Styresystemer og multiprogrammering (OSM) - G1

Dennis Bøgelund Olesen - 060593 - cwb759
Emil Lagoni - 051290 - frs575
Erik David Allin - 171292 - smt504

17. Februar 2013

Task 1

Insert

lol1

Extract

lol2

Search

Task 2

Filer involveret: fs/`io.c`, fs/`io.h` samt tests/`readwrite.c`.

I denne opgave, udnyttede vi os af typen device fra drivers, som tillod os at bruge kernel-kaldene read og write. Vi skulle altså lave en driver pointer.
Da device-strukturen har et generisk device i sin struktur, kan vi udnytte den GCD vi har lavet. Vi kan nemlig se af GCD, at den har henholdsvis read og write, som gør nøjagtigt det vi ønsker.
I forbindelse med dette bruger vi kernel assert til at sikre os, at vi peger på et device.
At `io.c` og `io.h` ligger i mappen fs, er taget fra buenos roadmap, som har inddelt read og write som systemkald, der relaterer til filsystemer. som beskrevet i buenos roadmap, side 44.

Testing

For at teste readwrite, lavede vi filen readwrite.c i mappen tests0.
Efter at have compilet denne, lavede vi:

```
util/tfstool create fyams.harddisk 2048 disk1
```

og

```
util/tfstool write fyams.harddisk tests/readwrite readwrite
```

Når dette er lavet. kan testen køres med kommandoen:
`fyams-sim buenos 'initprog=[disk1]readwrite'`
Når dette er startet er det muligt at taste i terminalen, hvorefter read så læser det du skriver, og write skriver det ud til terminalen igen.
I vores test er der brugt en int buffer. Dette betyder, at alt fylder 4 bytes, så der kan altså ikke læses 63 chars, men derimod kun en fjerdedel.
C kan dog sagtens se chars som integers.

Bilag

dlList.c

```
#include <stdlib.h>
#include <stdint.h>
#include "dlList.h"

void insert(dlist *this, item *thing, bool atTail) {
  node *newNode = malloc(sizeof(node));
  newNode->thing = thing;

  if (atTail) {
    newNode->ptr = (this->tail);
    this->tail->ptr = (node*)((uintptr_t)this->tail->
    ptr ^ (uintptr_t)newNode);
    this->tail = newNode;
  }
  else {
    newNode->ptr = (this->head);
    this->head->ptr = (node*)((uintptr_t)this->head->
    ptr ^ (uintptr_t)newNode);
    this->head = newNode;
  }
```

```
}


item* search(dlist *this, bool (*matches)(item*)) {
  if (matches(this->head->thing))
    return this->head->thing;

  node *prev = this->head;
  node *next = this->head->ptr;

  while ((node*)((uintptr_t)next->ptr ^ (uintptr_t)prev)) {
    if (matches(next->thing))
      return next->thing;
    node *tmp = next;
    next = (node*)((uintptr_t)next->ptr ^ (uintptr_t)prev);
    prev = tmp;
  }
  if (matches(this->tail->thing))
    return next->thing;

  return 0;
}


void reverse(dlist *this) {
  dlist *tmp = this;
  this->head = tmp->tail;
  this->tail = tmp->head;
}

item* extract(dlist *this, bool atTail) {
  item *ext;
  node *address;
```

```
  node *cleanup;
  if (atTail) {
    address = this->tail->ptr;
    ext = this->tail->thing;
    address->ptr = (node*)((uintptr_t)address->
    ptr ^ (uintptr_t) this->tail);
    cleanup = this->tail;
    this->tail = address;
    free(cleanup);
    return ext;
  }
  else {
    address = this->head->ptr;
    ext = this->head->thing;
    address->ptr = (node*)((uintptr_t)address->
    ptr ^ (uintptr_t) this->head);
    cleanup = this->head;
    this->head = address;
    free(cleanup);
    return ext;
  }
}
```

dlList.h

```
#ifndef DL_LIST_H
#define DL_LIST_H

typedef int bool;
typedef void item;

typedef struct node_ {
  item           *thing;
  struct node_   *ptr;
} node;

typedef struct dlist_ {
  node *head, *tail;
} dlist;

/* Inserts an item to either the start or end of the list */
void insert(dlist *this, item* thing, bool atTail);

/* Extracts either the first or last element in the list,
   remove it from the list and returns the item. */
item* extract(dlist *this, bool atTail);

/* Flips the direction of the links */
void reverse(dlist *this);

item* search(dlist *this, bool (*matches)(item*));
#endif // DL_LIST_H
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "dlList.h"


int main() {

  // Tildeler memory/plads til listen samt dens head og tail.
   node *tail = malloc(sizeof(node));
   node *head = malloc(sizeof(node));
   dlist *liste = malloc(sizeof(dlist));

  /* Tildeler memory/plads til de elementer,
     der senere bliver indsat via insert. */
   int *i = malloc(sizeof(int));
   int *j = malloc(sizeof(int));
   int *n = malloc(sizeof(int));
   int *k = malloc(sizeof(int));

  // Vaerdier for elementer der senere bliver indsat.
   i = (int*)1;
   j = (int*)2;
   n = (int*)3;
   k = (int*)4;

  // Tail og head tildeles vaerdier.
   tail->thing = j;
   tail->ptr = head;
```

```
head->thing = n;
head->ptr = tail;

/* Funktioner for at teste om bestemte vaerdier findes i
   dllist via search. */
// Kaster warns
bool *eqone(int a) {
  return (bool*)(a == 1);
}

bool *eqseven(int a) {
  return (bool*)(a == 7);
}

// Vaerdier for at tjekke tiden det tager at indsaette elementer.
clock_t startInsert1, startInsertAll, endInsert1, endInsertAll;


/* Head og tail tildeles deres pladser i listen.
   Der tjekkes herudover, om de har de korrekte vaerdier
   via et print. */
liste->head = head;
liste->tail = tail;
printf("Tail: %p\n", liste->tail);
printf("Head: %p\n", liste->head);


/* Der saettes en clock for at tjekke programmets hidtige koeretid,
   og der bliver indsat en raekke elementer.
   Senere saettes der to "slut" clocks, der senere
   bruges til at tjekke tiden det har taget
   at indsaette elementerne. */
startInsert1 = clock();
```

```
  startInsertAll = clock();
  insert(liste, i, 1);
  endInsert1 = clock();
  insert(liste, i, 1);
  insert(liste, i, 1);
  insert(liste, i, 1);
  insert(liste, i, 1);
  insert(liste, i, 1);
  insert(liste, k, 1);
  insert(liste, k, 1);
  insert(liste, k, 0);
  endInsertAll = clock();


  printf("Insertion time for 1 element: %f\n",
  (double)(endInsert1 - startInsert1) / CLOCKS_PER_SEC);

  printf("Insertion time for alle elementer: %f\n",
  (double)(endInsertAll - startInsertAll) / CLOCKS_PER_SEC);

  printf("%p vaerdi af thing i tail\n",tail->thing);
  printf("%p vaerdi af thing i nye tail\n",liste->tail->thing);

 /* Denne test er ikke korrekt. Slet eller fix
  printf("%p pointer i nye tail\n",liste->tail->ptr);
  printf("%p gamle tail (skal vaere lig pointer i nye tail)\n",tail);
 */

  printf("Tester om 1 er i listen. Returner %p,
  hvilket betyder at den er der.\n",
  search(liste, (item*)eqone));
  printf("Tester om 7 er i listen. Returner %p,
  hvilket betyder at den ikke er det.\n",
```

```
  search(liste, (item*)eqseven));


  // Udkommenter for at teste reverse
  /* Den vil lave print om fra 4,3,2,1 til 1,2,3,4
    reverse(liste);
  */
  printf("\n Foelgende er test for extract\n");
  printf("%p er thing i head (rigtigt hvis = 4)\n",
  liste->head->thing);
  extract(liste,0);
  printf("%p er nu thing i nye head (rigtigt hvis = 3)\n",
  liste->head->thing);
  extract(liste,0);
  printf("%p er nu thing i nye head (rigtigt hvis = 2)\n",
  liste->head->thing);
  extract(liste,0);
  printf("%p er nu thing i nye head (rigtigt hvis = 1)\n",
  liste->head->thing);

  return 0;
}
```

io.h

```
#ifndef IO_H
#define IO_H

int syscall_read(int fhandle, void *buffer, int length);
int syscall_write(int fhandle, const void *buffer, int length);

#endif // IO_H
```

io.c

```
#include "drivers/bootargs.h"
#include "drivers/device.h"
#include "drivers/gcd.h"
#include "drivers/metadev.h"
#include "drivers/polltty.h"
#include "drivers/yams.h"
#include "fs/vfs.h"
#include "kernel/assert.h"
#include "kernel/config.h"
#include "kernel/halt.h"
#include "kernel/idle.h"
#include "kernel/interrupt.h"
#include "kernel/kmalloc.h"
#include "kernel/panic.h"
#include "kernel/scheduler.h"
#include "kernel/synch.h"
#include "kernel/thread.h"
#include "lib/debug.h"
#include "lib/libc.h"
#include "net/network.h"
#include "proc/process.h"
#include "vm/vm.h"


int syscall_read(int fhandle, void *buffer, int length) {
if(fhandle == 0) {
device_t *dev;
gcd_t *gcd;

/* Skaffer device */
dev = device_get(YAMS_TYPECODE_TTY,0);
```

```
KERNEL_ASSERT(dev != NULL);
/* skaffer generisk device fra device */
gcd = (gcd_t *)dev->generic_device;
KERNEL_ASSERT(gcd != NULL);


/* Ifoelge drivers/gcd.h, laeser read, "at most len bytes
from the device to the buffer. the function returns
the number of bytes read." */

return gcd->read(gcd,buffer,length);
}
return -1;

/*int tmp = fhandle;
int tmp2 =(int) buffer;
int tmp3 = length; */
}
int syscall_write(int fhandle, const void *buffer, int length) {
if(fhandle == 1) {
device_t *dev;
gcd_t *gcd;

/* Skaffer device */
dev = device_get(YAMS_TYPECODE_TTY,0);
KERNEL_ASSERT(dev != NULL);
/* skaffer generisk device fra device */
gcd = (gcd_t *)dev->generic_device;
KERNEL_ASSERT(gcd != NULL);


/* Ifoelge drivers/gcd.h, skriver write,
"at most len bytes from the buffer to the device.
```

```
The function returns the number of bytes read." */
return gcd->write(gcd, buffer, length);
}
return -1;
}
```

readwrite.c

```
#include "lib.h"
int main(void)
{
 int a[100];
    syscall_read(0,a,100);
    syscall_write(1,a,100);
 return 0;
}
```