AUTHOR: Amber Ott
UofC ID: 10013789 (aeott@ucalgary.ca)
CLASS: CPSC457 – Assignment 3
DATE: November 20, 2008

What works/doesn't: I think all parts of the snapshot program work according to the assignment spec.

 # output scripts #

SimpleSnapshot.out	–	This shows part one working, which shows very basic information about the processes on the system.
Filtering.out	–	This shows the filtering working.
Zooming.out	–	This shows the zooming of one process working.
Movie.out	-	This shows the movie function working while zooming on one process while its memory space grows.

 # Linux files – ones I imported into /tmp/aeott/linux #

**/tmp/aeott/uml/linux/include/asm-um/arch/unistd_32.h (line 333 – my four system calls)

**/tmp/aeott/uml/linux/arch/x86/kernel/syscall_table_32.S (line 327 – for my four system calls)

**/tmp/aeott/uml/linux/kernel/Makefile - Added snapshot.o to obj-y list (line 12)

**/tmp/aeott/uml/linux/include/linux/snapshot.h - my process_keeper, mem_keeper, and vm_keeper struct definitions are here.

**/tmp/aeott/uml/linux/arch/um/kernel/snapshot.c
**/tmp/aeott/uml/linux/kernel/snapshot.c

This is where my four system calls are defined
(please see the files for more documentation):

int sys_snapshot(struct task_struct* task, struct process_keeper* pk)
PURPOSE: Initializes a process keeper structure with memory information from a process

int sys_snap_pid(int pid, int uid, int user, int * numPs)
PURPOSE: Takes memory snapshots of specified processes (by PID, UID, user-status, or by system-status).

int sys_get_ps(struct process_keeper * pe)
PURPOSE: Copies the current process_keepers that were created in a snapshot into the specified USER array of process_keepers (must be same size).

int sys_get_vmas(struct vm_keeper* ve, int index)
PURPOSE: Copies the current kernel array of vmas for a specified process (initialized in a previous snapshot) to a USER array of the same type and size.

 # User-level program

snap.c – This is the main driver program for my assignment.

 # How I did it:

In snap.c: line 50 to 104 is where I call my system calls to take the memory snapshot
In snapshot.c: the whole file shows my system call implentations

Simple Snapshot:

1. I used my kernel system call snap_pid(int, int, int, int) to go through every process in the system and create (kmalloc) a process_keeper struct for each one.
2. For every process I call my system call snapshot(task_struct*t, process_keeper* pk) which sets pk's memory size (which I get from its mm_struct->vm_total). If the mm_struct is NULL, then I zero everything out for that struct.
3. For each process I also set its pid and uid in its process_keeper struct and I keep a count of each process (which is copied to user space as the number of processes in the system.)
4. Then I call the system call get_ps(process_keeper *) which copies the kernel process_keepers into

user space.

5. In the main driver (snap.c) I go through every process_keeper and display this information.

Filtering:

1. Like (1) above, except filter the tasks that I create process_keepers for. For each process I do the corresponding comparison, and only those that satisfy the comparison will be snapshot.
   PID filter: pid == task->pid
   UID filter: uid == task->uid
   Users filter: task->uid > 0 && task->pid > 100
   Systems filter: task->uid == 0 && task->mm_struct ==NULL
2. Like (2), (3), (4) and (5) in the Simple Snapshot.

Zooming:

1. I just added more fields to my process_keeper struct. Here is an explanation of what I used and why:

   The mm is within every process:

```
pk->num_VM_areas      = mm->map_count;       // the number of vmas the process has
pk->text_seg_start    = mm->start_code;      // the text segment start address
pk->text_seg_end      = mm->end_code;        // the text segment end address
pk->data_seg_start    = mm->start_data;      // the data segment start address
pk->data_seg_end      = mm->end_data;        // the data segment end address
pk->stack_start       = mm->start_stack;     // the stack segment start address
pk->heap_start        = mm->start_brk;       // the heap segment start address
pk->total_vm          = mm->total_vm;        // the total size of memory the process uses
pk->share             = mm->mm_users.counter;// the sharing information ( how many other
                                                processes share this space
```

Movie:

1. If the movie is enabled, the user specifies the number of frames to output and how many seconds of lag time in between seconds. After each snapshot is taken, the frame number is decremented (program stops when this gets to zero) and then the program sleeps for the specified number of seconds. That's it!