John Anderson

CS 149 Section 3

HW #3

Part 1:

* 1. As the address space grows without increasing the page size more fragmenting pages will be adding to.
  2. The space If the page size increases then the size of the page’s increases and not the size of the number of pages for fragmenting in the list.
  3. Having really big pages can cause internal fragmentation.
  4. Increasing the percent of pages located in each address space makes some of the VPS’s not valid the lower you go and higher valid VPS’s and higher virtualization room with high percent of page fracturing.
  5. The first one seems to gets index out of bounds error.
  6. Seems fine.
  7. The Last one is externally big and can be unrealistic for some causes where this size is not needed. Also has a lot of data it has to keep track of because of size.

1. 1. I get an error for the simulation saying that the size of Memory must be bigger than address space. So the simulation won’t be allowed to run. It won’t work because the address can’t fit into the physical space that was allocated to it, so it’s not possible.

Part 2:

* 1. All of the Free() should return 0(or maybe the fragmentations) and I think alloc() is giving information on where free space would be and what elements are in list currently.
  2. The free list after each requests will try to find the Best fit for the data
  3. The free list will have wasted memory space.

1. Alloc returns a high address to put data in, and the data in the free list is a lot more fragmented making a lot of memory be wasted
2. When using first it will just put the data in the first possible open memory slot that is available, which increases the speed, because it doesn’t spend as much time looking for space.
3. ADDRSORT and SIZESORT+ seem to doing the same things to the data by having them be sorted form the smallest to the largest and SIZRSORT- has the largest sorted first.
   1. It looks like the longer it goes the alloc leads to fracturing and not run all 1000 random allocations
   2. The pattern seems like the longer it runs the bigger chance that there will be wasted memory space and less elements of data can be added.
   3. Frist fit seems to hand all of the allocations the best, and worst fit seems to fracture the data early as it goes on.
   4. As the fraction nears 0 it keeps fragmenting until every spot in the free list is its own fragmented peace of data
   5. As fraction nears 100 it seems to avoid fragmentation for data keeping it together in 1 data chunk.
4. It seems that using the -A you can make the data start at a memory point reducing fragmentation because the free memory used it the (-A) won’t be used keeps the data after the free memory. When I tried WORST, BEST, and FIRST, they all had 1 element not being fragmented.

Part: 3

1. The buddy allocator is more likely to lead to internal fragmentation then the slab. Because if you have a large lower limit for buddy system it can cause more fragmentation then the slab will cause.
2. For External fragmentation in most causes they seem to be similar, but in unusually causes the Slab has far worse external fragmentation then buddy system because slab can request larger buffers then needed and over time they can lead to more fragmentation then would appear in Buddy system.
3. The size of the slab can’t be changed until it is clear, so if a slab is created and 1 of the ideas in the slab is constantly being used and not leaving the cash, then the size of the slab is never changed, leaving a lot of fracturing to the data.