * kokkenMSbox.h
3  */
4 #ifndef __UML_KOKKENMSBOX_H__
5 #define __UML_KOKKENMSBOX_H__
6 extern int sys_kokkenHent(char *buffer, int length);
7 extern int sys_kokkenSend(char *buffer, int length);
8 #endif

```

## 6.3 kokkenMSbox.c

```

1  /*
2  * kokkenMSbox.c
3  */
4 #include "linux/kernel.h"
5 #include "linux/unistd.h"
6 asmlinkage
7
8 typedef struct _msg_t msg_t;
9 struct _msg_t{
10     msg_t* next;
11     int length;
12     char* message;
13 };
14
15 static msg_t* first = NULL;
16 static msg_t* last = NULL;
17 unsigned long flags;
18
19 int sys_kokkenSend(char* buffer, int length)
20 {
21     if (length < 1) {
22         return -22;
23     }
24     if(access_ok(VERIFY_READ, buffer, length)) {
25         local_irq_save(flags);
26         int i;
27         for( i=0 ; i < length ; i++) {
28             if(buffer[i] == '\0') {
29                 break;
30             }
31             if(i == length-1 &&
32                 buffer[i] != '\0') {
33                 return -22;
34             }
35         }
36         msg_t* msg =
37             kmalloc(sizeof(msg_t), GFP_DMA);

```

```

36             if (msg == NULL) {
37                     return -12;
38             }
39             msg->next = NULL;
40             msg->length = length ;
41             msg->message =
42                     kmalloc (length , GFP_DMA) ;
43             if (msg->message == NULL) {
44                     return -12;
45             }
46             copy_from_user (msg->message , buffer ,
47                             length );
48
49             if ( first == NULL) {
50                     first = msg;
51                     last = msg;
52             }
53             else {
54                     last->next = msg;
55                     last = msg;
56             }
57             local_irq_save (flags );
58             return 0;
59         }
60     }
61 }
62
63 int sys_kokkenHent (char* buffer , int length )
64 {
65     int i ;
66     for( i=0 ; i < length ; i++) {
67         if (buffer [i] == '\0') {
68             break;
69         }
70         if (i == length-1 && buffer [i] != '\0') {
71             return -22;
72         }
73     }
74
75     if (first != NULL) {
76         msg_t* msg = first ;
77         int mlength = msg->length ;
78         first = msg->next ;
79
80         /* copy message */
81         if (access_ok (VERIFY_WRITE , buffer ,
82                         mlength)) {

```

```

82         local_irq_save(flags);
83         copy_to_user(buffer,
84                         msg->message, mlength);
85         /* free memory */
86         kfree(msg->message);
87         kfree(msg);
88         local_irq_save(flags);
89     }
90     else {
91         return -12;
92     }
93 }
94 return -1;
95 }
```

## 6.4 Test 1

```

1  /*
2  * Test of systemcall kokkenHent og kokkenSend
3  * Send 1, get 1
4  */
5 #include <stdio.h>
6 #include <errno.h>
7 #include "asm/arch/unistd.h"
8
9 _syscall2(int, kokkenSend, char*, buffer, int, length);
10 _syscall2(int, kokkenHent, char*, buffer, int, length);
11
12 int main(int argc, char** argv) {
13
14     char *in = "Test_message:Hello_World";
15     char msg[50];
16     int msglen;
17
18     /* Send a message containing 'in' */
19     msglen = kokkenSend(in, strlen(in)+1);
20     printf("Return_value_of_send: %i\n", msglen);
21     /*char *s ="Fejlen er";
22     perror(s);*/
23
24
25     /* Read a message */
26     msglen = kokkenHent(msg, 50);
27     printf("First_message_retrieved: %s\n", msg);
28     printf("Return_value: %i\n", msglen);
29
30     return 0;
31 }
```

## 6.5 Test 2

```

1  /*
2   * Test of systemcall kokkenHent og kokkenSend
3   * send 2, get 2
4   */
5 #include <stdio.h>
6 #include <errno.h>
7 #include "asm/arch/unistd.h"
8
9 _syscall2(int, kokkenSend, char*, buffer, int, length);
10 _syscall2(int, kokkenHent, char*, buffer, int, length);
11
12 int main(int argc, char** argv) {
13
14     char *in = "Test_message:Hello_World";
15     char *in2 = "Another_test_sample";
16     char msg[50];
17     int msglen;
18
19
20     /* Send a message containing 'in', 'in2' */
21     kokkenSend(in, strlen(in)+1);
22     kokkenSend(in2, strlen(in2)+1);
23
24     /* Read a message */
25     msglen = kokkenHent(msg, 50);
26     printf("First_message_retrieved: %s\n", msg);
27     printf("Return_value: %i\n", msglen);
28     msglen = kokkenHent(msg, 50);
29     printf("Second_message_retrieved: %s\n", msg);
30     printf("Return_value: %i\n", msglen);
31
32     return 0;
33 }
```

## 6.6 Test 3

```

1  /*
2   * Test of systemcall kokkenHent og kokkenSend
3   * send 3, get 3
4   */
5 #include <stdio.h>
6 #include <errno.h>
7 #include "asm/arch/unistd.h"
8
9 _syscall2(int, kokkenSend, char*, buffer, int, length);
10 _syscall2(int, kokkenHent, char*, buffer, int, length);
11
12 int main(int argc, char** argv) {
```

```

13
14 char *in = "Test_message:Hello_World";
15 char *in2 = "Another_test_sample";
16 char *in3 = "This_is_also_a_test_sample";
17 char msg[50];
18 int msglen;
19
20
21 /* Send a message containing 'in', 'in2' and 'in3' */
22 kokkenSend(in, strlen(in)+1);
23 kokkenSend(in2, strlen(in2)+1);
24 kokkenSend(in3, strlen(in3)+1);
25
26 /* Read a message */
27 msglen = kokkenHent(msg, 50);
28 printf("First_message_retrieved: %s\n", msg);
29 printf("Return_value: %i\n", msglen);
30 msglen = kokkenHent(msg, 50);
31 printf("Second_message_retrieved: %s\n", msg);
32 printf("Return_value: %i\n", msglen);
33 msglen = kokkenHent(msg, 50);
34 printf("Third_message_retrieved: %s\n", msg);
35 printf("Return_value: %i\n", msglen);
36
37 return 0;
38 }
```

## 6.7 Test 4

```

1 /*
2  * Test of systemcall kokkenHent og kokkenSend
3  * negative length
4 */
5 #include <stdio.h>
6 #include <errno.h>
7 #include "asm/arch/unistd.h"
8
9 _syscall2(int, kokkenSend , char*, buffer , int , length);
10 _syscall2(int, kokkenHent , char*, buffer , int , length);
11
12 int main(int argc , char** argv) {
13
14 char *in = "Negative_testing:Hello_World";
15 char msg[50];
16 int msglen;
17
18
19 /* Send a message containing 'in', but negative length */
20 msglen= kokkenSend(in, -1);
21 printf("Return_value: %i\n", msglen);
```

```

22 char *s = "Fejlen\_er";
23 perror(s);
24
25
26
27 return 0;
28 }
```

## 6.8 Test 5

```

1  /*
2   * Test of systemcall kokkenHent og kokkenSend
3   * Send buffer too small
4   */
5 #include <stdio.h>
6 #include <errno.h>
7 #include "asm/arch/unistd.h"
8
9 _syscall2(int, kokkenSend, char*, buffer, int, length);
10 _syscall2(int, kokkenHent, char*, buffer, int, length);
11
12 int main(int argc, char** argv) {
13
14     char *in = "Test\_message:\_Hello\_World.\_Length\_is\_"
15     39";
16
17     /* Send a message containing 'in' */
18     int returnValue = kokkenSend(in, strlen(in)-1);
19     printf("Return\_value:\_%i\_\\n", returnValue);
20     char *s ="The\_error\_is";
21     perror(s);
22
23 }
```

## 6.9 Test 6

```

1  /*
2   * Test of systemcall kokkenHent og kokkenSend
3   * recieve buffer too small
4   */
5 #include <stdio.h>
6 #include <errno.h>
7 #include "asm/arch/unistd.h"
8
9 _syscall2(int, kokkenSend, char*, buffer, int, length);
10 _syscall2(int, kokkenHent, char*, buffer, int, length);
11
12 int main(int argc, char** argv) {
13 }
```

```

14     char *in = "Message:HelloWorld\n";
15     char msg[50];
16
17     /* Send a message containing 'in' */
18     int returnValue = kokkenSend(in, strlen(in)+1);
19     printf("Return_value_(Send):%i\n", returnValue);
20
21     /* Read a message */
22     returnValue = kokkenHent(msg, strlen(in)-1);
23     printf("Return_value_(Hent):%i\n", returnValue);
24     char *s ="The_error_is";
25     perror(s);
26
27     return 0;
28 }
```

## 6.10 Test 7

```

1  /*
2   * Test of systemcall kokkenHent og kokkenSend
3   * get without any messages in queue
4   */
5 #include <stdio.h>
6 #include <errno.h>
7 #include "asm/arch/unistd.h"
8
9 _syscall2(int, kokkenSend, char*, buffer, int, length);
10 _syscall2(int, kokkenHent, char*, buffer, int, length);
11
12 int main(int argc, char** argv) {
13
14     char *in = "Test_message:HelloWorld.Length_is_
15         39";
16     char msg[50];
17     int msglen;
18     int returnValue;
19
20     /* Read a message */
21     returnValue = kokkenHent(msg, 50);
22     printf("Return_value:%i\n", returnValue);
23     char *s ="Fejlen_er";
24     perror(s);
25
26     return 0;
27 }
```