Computer Science Practice and Experience: Operating Systems (Comp Sci 3SH3), Term 2, Winter 2018
Dr. Neerja Mhaskar

Assignment-I [40 points]

Due at 11:59pm on March 17th, 2018

- **No late assignments accepted.**
- Make sure to submit a version of your assignment ahead of time to avoid last minute uploading issues.
- Note that groups copying each other's solution will get a zero, and reported to the department.
- Only one copy of your assignment should be submitted.
- In your C programs, you should follow good programming style, which includes providing instructive comments and well-indented code.
- You submission should include a readme file. Your readme file should contain the names, studentIDs and MACIDs of each student in the group. It should also clearly state the work done by each student in the group.
- You need to ensure that all your solutions work correctly on the Linux virtual machine.

Memory Management in OS

In this assignment you will write a simple **memory management simulator in C** that supports paging. In this setup the logical address space (2^{16} = 65,536 bytes) is **larger than** the physical address space (2^{15} bytes), and the size of each page size is 256 bytes.

Your simulator needs to accomplish the following:

1. Simulate a **memory management unit (MMU)**, which translates logical addresses to physical addresses
    a. Translating a logical address to physical address in this setup will involve the following: Check TLB for the page. If page not found in the TLB, check the page table if the page exists in memory. If it does not then a page fault occurs.
    b. Handling page faults involves copy the page from the backing store to memory. Since logical address space is **larger than** physical address space, a page request might involve replacing a page in memory with the new page. The page replacement policy to be used is FIFO page replacement policy.

You assignment is divided into the following three parts:

MMU – Address Translation [10 marks]

Your program will translate logical to physical addresses using a page table as outlined in the lecture notes under paging.

1. To simulate a program's memory address requests we use the text file 'addresses.txt'. This file can be downloaded from Avenue -> content -> Assignments. This file contains integer values representing logical addresses ranging from 0 − 65535 (the size of the logical address space).
2. Your program will open this file using the fopen() function and read each logical address from the file and compute its page number and offset **using bitwise operators in C**. See slides on bitwise operators in C under lecture notes on Main Memory.
3. After extracting the page number from the logical address your program will look up the TLB. In the case of a TLB-hit (an entry corresponding to a page number exists), the frame number is obtained from the TLB. In the case of a TLB-miss (**NO** entry corresponding to the page number exists), look up the page table. In the latter case either the frame number is obtained from the page table or a page fault occurs.
4. The **page table** can simply be an array. Since the system implements demand paging, all the entries of the page table can be set to -1 to indicate a page is not in memory.

   Refer to lecture notes on Main memory and Lab8a8b9a material to work on this part of the assignment.

MMU - TLB [10 marks]

1. To simulate entries of TLB in your program you could create a data structure called TLBentry which stores page number and frame number pair.
2. You will need three functions related to the TLB:
   a. A `search_TLB` function - To search the TLB for an entry corresponding to a page number.
   b. A `TLB_Add` function – To add an entry to the TLB. Since TLB use the FIFO policy, if TLB is full any entry added to the TLB replaces the oldest entry.
   c. A `TLB_Update` function – To update the TLB when a page 'p' is replaced in physical memory, and an entry corresponding to page 'p' exists in the TLB. Normally this would require removing this page's entry from the TLB, shuffling the TLB contents to maintain the FIFO order, and adding the new page entry at the top of the TLB. However, **to simply the assignment**

you can handle the update function by simply add the new page entry at the same location as that of the page p.

3. All of the above mentioned functionality can be achieved by simulating the TLB as a **circular array**.

Note: It is recommended that you bypass the TLB and use only a page table initially, and once you have your program working correctly integrate the TLB. Note that, address translation can work without a TLB; the TLB just makes it faster.

Handling Page Faults [20 marks]

Handling page faults involves copying the page from the backing store to a frame in memory.

1. The backing store is represented by the file BACKING_STORE.bin. It is a binary file of size 65,536 bytes. This file can be downloaded from Avenue -> content -> Assignments.
2. Open this file using the open() system call in your program and map it to a memory region using the system call mmap().
3. When page fault occurs, you will read in a 256-byte page from this memory mapped file and copy it in an available frame in physical memory using the memcpy() function.
4. Since logical address space is **larger than** the physical address space, a page request might involve replacing a page in memory with the new page. The page replacement policy to be used is FIFO page replacement policy; that is the oldest page in memory is replaced by the new page**. Note:** when you replace a page in memory you will need to update two entries of the page table; one corresponding to the page being replaced, in which case the entry is -1, and the other corresponding to the new page that is brought into memory.
5. The easiest way to accomplish the FIFO page replacement policy is to **simulate the physical memory as a circular array**.

Example of page fault is as follows: if a logical address with page number, for example 15 resulted in a page fault, your program would read in page 15 from BACKING STORE (remember that pages begin at 0 and are 256 bytes in size) and store it in an empty frame in physical memory. If an empty frame does not exist then you need to replace the oldest page. Suppose page 2 was the oldest page in memory. In this case you will bring in page 15 and store it in the frame occupied by page 2. You will need to update the page table to reflect this change. Any subsequent accesses to page 15 will be resolved by the page table.

Refer to lecture notes on Memory Mapped files under lecture notes on Virtual Memory and Lab10a10b material to work on this part of the assignment.

Program Output

Your program is to output the following values:

a. Logical address being translated (that is the integer value being read from addresses.txt).
b. The corresponding physical address.
c. The signed byte value stored at the corresponding physical address.
d. Compute and output Statistics: In addition to the above output your program is to report the following statistics:
    i. Total number of page-faults — the number of address references that resulted in page faults.
    ii. Total number of TLB hits —the number of address references that were resolved in the TLB.

These statistics can be computed by simply maintaining integer variables to keep track of the page faults and TLB hits.

Program output: ./assignment2
****
Sample output posted on Avenue -> Content -> Assignments (output.txt file).

Deliverables:

1. **assignment2.c** - You are to provide your solution to this assignment as a single C program named assignment2.c.