Operating Systems
Jacobs University Bremen
Romelda Blaceri

Course: CO20-320202
4 November, 2019

Homework 4

Problem 4.1

Solution:

a) The table will be as following:

reference string	1	2	3	4	1	1	4	2	1	2
frame 0	1	1	3	3	1	1	1	1	1	1
frame 1	-	2	2	4	4	4	4	2	2	2
faults	⋆	⋆	⋆	⋆	⋆			⋆		

The pages are mapped in the following way:
→ 1 is mapped in frame 0 → {1}
→ 2 is mapped in frame 1 → {1 2}
→ 3 replaces 1, since it was the first page mapped → {3 2}
→ 4 replaces 2, since it was mapped before 3 → {3 4}
→ 1 replaces 3, since it was mapped before 4 → {1 4}
→ 1 is already mapped to frame 0, so no change → {1 4}
→ 4 is already mapped to frame 1, so no change → {1 4}
→ 2 replaces 4, since 1 was mapped after it → {1 2}
→ 1 is already mapped to frame 0, so no change → {1 2}
→ 2 is already mapped to frame 1, so no change → {1 2}

The number of faults: 6.

b)

reference string	1	2	3	4	1	1	4	2	1	2
frame 0	1	1	1	4	4	4	4	4	4	4
frame 1	-	2	2	2	1	1	1	1	1	1
frame 2	-	-	3	3	3	3	3	2	2	2
faults	⋆	⋆	⋆	⋆	⋆			⋆		

The pages are mapped in the following way:
→ 1 is mapped in frame 0 → {1}
→ 2 is mapped in frame 1 → {1 2}
→ 3 is mapped in frame 2 → {1 2 3}
→ 4 replaces 1, since it was the first page → {4 2 3}
→ 1 replaces 2, since it was mapped before 3 and 4 → {4 1 3}
→ 1 is already mapped to frame 1, so no change → {4 1 3}
→ 4 is already mapped to frame 0, so no change → {4 1 3}
→ 2 replaces 3, since it was mapped before 1 and 4 → {4 1 2}
→ 1 is already mapped to frame 1, so no change → {4 1 2}
→ 2 is already mapped to frame 2, so no change → {4 1 2}

The number of faults: 6.

c)

reference string	1	2	3	4	1	1	4	2	1	2
frame 0	1	1	1	1	1	1	1	1	1	1
frame 1	-	2	3	4	4	4	4	2	2	2
faults	⋆	⋆	⋆	⋆				⋆		

The pages are mapped in the following way:
→ 1 is mapped in frame 0 → {1}
→ 2 is mapped in frame 1 → {1 2}

→ 3 replaces 2, since it will be used after 1 → {1 3}
→ 4 replaces 3, since it will not be used anymore → {1 4}
→ 1 is already mapped to frame 0, so no change → {1 4}
→ 1 is already mapped to frame 0, so no change again → {1 4}
→ 4 is already mapped to frame 1, so no change → {1 4}
→ 2 replaces 4, since it won't be used anymore in our order → {1 2}
→ 1 is already mapped to frame 0, so no change → {1 2}
→ 2 is already mapped to frame 1, so no change → {1 2}

The number of faults: 5.

d)

reference string	1	2	3	4	1	1	4	2	1	2
frame 0	1	1	1	1	1	1	1	1	1	1
frame 1	-	2	2	2	2	2	2	2	2	2
frame 2	-	-	3	4	4	4	4	4	4	4
faults	⋆	⋆	⋆	⋆						

The pages are mapped in the following way:
→ 1 is mapped in frame 0 → {1}
→ 2 is mapped in frame 1 → {1 2}
→ 3 is mapped in frame 2 → {1 2 3}
→ 4 replaces 3, since it won't be used anymore considering our order → {1 2 4}
→ 1, 1, 4, 2, 1, 2 are already mapped to their respective frames, so we don't have any change till the end → {1 2 4}

The number of faults: 7.

e)

reference string	1	2	3	4	1	1	4	2	1	2
frame 0	1	1	3	3	1	1	1	2	2	2
frame 1	-	2	2	4	4	4	4	4	1	1
faults	⋆	⋆	⋆	⋆	⋆			⋆	⋆	

The pages are mapped in the following way:
→ 1 is mapped in frame 0 → {1}
→ 2 is mapped in frame 1 → {1 2}
→ 3 replaces 1, since it has not been used for the longest period of time comparing to 2 → {3 2}
→ 4 replaces 2, since 3 was used after it → {3 4}
→ 1 replaces 3, since 4 was used after it → {1 4}
→ 1, 4 cause no change → {1 4}
→ 2 replaces 1, since the last mapped, therefore used page was 4 → {2 4}
→ 1 replaces 4, since 2 was just mapped → {2 1}
→ 2 is already mapped to frame 0, so no change → {1 2}

The number of faults: 5.

f)

reference string	1	2	3	4	1	1	4	2	1	2
frame 0	1	1	1	4	4	4	4	4	4	4
frame 1	-	2	2	2	1	1	1	1	1	1
frame 2	-	-	3	3	3	3	3	2	2	2
faults	⋆	⋆	⋆	⋆	⋆			⋆		

The pages are mapped in the following way:
→ 1 is mapped in frame 0 → {1}
→ 2 is mapped in frame 1 → {1 2}
→ 3 is mapped in frame 2 → {1 2 3}
→ 4 replaces 1, since it was the first used page, so the one that hasn't been used for the longest period of time → {4 2 3}

→ 1 replaces 2, since it was used before 3 and 4 → {4 1 3}
→ 1 is already mapped to frame 1, so no change → {4 1 3}
→ 4 is already mapped to frame 0, so no change → {4 1 3}
→ 2 replaces 3, since it was used before 1 and 4 → {4 1 2}
→ 1 is already mapped to frame 1, so no change → {4 1 2}
→ 2 is already mapped to frame 2, so no change → {4 1 2}

The number of faults: 6.

Problem 4.2

Solution:

a) The command: `echo "hello world" | \ ./cat++ -l ./librot13.so -l ./libupper.so -l ./librot13.so` prints `HELLO WORLD` in the screen. The transformations are performed using the transform functions in upper.c and rot13.c. What these transformations do is:
→ Transform function in rot13.c adds 13 to each letter (character ch) if ch+13 doesn't exceed the letter z (or Z in the case of uppercase characters), and subtracts 13 otherwise.
→ Transform function in upper.c converts all letters in the string to uppercase letters.
Therefore, the above command, first executes rot13, which turns the string into: `uryyb jbeyq` (h+13 = u, e+13 = r, l+13=y, o+13>z => o-13=b and so on).
Then, upper will turn the string into: `URYYB JBEYQ`. After that, we have again rot13. Considering the fact that the English alphabet has only 26 letters, when we perform rot13 transformation two times, in the second time we get the same string as the one we had in the beginning. So, we will have "hello world" again, but now in uppercase letters, since we previously transformed to upper. So, final string: `HELLO WORLD`.

b) This is the table we get after we run `pmap` (extended):

```
15344:   ./cat++ -l ./librot13.so -l ./libupper.so -l ./librot13.so
         Address Perm   Offset Device   Inode  Size KernelPageSize MMUPageSize  Rss Shared_Clean  Mapping
    559da1633000 r--p 00000000 08:03 1704166     4              4           4    4            0  cat++
    559da1634000 r-xp 00001000 08:03 1704166     4              4           4    4            0  cat++
    559da1635000 r--p 00002000 08:03 1704166     4              4           4    4            0  cat++
    559da1636000 r--p 00002000 08:03 1704166     4              4           4    4            0  cat++
    559da1637000 rw-p 00003000 08:03 1704166     4              4           4    4            0  cat++
    559da3341000 rw-p 00000000 00:00       0   132              4           4   20            0  [heap]
    7f8a3d6f3000 rw-p 00000000 00:00       0    12              4           4    8            0
    7f8a3d6f6000 r--p 00000000 08:03 1051202   148              4           4  144          144  libc-2.29.so
    7f8a3d71b000 r-xp 00025000 08:03 1051202  1484              4           4 1032         1032  libc-2.29.so
    7f8a3d88e000 r--p 00198000 08:03 1051202   292              4           4  136          136  libc-2.29.so
    7f8a3d8d7000 r--p 001e0000 08:03 1051202    12              4           4   12            0  libc-2.29.so
    7f8a3d8da000 rw-p 001e3000 08:03 1051202    12              4           4   12            0  libc-2.29.so
    7f8a3d8dd000 rw-p 00000000 00:00       0    16              4           4   16            0
    7f8a3d8e1000 r--p 00000000 08:03 1051204     4              4           4    4            4  libdl-2.29.so
    7f8a3d8e2000 r-xp 00001000 08:03 1051204     8              4           4    8            8  libdl-2.29.so
    7f8a3d8e4000 r--p 00003000 08:03 1051204     4              4           4    0            0  libdl-2.29.so
    7f8a3d8e5000 r--p 00003000 08:03 1051204     4              4           4    4            0  libdl-2.29.so
    7f8a3d8e6000 rw-p 00004000 08:03 1051204     4              4           4    4            0  libdl-2.29.so
    7f8a3d8e7000 rw-p 00000000 00:00       0     8              4           4    8            0
    7f8a3d91a000 r--p 00000000 08:03 1050691     4              4           4    4            4  ld-2.29.so
    7f8a3d91b000 r-xp 00001000 08:03 1050691   132              4           4  132          132  ld-2.29.so
    7f8a3d93c000 r--p 00022000 08:03 1050691    32              4           4   32           32  ld-2.29.so
    7f8a3d944000 r--p 00029000 08:03 1050691     4              4           4    4            0  ld-2.29.so
    7f8a3d945000 rw-p 0002a000 08:03 1050691     4              4           4    4            0  ld-2.29.so
    7f8a3d946000 rw-p 00000000 00:00       0     4              4           4    4            0
    7ffda68ef000 rw-p 00000000 00:00       0   132              4           4   12            0  [stack]
    7ffda69cb000 r--p 00000000 00:00       0    12              4           4    0            0  [vvar]
    7ffda69ce000 r-xp 00000000 00:00       0     4              4           4    4            4  [vdso]
ffffffffff600000 r-xp 00000000 00:00       0     4              4           4    0            0  [vsyscall]
                                                ====           ====        ====  ====         ====
                                                2492            116         116  1624         1496
```

Concerning the first question, we observe that there are added 5 lines in the table whenever a different library is called, so there are added 5 memory segments when one of the libraries implementing a transform is loaded. As for the second question, we keep in mind that in the address section, we actually have the logical addresses, as user space programs hardly ever deal with physical addresses. We notice that the logical addresses are different between the two same libraries that are called; we'd reasonably expect them to be the same (since its the same program running), but there is a security feature called address space layout randomization that intentionally randomizes them. Also, considering the columns of Size and Resident Size (Rss), they're different since the entire library segment has not been loaded. Now, we also consider the 2 last columns: Rss and Shared clean. In some cases they have the same values, but we also have many values that differ. Shared memory means the physical pages are shared between multiple processes, and for the matching cases with Rss, those library pages that are in memory are shared.

c) Considering the first case, `cat -n`, the current version of the cat++ program allows us to implement such a transform, as the implementation consists of printing each line numbered and with some spaces after the number and before the actual line, which we're able to do in C. So basically we need to add a prefix to each line, which consists of a number and some spaces. Since spaces are characters, adding them as a prefix to each line won't be a problem. As for the number, we can solve the issue of printing the number of the line in each call by adding static in the initialization of the number variable. So if we initialize some `static int count=1` in the beginning and increase `count++` in the end of our function, `count` will increase by 1 every time we call the transform function, so every line will start with a number one bigger than the previous line, which gives us cat -n.
Considering `wc`, the current version of the cat++ program does not allow us to implement such a transform. This is because in our version of the cat++ the transforms are applied line by line, so we would have to implement a transform that keeps track of the number of lines, words, and chars and also the name of the file from which the lines are taken. But we have no possible way of getting the file name and no way to know which line is the last so that we can print the desired output only once. Thus there is no way we can implement a transform emulating `wc`.