Maitreyi Rajaram
net id: mr1423

Virtual Machine Implementation

Implementation:

Data structures -
The page directory (PGDIR) consists of page tables which hold page table entries (void* pa).
Bitmaps are used to track the physical and virtual pages available (setting the bit to 1 denotes an allocation while unsetting to 0 denotes a freed page).

When myalloc is called, physical memory is set if required and the number of pages to allocate is calculated based on the number of bytes passed. Get_next_avail checks both physical and virtual page bitmaps and returns pointers to the beginning of the allocation required. Bitmaps are then set as long as both physical/virtual memory is available and the paging is mapped in PGDIR. *Assumption that physical memory is contiguous for this implementation*

Pagings are mapped using Translate() to get the page directory index, page table index, and offset bits to obtain/set the physical address ptr at the appropriate dest.

myfree simply unsets bits in the physical/virtual bitmaps accordingly (assumes that user will overwrite the paging on next allocation).

Critical sections that were made thread safe:
-setting/unsetting bitmaps in myalloc and myfree
- memcpy (setting val to physical page) in PutVal
Allocations should not overlap and physical pages are not shared

TLB –
Implemented using a linked list consisting of “tlb entry nodes” that hold pointers to both the physical address and virtual address.
If “hit” then priority number in node is incremented. Node is removed from list and inserted at head.
If “miss” then node (if present in TLB) is moved to tail. “Miss” counter incremented.
If miss and TLB is full then tail is evicted and head becomes new hit.
To lookup page in TLB, traverse LL and move most recently looked up (ie “hit”) to the head. Uses virtual address – offset to index. If not found in TLB or PGDIR, do not count as a miss.

Benchmark output for VM Library:

```
Allocating three arrays of 400 bytes
Addresses of the allocations: 1000, 2000, 3000
Storing integers to generate a SIZExSIZE matrix
```

```
Fetching matrix elements stored in the arrays
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Performing matrix multiplication with itself!
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
Freeing the allocations!
Checking if allocations were freed!
   free function works
```

Support for Different Page sizes –

The VPN bits are calculated based on the Page size set in PGSIZE. The total number of bits is log2(virtual memory ie MAX_MEMSIZE). The offset bits are log2(PGSIZE) which leaves the remaining bits allocated for both the page directory and page table indexing. The number of bits for each index is determined by an even split (if this is an odd number of bits, then page directory will always have the additional bit).

Possible Issues/Challenges:

The assumption in this implementation is that physical memory is contiguous. Modifications required in order to account for segments would be returning an array of void* from get_avail_pages_phys so that the library is able to track all used pages in both the bitmap/page directory. (Made this assumption based on project discussion at recitation/the method signature given). Additionally, the simple traversal in this implementation to mark the physical pages bitmap and mapping within the page directory currently use a fixed loop determined by the number of pages (ie. Begins at the first address indicated by a void*). However, if the method were modified to hold an array of void* representing all of the physical pages, then the traversals mentioned above would traverse the array of page pointers.

Some aspects of LRU eviction do not work in TLB implementation – logic is explained.