Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Operating Systems and Networks	Spring 2016
Assignment 3	Due 22 March 2016

1. **Memory Management**

   (a) **Introduction**

   What are the goals of memory managent in a modern OS?

   (b) **Segmentation**

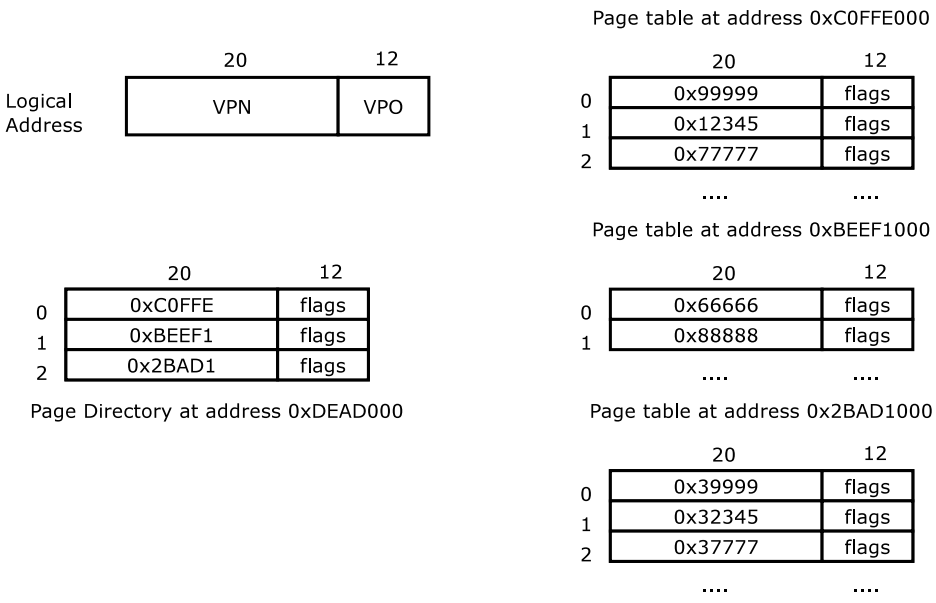
   Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

   What are the physical addresses for the following logical addresses given as (segment, offset) tuples?

   i. 0, 430
   ii. 1, 10
   iii. 2, 500
   iv. 3, 400
   v. 4, 112

   (c) **Paging**



   Answer the following questions concerning the given P6 page table:

   i. How does paging provide isolation of processes?
   ii. How does paging allow multiple processes to share a memory region?
   iii. Which physical address is referenced by the virtual address 0x00802BAD?
   iv. Which virtual address references the physical address 0x77777777?
   v. Only the 20 most significant bits of a page directory entry are used to reference the location of a page table, the remaining 12 bits are used for flags. What does this imply for the location of page tables?

vi. What does the kernel have to do so that different processes use different page tables?

vii. If a memory reference takes 100 nanoseconds, how long does a paged memory reference take if there is no TLB or cache?

(d) **Virtual Memory**

Consider a paged virtual address space composed of 1024 pages of 4 KB each, which is mapped into a 1 MB physical memory space. What is the format of the logical address; i.e., which bits are the offset bits and which are the page number bits? Explain.

(e) **Page Replacement**

Consider the following page access pattern:

```
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6
```

How many page faults would occur assuming one, two, three, four, five, six, or seven frames for the following replacement algorithms?

i. LRU replacement

ii. FIFO replacement

iii. Optimal replacement

2. **Signals**

As explained in the lecutre, processes cannot only communicate with each other using pipes (which we covered in the last assignment) but also by sending signal to another process. In this exercise your task is fork a child process and then let the parent process interact with its child by sending signals to it. Here is what your program should do:

1. From the parent process, fork a child process
2. The parent process interacts with the child by sending signal to it. The parent actions are listed in the following listing.
   (a) Sleep for two seconds.
   (b) Pause the child.
   (c) Sleep for two seconds.
   (d) Continue the child.
   (e) Sleep for two seconds.
   (f) Send the SIGTERM signal to the child.
   (g) Sleep for two seconds.
   (h) Send the SIGTERM signal to the child again.
   (i) Return the main function.
3. The child process continously increments a counter by one and prints its value before going to sleep for one second. When the child recieves the termination signal for the first time, it ignores it. When the child recieves the terminal signal for the second time, the child performs the default action and terminates the process.

(Hint: Lookup the definition for the `kill` and `signal` command from the man page.)

3. **Synchronisation**

In this exercise you will implement a simple lock, that lets you synchronise two parallel running threads. For this, create two threads that both increment a global counter variable `10e6` times. After you joined the threads, print out the value of the global counter variable. For the creation and handling of the threads have a look at the `pthreads` library.

(a) What value do you expect the global counter variable to have after the threads have joined? What value do you observe?

(b) Next, improve your program by implementing a TestAndSet (TAS) lock. Recall that the idea of TAS is to test the value of the lock variable to be zero and in that case exchange it by one. If a thread managed to get hold of the zero value from the lock variable, it is allowed to enter the critical section, otherwise the thread tests the value of the lock variable again. When the thread leavs the critical region it unlocks the lock by resetting the lock variable to zero again. Implement the testing on the lock variable and value exchange using the `xchg` assembly instruction.

(c) What is now the output of your program?