Jon Cobi Delfin

NetID: jjd279

iLab machine: interpreter.cs.rutgers.edu

****I added the -lpthread and -lm flags to the makefiles, please be sure to add them**

Design:

set_physical_mem():

First, I determined the number of bits required to separate out the page directory entry, page table entry, and offset from the addresses. The result was also used in determining the number of page directory entries. I also had to mmap a region of memory so it can act as physical memory. Since this was a 2-level page table, I only allocated enough memory initially to hold the number of page directory entries; individual page tables (inner level) will be allocated when needed. Finally, I created the bitmaps. The virtual bitmap keeps track of the virtual pages while the physical bitmap keeps track of the available physical addresses. Also, I initialized a mutex for multi-threaded testing.

translate():

I modified the parameters for this function, so it now takes an unsigned long **pgdir instead of pde_t *pgdir

This function simply separates out the page directory entry, page table entry, and offset from the virtual address. It uses a series of bit masks and bit shifts to manipulate the given virtual address. Using the pde and pte, I accessed the corresponding physical translation and added the offset onto the end of it.

page_map():

I modified the parameters for this function, so it now takes an unsigned long **pgdir instead of pde_t *pgdir

The beginning of this function is similar to translate, where I extract the pde, pte, and offset from the virtual address. This function also checks the virtual bitmap to see if the page table is available, and allocates its when necessary. It then maps the physical translation to that entry in the page table and sets the bit in the virtual bitmap.

get_next_avail():

This function searches the virtual bitmap and returns the beginning address of the available virtual space.

a_malloc():

This function is uses get_next_avail(), findPhys(), and page_map() to find a physical address space that can hold the num_bytes and returning its virtual translation. It starts by calling findPhys to find the physical space large enough to hold the num_bytes and sets the corresponding entries in the physical bitmap. It then finds a virtual address using get_next_avail() and maps the physical to virtual translation using page_map().

a_free():

In order to free the size requested, I first had to translate the virtual address to a physical one. Using that address, I search the physical address space and toggle corresponding bits to 0, indicating that it can be overwritten. It also checks the virtual bitmap to see if an entire page table is empty, so that it can be freed.

put_value():

I use translate to get the physical address and store the contents of val into the physical address

get_value():

Similar to put_value, except here I get the value in the physical address and store it in val.

mat_mult():

Using the hints provided and the benchmark/test.c as a rough guideline, I take the values in each of the matrices using get_value, perform the calculations, and store the results in the answer matrix using put_value().

Helper functions:

setbit(), getbit(), clearbit():

These functions set, get, and clear a bit in the bitmaps.

Note about virtual bitmap: I used an unsigned long, which is 4 bytes long. With 8 bits per byte, I can reference 32 entries in the bitmap with a single unsigned long. The size of my virtual bitmap was calculated using MAX_MEMSIZE/PGSIZE to get the number of virtual pages. I then divided by 32 to get the actual number used to reference each page.

findPhys():

This function simply searches the physical address space for the number of bytes requested and returns the corresponding physical address