Jinxuan Jiang

CMPS111-HW4

Question 1

a.) Granting read access for all users:

To grant read access for all users, consider the following:

If a group are supposed (say ‘all’), then granting read access for all users in ACL is quiet easy as it would just mean: create a group called ‘all’; Make a ACL for the file name, with ACL as ‘all’ R

If groups are not supported, then granting read access for all users is easier to implement with capabilities list as: ACL will now require along list, enumerating all users; with capability list, just put capability for the file in a well-known place in a capability system.

b.) Revoking write access from all users:

To revoke write access for a file from all users, the best approach would be ACL: the only thing that needs to be done is to edit the ACL to make the change. Again, if the group all W was the entry added to ACL. It just needs to be removed, in turn revoke all those rights. It can also be done with capability list by changing the check field stored with the object.

c.) Granting write access to three specific users

To revoke write access to three specific users of the two methods can be used. It can be implemented with ACL as follows: construct ACL for the file for all users & the ACL for file would be: three ‘users:w’. With capability the implementation would involve making CL for each users: First construct the capability list for the file, ‘file:w’, then, for each user names assign this capability.

d.) Revoking execute access from four specific users:

To revoke execute from four specific users would be ACL: The only thing that needs to be done is to edit the ACL to make the change. Capability list as it is, do not allow selection.

Question 2

a)

Here, processor B is more likely to get affected by a buffer overflow attack in the operating system, because both the program and the data is stored in the single cache. Basically, the motive of buffer overflow attack is to leak data from the program to other buffers which are not allocated to it and it will result into corruption of data which is not allocated to the given executable code and the attacker will get succeeded in affecting some private information of the use, so what will happen here is in the single cache the executable code will allow the buffer to overflow and the leak will affect the data in processor 2 whereas in processor 1 being the executable code and the data in different locations will make sure that the overflow doesn't affect the data.

b)

If the Operating System is providing runtime protection of the memory then it will be hard for the buffer to affect the memory be it heap or stack implementation regardless of the processor.

Question 3

(a) First fit: (a) 20MB (b) 10MB (c) 18MB

i) For 12MB it will check holes which it fits into : 10MB, 4MB, 20MB. Since 20 MB is greater than 12MB. It will fit.

ii) For 10MB it will check holes which it fits into : 10MB. Since 10 MB is equals, it will fit.

iii) For 9 MB it will check holes which it fits into : 4MB, 18MB. Since 18 MB is greater than 9 MB, it will fit.

(b) Best fit: (a) 12MB (b) 10MB (c) 9MB

i) For 12MB it will check holes which it fits into : 10MB, 4MB, 20MB, 18MB, 7MB, 9MB, 12MB. Since 12MB is best fit for 12MB. It will fit.

ii) For 10MB it will check holes which it fits into : 10MB. Since 10 MB is equals to 10MB, it will fit.

iii) For 9 MB it will check holes which it fits into : 4MB, 20MB, 18MB, 7MB, 9MB. Since 9 MB is greater than 9MB, it will fit.

(c) Worst fit: (a) 20MB (b) 18MB (c) 15MB

i) For 12MB it will check holes which it fits into : 10MB, 4MB, 20MB. Since 20 MB is highest, it will fit.

ii) For 10MB it will check holes which it fits into : 10MB, 4MB, 18MB. Since 18MB is highest, it will fit.

iii) For 9 MB it will check holes which it fits into : 10MB, 4MB, 7MB, 9MB, 12MB, 15MB. Since 15 MB is highest, it will fit.

(d) Next fit: (a) 20MB (b) 18MB (c) 9MB

i) For 12MB it will check holes which it fits into : 10MB, 4MB, 20MB. Since 20 MB is next highest to 12MB. It will fit.

ii) For 10MB it will check holes which it fits into : 18MB. Since 18MB is next highest to 18MB, it will fit.

iii) For 9 MB it will check holes which it fits into : 7MB, 9MB. Since 9 MB is next highest to 9MB, it will fit.

Question 4:

The number of bits occupied by Virtual Page Numbers = 9+11=20 bits

The number of bits occupied by page offset = total bits - VPN bits = 32 - 20 = 12 bits

The number of bytes per page = 2^12 = 4096 bytes = 4 KB

Hence, the page size is 4 KB

Number of pages in the address space = 2^20 pages

Question 5:

a.)

Internal Fragmentation

1. When memory is allocated to a process is larger than the memory requested by the process, the amount of memory not used by the process leads to internal fragmentation, this memory cannot be allocated to another process and wasted
2. Internal fragmentation occurs in programs or process where the program is divided into same size fixed partitions, in which at least one partition would be smaller than the memory block allocated
3. Internal fragmentation can be overcome by allocating multiple variable sized memory blocks, and compact all blocks into one large block
4. Internal fragmentation is usually observed in paging systems where fixed size pages are allocated for a program and last partition of the program may not require the entire page.

External Fragmentation

1. External fragmentation occurs when a process is allocated with memory which is not contagious; sometimes the memory blocks in between these allocated memory blocks remain unused leading to external fragmentation
2. External fragmentation occurs when programs are allocated exactly the requested amount of memory in multiple variable sized memory blocks.

3. External fragmentation can be reduced by compaction that is moving all the free memory blocks to one place and making it into a large memory block which can be allocated to a process
4. External fragmentation is seen in segmentation where memory is not allocated contagiously and in fixed partitions, leading to unused memory blocks

b.)

In computer operating systems, **paging** is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory. In this scheme, the operating system retrieves data from secondary storage in same-size blocks called *pages*. Paging is an important part of virtual memory implementations in modern operating systems, using secondary storage to let programs exceed the size of available physical memory.

c.)

We can reduce external fragmentation via **Compaction** and **Paging**

For compaction

- Shuffle memory contents around around to place all free memory together large block
- Compaction is possible only if relocation is dynamic and is done at execution time

But it generates a problem:

All user processes have to wait whilst OS re-shuffles the free memory.

For paging

- Map logical address space of a process to non-contiguous physical memory

And paging solves the problem above:

- Process is allocated physical memory from where ever it is available.

That’s why paging is considered a better memory management technique than compaction.