

IT UNIVERSITY OF COPENHAGEN

OPERATIVSYSTEMER OG C

BOSC

Obligatorisk Opgave 3

Author:

Omar KHAN (omsh@itu.dk)
Mads LJUNGBERG (malj@itu.dk)

November 18, 2015

Contents

1	Introduktion	2
2	Teori	3
2.1	Page Faults	3
2.2	Demand Paging	3
2.3	Udskiftning af sider	3
2.3.1	Tilfældig udskiftning	3
2.3.2	FIFO udskiftning	3
2.3.3	Custom udskiftning	3
3	Implementation	3
3.1	Page Fault Håndtering	3
3.2	Udskiftning af sider	3
3.2.1	Tilfældig udskiftning	3
3.2.2	FIFO udskiftning	3
3.2.3	Custom udskiftning	3
4	Testing	3
5	Reflektion	3
6	Konklusion	3
7	Appendix A - Sourcecode	3

1 Introduktion

Hukommelse er en vigtig del af et operativ system, da programmer skal indlæses i hukommelsen for at kunne køre. I moderne operative systemer er der typisk to former for hukommelse, nemlig den fysiske og den virtuelle hukommelse. Den fysiske hukommelse er det vi kender som RAM(Random Access Memory) og det er i denne hukommelse et program skal indlæses før kørsel. Virtuel hukommelse er derimod en proces der står for at udskifte data mellem den fysiske hukommelse og lagerenheden.

I denne rapport fokuseres der på teorien bag virtuel hukommelse, særligt omkring udskiftning af data mellem fysisk hukommelse og lager, samt hvordan det kan implementeres i et operativ system.

2 Teori

2.1 Page Faults

2.2 Demand Paging

2.3 Udskiftning af sider

2.3.1 Tilfældig udskiftning

2.3.2 FIFO udskiftning

2.3.3 Custom udskiftning

3 Implementation

3.1 Page Fault Håndtering

3.2 Udskiftning af sider

3.2.1 Tilfældig udskiftning

3.2.2 FIFO udskiftning

3.2.3 Custom udskiftning

4 Testing

5 Reflektion

6 Konklusion

7 Appendix A - Sourcecode

```
1 /*
2 Main program for the virtual memory project.
3 Make all of your modifications to this file.
4 You may add or rearrange any code or data as you need.
5 The header files page_table.h and disk.h explain
6 how to use the page table and disk interfaces.
```

```

7  */
8
9  #include "page_table.h"
10 #include "disk.h"
11 #include "program.h"
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16 #include <errno.h>
17
18 char *physmem;
19
20 struct disk *disk;
21 int npages, nframes;
22 int *loaded_pages;
23 int pageswap, fifo_counter, fault_counter = 0;
24
25 void get_swap_frame(int *vFrame)
26 {
27     switch(pageswap)
28     {
29         case 0:
30             *vFrame = lrand48() % nframes;
31             return;
32         case 1:
33             *vFrame = fifo_counter;
34             fifo_counter++;
35             fifo_counter = fifo_counter % nframes;
36             return;
37         case 2:
38             //TODO: MAKE A FASTER ALGORITHM(CHECK NFU)
39             return;
40     }
41 }
42
43 void page_fault_handler( struct page_table *pt, int page )
44 {
45     fault_counter++;
46     int flag;
47     int frame;
48
49     printf("Page fault occurred\n");
50     //get frame and flag for the page
51     page_table_get_entry(pt, page, &frame, &flag);
52     page_table_print_entry(pt, page);

```

```

53 int i;
54 switch(flag)
55 {
56     case 0:
57         printf("IN 0\n");
58         //check for free frame
59         for(i = 0; i < nframes; i++)
60         {
61             if(loaded_pages[i] == -1)
62             {
63                 //read from disk to physmem
64                 page_table_set_entry(pt, page, i, PROT_READ);
65                 disk_read(disk, page, &physmem[i*PAGE_SIZE]);
66                 page_table_print_entry(pt, page);
67                 printf("\n");
68                 loaded_pages[i] = page;
69                 return;
70             }
71         }
72         printf("SIDESWAPPING\n");
73         //variables for victim
74         int vFrame, vPage, vFlag;
75         //get the victim frame
76         get_swap_frame(&vFrame);
77         //set the victim page
78         vPage = loaded_pages[vFrame];
79         //get the victim flag
80         page_table_get_entry(pt, vPage, &vFrame, &vFlag);
81         //check for RW flag
82         if(vFlag == (PROT_READ|PROT_WRITE))
83         {
84             //write victim from physmem to disk
85             disk_write(disk, vPage, &physmem[vFrame*PAGE_SIZE]);
86         }
87         //read from disk to victim frame
88         disk_read(disk, page, &physmem[vFrame*PAGE_SIZE]);
89         //update page table entries
90         page_table_set_entry(pt, page, vFrame, PROT_READ);
91         page_table_set_entry(pt, vPage, 0, 0);
92         page_table_print_entry(pt, page);
93         printf("\n");
94         //update loaded_pages
95         loaded_pages[vFrame] = page;
96         return;
97
98     case PROT_READ:

```

```

99     printf("IN READ\n");
100     page_table_set_entry(pt, page, frame, PROT_READ|PROT_WRITE);
101     page_table_print_entry(pt, page);
102     printf("\n");
103     return;
104 }
105 printf("page fault on page %d\n",page);
106 exit(1);
107 }
108
109 int main( int argc, char *argv[] )
110 {
111     if(argc!=5)
112     {
113         printf("use: virtmem <npages> <nframes> <rand|fifo|custom>
114             <sort|scan|focus>\n");
115         return 1;
116     }
117     npages = atoi(argv[1]);
118     nframes = atoi(argv[2]);
119     const char *algorithm = argv[3];
120     const char *program = argv[4];
121
122     loaded_pages = malloc(sizeof(int) * nframes);
123     int i;
124     for(i = 0; i < nframes; i++)
125     {
126         //indicate that there is no pages loaded yet
127         loaded_pages[i] = -1;
128     }
129
130     disk = disk_open("myvirtualdisk",npages);
131     if(!disk)
132     {
133         fprintf(stderr,"couldn't create virtual disk:
134             %s\n",strerror(errno));
135         return 1;
136     }
137
138     struct page_table *pt = page_table_create( npages, nframes,
139         page_fault_handler );
140     if(!pt)
141     {
142         fprintf(stderr,"couldn't create page table: %s\n",strerror(errno));
143         return 1;

```

```

142     }
143
144     char *virtmem = page_table_get_virtmem(pt);
145
146     physmem = page_table_get_physmem(pt);
147
148     if(!strcmp(algorithm, "rand"))
149     {
150         pageswap = 0;
151     }
152     else if(!strcmp(algorithm, "fifo"))
153     {
154         pageswap = 1;
155         fifo_counter = 0;
156     }
157     else if(!strcmp(algorithm, "custom"))
158     {
159         pageswap = 2;
160     }
161     else
162     {
163         fprintf(stderr, "unknown algorithm: %s\n", argv[2]);
164     }
165
166     if(!strcmp(program, "sort"))
167     {
168         sort_program(virtmem, npages*PAGE_SIZE);
169     }
170
171     else if(!strcmp(program, "scan"))
172     {
173         scan_program(virtmem, npages*PAGE_SIZE);
174     }
175
176     else if(!strcmp(program, "focus"))
177     {
178         focus_program(virtmem, npages*PAGE_SIZE);
179     }
180
181     else
182     {
183         fprintf(stderr, "unknown program: %s\n", argv[3]);
184     }
185     printf("Faults: %d\n", fault_counter);
186     page_table_delete(pt);
187     disk_close(disk);

```



```
188  
189     return 0;  
190 }
```