Operating Systems – spring 2022
Assignment 5

Instructor: Hans P. Reiser

Submission Deadline: Monday, February 21st, 2022 – 23:59

A new assignment will be published every week, right after (or a bit before) the last one was due. It must be completed before its submission deadline. (Hard deadlines, no extension.)

T-Questions are theory homework assignments. **The answers to the assignments must be uploaded to Canvas (in the quiz).**

P-Questions are programming assignments. Download the provided template from Canvas. Do not fiddle with the compiler flags. Submission instructions can also be found in first assignment.

The topic of this assignment is paging-based address translation and TLBs.

T-Question 5.1: Paging

Consider a system that translates virtual addresses to physical addresses using hierarchical page tables. Every page table page comprises 512 entries, with each entry having a size of 8 bytes. The size of both the virtual and the physical address spaces is 512 GiB. The page size is 4096 bytes.

a. How many levels L do we need in a hierarchical (multi-level) page table that can map the complete virtual address space to physical addresses? At each level, what is the maximum of page table pages that we can have? **2 T-pt**
Note: Level 1 designates the lowest level, Level L the highest level. For example, standard 32 bit Intel x86 has one page at L2 (=Page Directory) and (up to) 1024 pages at L1 (=Page Table)

b. Into what parts would a MMU for the system above split the virtual address during address translation? For each part give its length in bits. **2 T-pt**

c. Explain how the number of processes in a system correlates with the amount of memory needed for page tables, if the system uses inverted page tables. **1 T-pt**

d. Explain how the number of processes in a system correlates with the amount of memory needed for page tables, if the system uses hierarchical four-level pages tables (for example, such as you can find on 64 bit Intel x86). **1 T-pt**

e. Modern Intel x86 CPUs support page tables with huge pages of 1 GiB size. What is the main benefit of using such huge pages? What would be a main disadvantages if a Linux system used these huge pages as the default page size for all applications? **2 T-pt**

f. Assume you have a system with 4-level page tables. A TLB lookup takes 1 ns, a memory access 100 ns (no other caches considered here). Calculate the average (effective) memory access time of the access to the data array in the inner loop. **2 T-pt**
(You may assume that initially the TLB is empty, and that no other activities concurrent to the execution of loop() take place.)

```
uint32_t data[1024];
uint32_t sum=0;

void loop() {
        for(int i=0; i<1000; i++) {
                for(int j=0; j<1000; j++) {
                        sum += data[j];
                }
        }
}
```

P-Question 5.1: TLB and Paging Measurements

Download the template **p1** for this assignment from Canvas. You may only modify and upload the file `measure_tlb.c`, together with a file `results.png` (see part (f)).

In this question you will implement code to measure the impact of paging and TLB misses on memory access performance.

a. Write a function that uses `gettimeofday()` (see manpage) to create a timestamp in microseconds. **1 P-pt**

```
uint64_t getTimeStamp();
```

b. Write a function that measures and returns the execution time (in microseconds) of another function, using your function from part (a). This other function takes a generic `void *` as a parameter. **1 P-pt**

```
uint64_t measureFunction( void(*function)(void *), void *arg );
```

c. Measurements of timing depend on many nondeterminstic factors, such as other users/proceses on the system. We thus want to repeat a measurement multiple times and generate statistics (calculate the minimum, average, and maximum value). Implement a function that invokes `measureFunction` *statcount* times and returns (call-by-reference!) the statistics it collected. **2 P-pt**

```
void measureStatistics( int statcount, Statistics *stat,
   void(*function)(void *), void *arg );
```

d. Implement a function that takes two parameters n and $pages$. The function shall make n memory accesses on $pages$ different memory pages (each memory access shall access a different memory page, until *pages* different pages have been accessed, and then start again at the first page.) Your function first has to `malloc()` to allocate space on the heap that is large enough to contain the required number of pages. In case of insufficient available memory, your application shall print an error message and exit with status code 1. **3 P-pt**

```
void accessMemory(int n, int pages);
```

e. The `accessMemory` function takes two paramters, while the generic measuremenmt function passes a single `void *` to the function it invokes. Implement a simple wraper function that receives a pointer to a `MeasureParameters` struct (as the generic `void *`) and then invokes your `accessMemory` function **1 P-pt**

```
void accessMemoryWrapper(MeasurementParamters *param);
```

f. Adjust the parameter ITERATIONS such that the measurements for each page number take around one second for small page numbers. Run the measurement and use a tool of your choice (Excel, gnuplot, R, etc.) to create a plot (X-axis: number of pages; Y-axis: measured time). The caption shall state a system description (uname -a) and CPU (see model name in /proc/cpuinfo), or simply "'skel" if using skel. Add this graph as a picture in PNG format (results.png) to your submission. **2 P-pt**

Total: 10 T-pt 10 P-pt