Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2022-10-27
Due: 2022-11-03

OS 2022 Problem Sheet #8

Problem 8.1: *page tables* (2+1+2 = 5 points)

Consider a paging system supporting 16-bit address spaces. The memory system operates on bytes (the smallest addressable unit is a byte) and the page size of 4096 bytes. A concrete device has 32 kibi bytes of physical memory. The first memory frame is reserved for the operating system. We denote the memory frames with $M.i$ where i is the frame number. Furthermore, we denote the pages of process P with $P.i$, where i is the page number.

a) For process A, we know that page 7 of the text segment maps to frame 5, page 8 of the stack segment maps to frame 1, and page 9 of the initialized data segment holding constants maps to page 3. For process B, we know that page 11 of the stack segment maps to frame 4, page 12 of the text segment makes to frame 5, and page 13 of the initialized data segment holding constants maps to page 3. Draw the page tables for the processes A and B. Every entry should indicate whether it is valid and if valid, the page access rights.

b) Draw a table showing what is currently stored in physical memory. Every entry should indicate the frame number, the start address of the frame, and how the frame is currently used.

c) Process A accesses page 6, which belongs to A's heap. Shortly afterwards, process B accesses page 6, which belongs to B's heap. The kernel allocates frames in increasing frame number order. Draw the updated page tables and the update table showing the usage of the physical memory.

Problem 8.2: *working set vs least recently used* (2 points)

Consider the design of a paging system that tracks the working set of all processes and aims at keeping the working sets of the processes resident. If pages are removed from the working set of a process, then these pages are removed from the resident set of pages and the frames get marked as unused. In other words, only pages that are part of the working set are resident, all other pages are not.

Assume the working set is calculated over the last N memory accesses for every process. Compare this system with a system that uses the least recently used (LRU) page replacement strategy for each process with N pages allocated to each process. What can you say about the pages that are resident in both systems at any point in time?

a) Which system do you think is preferable? Explain.

b) Can you suggest further improvements?

Problem 8.3: *memory chat* (3 points)

Write a program `mchat` that uses shared memory maps to exchange messages. The program, if called without any arguments, attaches to a shared memory map and shows the content (which has to be a nul-terminated string) on the standard output. If called with arguments, then the arguments are written to the top of the shared memory by moving the existing content downwards in order to make space available at the beginning and then filling the created space with the new content. To properly separate the messages, the program should add a newline at the end of the message and prefix the message with a string identifying the process adding a message.

Your program should support the following options:

- The option `-z` clears the text maintained in the shared memory segment.

- The option `-c ctrl` set the control file name used to attach to the memory segment. The default control file name is `/tmp/mchat.ctrl`.

Your program should only use memory mappings to update the messages, do not read from or write to the attached file. For this assignment, you are allowed to ignore any possible race conditions since the focus of this assignment is on memory mappings.

Here is an example execution trace:

```
$ ./mchat -z
$ ./mchat hello
$ ./mchat
[292982]: hello
$ ./mchat hello world
$ ./mchat
[292986]: hello world
[292982]: hello
$ ./mchat
[292986]: hello world
[292982]: hello
$ ./mchat -z hello people
$ ./mchat
[293007]: hello people
```