Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Operating Systems and Networks	Spring 2016
Assignment 4	Due 29 March 2015

1. **Paging**

   (a) **Paging Problem I**

   i. Consider the following two-dimensional array:
   `int X[32][32];`
   Suppose that a system can accomodate 4 pages in main memory and each page is 512 bytes long (assume an integer variable is of size 8 bytes). Suppose The `X` array is stored in row-major order (i.e., `X[0][1]` follows `X[0][0]` in memory). Which of the following two code fragments will generate the less number of page faults? Explain and compute the total number of page faults for each code.
   Fragment 1:
```
for (int j=0; j<32; j++)
  for (int i=0; i<32; i++)
   X[i][j]++;
```
   Fragment 2:
```
for (int i=0; i<32; i++)
  for (int j=0; j<32; j++)
   X[i][j]++;
```

Solution:

a) Fragment 2 will generate less page faults since the code has more spatial locality than Fragment 1. The inner loop causes only one page fault for every other iteration of the outer loop. (There will only be 16 page faults.) But (Fragment 1): Since a frame is 64 words, one row of the X array occupies half of a page (i.e., 32 words). The entire array fits into $32 \times 16/64 = 8$ frames. The inner loop of the code steps through consecutive rows of X for a given column. Thus every other reference to `X[i][j]` will cause a page fault. The total number of page faults will be $32 \times 32/2 = 512$.

   (b) **Paging Problem II**

Suppose that a virtual page reference stream contains repeatative long sequences of page references and then occasionally followed by a page reference outside the sequence. For example, the follwoing code:
```
for (int i=0; i<n; i++){
  for (int j=0; j<S; j++)
     A[j*PAGE_SIZE]++;
  B[random_integer]++;
}
```

   i. What should be the lower bound of the number of pages the main memory allows for this program so that page replacement algorithms like LRU and FIFO does not become ineffective (every access in the loop nest results in fault)?

   ii. If this program were allocated F page frames where $1 < F < S + 1$, describe a page replacement approach that would perform much better than the LRU, FIFO algoirthm.

Solution:

a) If the repeatative page access sequence length is S, every reference will page fault unless the minimum size of page table is $S + 1$,

b) If there are $1 < F < S + 1$ frames, a better algorithm will *pin* $F - 1$ pages to fixed frames and vary only one frame.

(c) **Thrashing Problem I**

i. If there are N processes running in parallel in a processor, we call N "the degree of multiprogramming". What do you expect the CPU utilization will be if we keep increasing the number N. Can these N processes running in parallel cause thrashing? Explain both of your answers.

ii. How can an OS detect thrashing? What action can the OS take to avoid thrashing?

Solution:

a) If we keep increasing N, the CPU utlization will keep going up. If each of these N programs access the same number of pages, they might result into thrashing.

b) The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming N. The OS can intelligently detect processes that are taking too much memory and kill them. However, it is generally not a good idea to let the operating system kill such processes because they might be important. Therefore, it's best for the user to take some actions to prevent thrashing. Some solutions include:

- Increase the amount of RAM in the computer, if N can not be reduced.
- Decrease N based on process priorities.
- Replace programs that are memory-heavy with equivalents that use less memory. (e.g. In most cases a C++ application will consume less memory than the equivalent Java application due to the large overhead of Java's virtual machine, class loading etc.)

(d) **Thrashing Problem II**

Consider a demand-paging system with the following fractions of time consumptions for running N processes (asssume the phenomenons are non-overlapping):

CPU utilization = 20%
Paging activity (including load from disk) = 75%
Other I/O devices =5%

For each of the following, say whether it will (or is likely to) improve CPU utilization. Explain your answers.

- install a faster CPU
- increase the degree of multiprogramming
- decrease the degree of multiprogramming
- install more memory
- install a faster hard disk

Solution: The system obviously is spending most of its time paging, indicating overallocation of memory. If the level of multiprogramming is reduced resident processes would page fault less frequently and the CPU utilization would improve. Another way to improve performance would be to get more physical memory or a faster paging disk.

a) Get a faster CPU - No.

b) Increase the degree of multiprogramming - No.

c) Decrease the degree of multiprogramming - Yes.

d) Install more main memory - Likely to improve CPU utilization as more pages can remain resident and not require paging to or from the disks.

e) Install a faster hard disk, or multiple controllers with multiple hard disks - Also an improvement, for as the disk bottleneck is removed by faster response and more throughput to the disks, the CPU will get more data more quickly.

2. **File System**

   i. Some OSes provide a system call "rename" to give a file a new name. Is there any difference at all between using this call to rename a file and just copying the file to a new file with the new name, followed by deleting the old one?

   ii. What are the differences between hard links and symbolic links?

Solution:

a) Yes there is a difference. The "rename" call does not change the creation time, or the time of last modification, but creating a new file causes it to get the current time as both the creation and last modification times. Also, if the disk is full, the copy will fail.

b) A hardlink isn't a pointer to a file, it's a directory entry (a file) pointing to the same inode. Even if you change the name of the other file, a hardlink still points to the file. If you replace the other file with a new version (by copying it), a hardlink will not point to the new file. You can only have hardlinks within the same filesystem. With hardlinks you don't have concept of the original files and links, all are equal (think of it as a reference to an object).
On the other hand, a symlink is actually pointing to another path (a file name); it resolves the name of the file each time you access it through the symlink. If you move the file, the symlink will not follow. If you replace the file with another one, keeping the name, the symlink will point to the new file. Symlinks can span filesystems. With symlinks you have very clear distinction between the actual file and symlink, which stores no info beside the path about the file it points to.