Brian Schillaci (netid: bcs115)
John Strauser (netid: jps313)
04/08/19

CS416 Project 3: User-level Memory Management Report

1. **Descriptions of implementation:**

TLB

- An array of the tlb struct from the header file of TLB_SIZE.
- GetTLB() - If the translation exists in the TLB, return it. Otherwise return NULL.
- AddTLB() - add a translation to the TLB. Uses hashing to determine which value to replace in the TLB, if necessary.
- RemoveTLB() - Takes the provided virtual address and its associated physical page out of the TLB.

Set_physical_mem

- Set init to 1 to make sure the function only runs once.
- Calculate the number of bits for each part of the address.
- Use memalign to allocate the physical memory.
- Use calloc to allocate the page directory, virtual bitmap, and physical bitmap.
- We needed to set the first virtual page in the bitmap to 1, in order to skip virtual page 0. This was needed in order to prevent a NULL failure for address 0x0 in malloc.

Translate

- Check if the virtual address exists in the TLB, if it does provide the page stored in the TLB and skip the rest of the function.
- From the virtual address calculate the page directory offset by isolating the left 10 bits of the virtual address parameter.
- Get the page table for this offset. If it is null return null.
- Get the page table offset by isolating the middle 10 bits of the virtual address parameter.
- Use this page table offset to get the page table entry. If it is not yet assigned, return null.
- Get the page offset by isolating the right 12 bits of the virtual address parameter.
- The physical address that the virtual address parameter translates to is the address the page table entry points to offset by the page offset.

- Add the virtual to physical translation to the TLB.

Page_map

- Similar to the translate, isolate the page directory offset and page table offset.
- Use those two values to determine the page table entry. If the page table that is to be mapped does not exist yet, allocate the space for it.
- If the page table entry has not been assigned a physical address yet, assign it the physical address parameter and return 1 for success. Otherwise, return -1 for failure as you cannot map a page that is already mapped.

A_malloc

- If the number of bytes to be allocated is negative or greater than the memory size, return null.
- Lock mutex for thread safety.
- If this is the first malloc call, initialize all memory in set_physical_mem.
- Determine the number of pages being requested.
- Make sure there is enough free physical pages, if there is not unlock the mutex and return null.
- Use get_next_avail to get the address of the next set of contiguous virtual pages. If no contiguous set of pages exists, unlock the mutex and return null.
- Using the physical bitmap, make each page's bit 1 to show the page is in use. Also map the page to its physical address.
- Remove the physical page and virtual page mapping from the TLB
- Once complete unlock the mutex and return the virtual address.

A_free

- Lock mutex for thread safety.
- If this is the first time the program is running, initialize all memory using set_physical_mem.
- Determine the number of pages that need to be freed.
- Use isValidVa() to make sure the virtual address provided is valid. If not, unlock the mutex and end the function.
- Using the virtual address parameter set each page table entry to 0 and set the corresponding bits in the virtual and physical bitmaps to 0 to show that they are not in use. Do this for each page that needs to be freed.

Get_value

- Lock mutex for thread safety.
- If this is the first time the code runs, call set_physical_mem.
- Ensure the virtual address parameter is not null and is a valid virtual address. If not, unlock the mutex and stop the function.

- For the size requested, translate each virtual address and put it in the val parameter.
- Unlock the mutex.

Mat_mult

- Loop through the matrices provided and use get_value to calculate sum. Then use put_value to store the sum.

2. **How much max memory able to allocate:**
   We were able to memalign a max of 3.3GB for our MEMSIZE. We left it at 2GB just to be safe for benchmarking.

3. **Number of page table entries after running the benchmark:**
   4 total pages were created for the benchmark. 3 occurred concurrently for the arrays used in mat_mult. The 4th occurred after those 3 pages were freed.
4. **benchmark performance (time to run):** 0.002 secs