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?

   Solution:
   - Provide memory to applications as a contiguous address space.
   - Protect the memory of one process from access by other processes, but allow sharing memory under some circumstances.
   - Provide more virtual memory than physical memory available.

   (b) **Segmentation**

   Consider the following segment table:

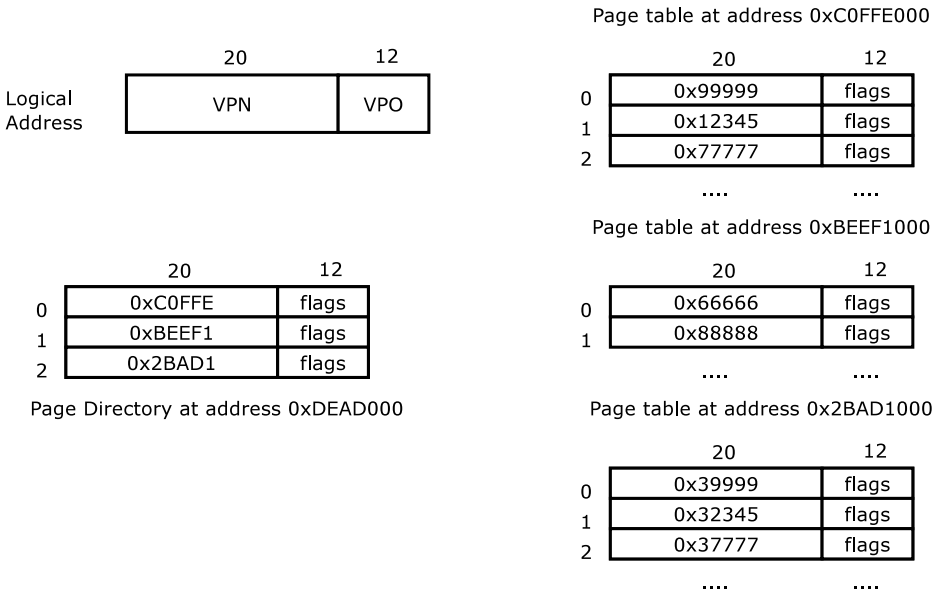
Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

   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

   Solution:
   a) 430: 219 + 430 = 649
   b) 10: 2300 + 10 = 2310
   c) 500: illegal reference
   d) 400: 1327 + 400 = 1727
   e) 112: illegal reference

   (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?

Solution:

a) If a the page table of a process does not contain an entry that maps one or more virtual addresses to a physical address, the process can not access that physical address. Furthermore, the flags in the page directory and the page table entries can indicate that a page can only be accessed by ring 0 (kernel mode).

b) Both processes must have entries in their page tables which map a virtual address to the same physical address. Note that the virtual addresses can be different for both processes.

c) The address 0x00802BAD is split in three parts, the first 10 bits index a page directory entry, the next 10 bits index an entry in the page table which the page directory entry points to, and the last 12 bits are the offset into the page referenced by the page table entry. For our address 0x00802BAD the first 10 bits are 0000000010, so we refernce the second page directory entry, which points us to the page table at address 0x2BAD1000. The next 10 bits of the address are also 0000000010 which point is to the page starting at address 0x37777000. Now we add the last 12 bits of the address and get 0x37777BAD.

d) The physical address 0x77777777 is inside of the page starting at 0x77777000. This page has index 2 in the page table with the index 0. Therefore it is referenced by the virtual address 0x00002777.

e) Page tables are always placed at 4K-boundaries, because the last 12 bits of the address are zero. The same is true for pages itself, as page table entries also only use 20 bits for the address.

f) When a different process is scheduled, the contents of the page directory base register (PDBR) are changed, so that it points to the page directory of the scheduled process.

g) We have to read the page directory entry, the page table entry and read/write the data itself. So 3 times 100 ns = 300 ns.

(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.

Solution: Each page size is 4 KB = 2^{12} Bytes. Hence, we need 12 bits in order to represent the offset and to be able to access each byte in the page. Since we have 1024 pages we need 10 bits to represent all the pages numbers and to be able to access each page. Then, the format of the logical address is:

10 bits – page number	12 bits – offset

(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

Solution:

```
4 Frames (example):
LRU :
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6
x x x x     x x       x x x     x
10 faults

FIFO :
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6
x x x x     x x x x   x x x   x x   x
14 faults

Optimal :
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6
x x x x     x x         x       x
8 faults
```

# Frames	LRU	FIFO	Optimal
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7

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.)

Solution: A sample implementation is available on the course website.

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?

Solution:

a) As each thread increments the value `10e6` times, the resutling counter variable should be twice this number. However, the observed value is much lower and varias between different program runs. This is as the two threads update the shared counter variable at the same time and therefore one thread can overwrite the incrementation done by the other thread.
b) A sample implementation is available on the course website.
c) The program now outputs the expected `2*10e6`.