Operating Systems
Constructor University
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2023-10-27
Due: 2023-11-03

Problem Sheet #8

Problem 8.1: *address spaces in a paging system* (1+1+1 = 3 points)

Consider a operating system that uses paging for memory management with a page size of 2048 B. The logical address space of processes is limited to a maximum of 16 pages. The physical memory has a size of 256 KiB.

a) How many frames has the physical memory?

b) How many bits has an address in the logical address space and how many bits has an address in the physical address space?

c) How many bits are used for the page number and how many bits are used for the offset within a page?

Problem 8.2: *paging and page tables* (1+1+1 = 3 points)

Consider a tiny computer system with a physical memory space of 16 frames. Each 12-bit logical address uses 4 bits for the page number and 8 bits for the offset within a page. There are two processes P_1 and P_2 with the logical address spaces shown below.

Process P_1		
Page	Logical Addresses	Segment
$p_{1,0}$	0x000-0x0FF	text
$p_{1,1}$	0x100-0x1FF	text
$p_{1,2}$	0x200-0x2FF	data
$p_{1,5}$	0x500-0x5FF	heap
$p_{1,6}$	0x600-0x6FF	stack
$p_{1,8}$	0x800-0x8FF	stack

Process P_2		
Page	Logical Addresses	Segment
$p_{2,0}$	0x000-0x0FF	text
$p_{2,1}$	0x100-0x1FF	text
$p_{2,4}$	0x400-0x4FF	data
$p_{2,5}$	0x500-0x5FF	data
$p_{2,6}$	0x600-0x6FF	heap
$p_{2,8}$	0x800-0x8FF	stack

Some pages reside in physical memory as shown in the table below. The notation $p_{i,n}$ refers to page n of the logical address space of process P_i. The OS pages are used by the operating system, unused frames are marked with a dash.

Frame	Physical Addresses	Loaded Page
0	0x000-0x0FF	OS
1	0x100-0x1FF	$p_{2,5}$
2	0x200-0x2FF	-
3	0x300-0x3FF	$p_{1,8}$
4	0x400-0x4FF	$p_{2,4}$
5	0x500-0x5FF	-
6	0x600-0x6FF	$p_{1,1}$
7	0x700-0x7FF	-
8	0x800-0x8FF	$p_{1,0}$
9	0x900-0x9FF	$p_{2,0}$
10	0xA00-0xAFF	-
11	0xB00-0xBFF	$p_{1,6}$
12	0xC00-0xCFF	$p_{2,1}$
13	0xD00-0xDFF	-
14	0xE00-0xEFF	-
15	0xF00-0xFFF	-

a) Write down the page tables for both processes P_1 and P_2. Each page table entry maintains the following additional bits: r = read access, w = write access, x = execute access, d = dirty, v = valid, commonly written in the form `rwxdv` if all bits are set or as `rw--v` if only the r, w, and v bits are set. Assume that all writable pages are dirty.

b) The CPU executes process P_1 and the machine instructions modify a global variable and a dynamically allocated string. A context switch occurs and the CPU executes process P_2, which performs a function call that allocates and initializes 16 bytes on the heap.

   Write down the content of the page tables after all write operations and initializations have been performed. If a page fault occurs use the first free physical frame to load the page.

c) The processes P_1 and P_2 establish a shared memory page to exchange data. The shared page appears as $p_{1,4}$ in the logical address space of P_1 and as $p_{2,2}$ in the logical address space of P_2. Show the resulting use of the physical memory frames (i.e., update the table shown above).

Problem 8.3: *memory mapped word count* (4 points)

Write a program `mwc` that prints the number of lines, words, and bytes contained in each input file mention in the command arguments, or standard input (if no file is specified). A word is a non-zero-length sequence of printable characters delimited by white space (use `isspace()` defined in `ctype.h`). Your program should use memory mapping for regular files and it should fallback to regular I/O for all other files or data received via the standard input. Test whether you program is running faster than the `wc` program installed on your system.

The original `wc` also prints a summary line if multiple files are listed on the command line. This is not required to implement, but you may choose to do so in order to stay aligned with the original `wc` program.

Here are some example executions:

```
$ ./build/mwc ./mwc.c
     175     477    3478 ./mwc.c
$ ./build/mwc < ./mwc.c
     175     477    3478
$ ./build/mwc < /dev/null
       0       0       0
$ ./build/mwc ./mwc.c ./mwc.c
     175     477    3478 ./mwc.c
     175     477    3478 ./mwc.c
     350     954    6956 total
```