OSM
G assignment 1

Tobias Hallundbæk Petersen (xtv657)
Ola Rønning (vdl761)
Nikolaj Høyer (ctl533)

February 17, 2014

Contents

1 A space-efficient doubly-linked list

```
void main() {
  dlist list = {0};
  int i = 5;
  int i1 = 6;
  int i2 = 7;
  bool b = 0;
  bool b1 = 1;
  insert(&list,&i,b);
  insert(&list,&i1,b);
  insert(&list,&i2,b);
  bool (*match)(item*) = find;
  int* s = search(&list,match);
  printf("The search found:%2d\n",*s);
  int* p = extract(&list,b);
  printf("%d\n",*p);
  int* p1 = extract(&list,b);
  printf("%d\n",*p1);
  int* p2 = extract(&list,b);
  printf("%d\n",*p2);
}
```

1.1 Insert and extract

```
void insert(dlist *this, item *thing, bool atTail) {
  node* n = (node*) malloc(sizeof(node));
  (*n).thing = thing;
  (*n).ptr = n;
  if((*this).head == NULL) {
    (*this).tail = n;
    (*this).head = n;
    return;
  }
  node* dlistNode = (atTail) ? (*this).tail : (*this).head;
  dlistNode -> ptr = (node*) ((long) dlistNode -> ptr ^ (long) n);
  (*n).ptr = dlistNode;
  (atTail) ? ((*this).tail = n) : ((*this).head = n);
}

item* extract(dlist *this, bool atTail) {
  node* returnNode = (*this).head;
  item* value;
  if(returnNode == ((*returnNode).ptr)) {
    (*this).tail = NULL;
    (*this).head = NULL;
    value = (*returnNode).thing;
    free(returnNode);
    return value;
```

```
  }
  if (atTail) {
    returnNode = (*this).tail;
    (*this).tail = (*returnNode).ptr;
    (*this).tail -> ptr = (node*) ((long)(*this).tail -> ptr ^ (long) ↩
        returnNode);
    value = (*returnNode).thing;
    free(returnNode);
    return value;
  }
  (*this).head = (*returnNode).ptr;
  (*this).head -> ptr = (node*) ((long)(*this).head -> ptr ^ (long) returnNode↩
      );
  value = (*returnNode).thing;
  free(returnNode);
  return value;
}
```

1.2 Search

```
item* search(dlist *this, bool (*matches)(item*)){
  node* itNode = this -> head;
  node* oldPtr = (node*) 0;
  node* tempPtr;
  if (matches(itNode -> thing)) {
    return itNode -> thing;
  }
  while (itNode != this -> tail) {
    tempPtr = itNode;
    itNode = (node*) ((long) itNode -> ptr ^ (long) oldPtr);
    oldPtr = tempPtr;
    if (matches(itNode -> thing)) {
      return itNode -> thing;
    }
  }
  return NULL;
}

bool find(item* thing) {
  return (*((int*)thing) == 5);
}
```

1.3 Reverse

```
void reverse (dlist *this) {
  void* temp = ((*this).head);
  (*this).head = (*this).tail;
```

```
  (*this).tail = temp;
}
```

2 Buenos system calls for basic I/O

2.1 Implement system calls `read` and `write`

We start by adding the appropriate system call cases to `proc/syscall.c`. In both `SYSCALL_READ` and `SYSCALL_WRITE` we call the appropriate functions defined in `kernel/read.h` and `kernel/write.h`. The parameters are extracted from `MIPS` registers `A1`, `A2` and `A3` and the result is placed in the return register `V0`.

```
...
    switch(user_context->cpu_regs[MIPS_REGISTER_A0]) {
    case SYSCALL_HALT:
        halt_kernel();
        break;
    case SYSCALL_READ:
        user_context->cpu_regs[MIPS_REGISTER_V0] =
            syscall_read(
              (int) user_context->cpu_regs[MIPS_REGISTER_A1],
              (void*) user_context->cpu_regs[MIPS_REGISTER_A2],
              (int) user_context->cpu_regs[MIPS_REGISTER_A3]
            );
        break;
    case SYSCALL_WRITE:
        user_context->cpu_regs[MIPS_REGISTER_V0] =
            syscall_write(
              (int) user_context->cpu_regs[MIPS_REGISTER_A1],
              (const void*) user_context->cpu_regs[MIPS_REGISTER_A2],
              (int) user_context->cpu_regs[MIPS_REGISTER_A3]
            );
        break;
    default:
        KERNEL_PANIC("Unhandled system call\n");
    }
...
```

We now implement the functions defined in `kernel/read.h` and `kernel/write.h` in files `kernel/read.c` and `kernel/write.c`

2.1.1 `kernel/read.c`

The `read` function simply gets the generic `TTY` device driver and reads input into the `buffer`. We also make sure that our `fhandle` is 0 as we expect it would be

in the read call - otherwise 0 characters are read and 0 returned.

```
#include "kernel/read.h"
#include "drivers/gcd.h"
#include "drivers/device.h"

int read_kernel(int fhandle, void *buffer, int length){
  int len;
  if (fhandle != 0) {
    return 0;
  }
  device_t *dev;
  gcd_t *gcd;
  dev = device_get(YAMS_TYPECODE_TTY, 0 );
  gcd = (gcd_t *)dev->generic_device;
  len = gcd->read(gcd, buffer, length);
  return len;
}
```

2.1.2 `kernel/write.c`

The `write` function is very similar, except that it writes using the generic device driver instead of reading.

```
#include "kernel/write.h"
#include "drivers/gcd.h"
#include "drivers/device.h"

int write_kernel(int fhandle,const void *buffer, int length){
  int len;
  if (fhandle != 1) {
    return 0;
  }
  device_t *dev;
  gcd_t *gcd;
  dev = device_get(YAMS_TYPECODE_TTY, 0 );
  gcd = (gcd_t *)dev->generic_device;
  len = gcd->write(gcd, buffer, length);
  return len;
}
```

2.2 `tests/readwrite.c`

Testing blabla - Tobias skriv noget her :)

```
#include "tests/lib.h"
```

```
int main(void)
{
  char buffer[1];
  char newline[1] = {'\n'};
  char larger[12] = {'k',' ','i','s',' ','l','a','r','g','e','r','\n'};
  char smaller[13] = {'k',' ','i','s',' ','s','m','a','l','l','e','r','\n'};
  char correct[8] = {'c','o','r','r','e','c','t','\n'};
  int l, len;
  int k = () % 100;
  while (1) {
    syscall_read(0,buffer,1);
    syscall_write(1,buffer,1);
    l = (buffer[0] - '0')*10;
    syscall_read(0,buffer,1);
    syscall_write(1,buffer,1);
    syscall_write(1,newline,1);
    l += (buffer[0] - '0');
    if (l < k) {
      len = 12;
      syscall_write(1, larger, len);
    } else if (l > k) {
      len = 13;
      syscall_write(1, smaller, len);
    } else {
      len = 8;
      syscall_write(1, correct, len);
      break;
    }
  }
  syscall_halt();
  return 0;
}
```