Institute of Technology
University of Washington – Tacoma
Instructor: Wes Lloyd

# Assignment 4

Page Table Walking - Linux Kernel Module

Due Date: Wednesday March 15th, 2017 @ 11:59 pm

## Objective

The purpose of this assignment is to modify/refactor the Linux Kernel Module from Assignment #2 to generate a new report which dereferences the memory pages of processes to determine the number of pages that are allocated contiguously vs. non-contiguously. Output should be to the same "/proc" file as in assignment 2.

Output should be generated in comma separated value (CSV) format.
The format should be as follows:

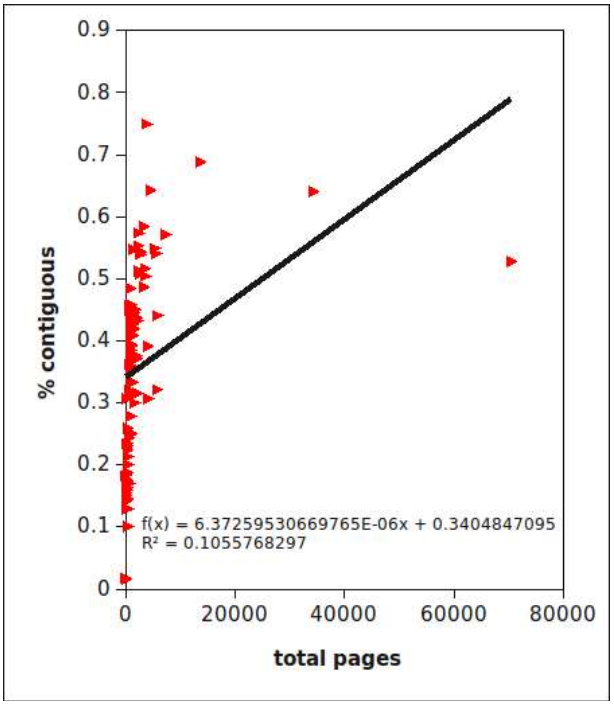
```
PROCESS REPORT:
proc_id,proc_name,contig_pages,noncontig_pages,total_pages
654,auditd,95,353,448
663,audispd,61,180,241
665,sedispatch,54,202,256
679,rsyslogd,545,443,988
680,smartd,177,406,583
. . .
5626,kworker/0:2,315,500,815
5641,kworker/3:0,315,500,815
6165,sleep,34,121,155
6182,cat,32,120,152
,,218407,124327,342734
```

The first line says "PROCESS REPORT:".
The second line is a comma-separated header line describing the columns. The columns are **proc_id** for the process ID, **proc_name** for the process name (the comm field of task_struct), **contig_pages** is the number of contiguous pages, **noncontig_pages** is the number of non-contiguous pages, and **total_pages** it the total number of pages for the process.

The last line of the report should have blank values for the first two columns and provide a sum of the total number of contig pages, total number of non-contig pages, and the total number of pages for all processes with PID > 650.

It should be possible to capture this CSV output and open this in a spreadsheet to further analyze the data – to - for example, calculate the % of contiguous pages per process, as well as the average % of contiguous pages. (*Floating point arithmetic in a kernel module is not recommended.*)

When completing this exercise on a CentOS 7 VM, the average % of contiguous pages per process was ~35.69%. When totaling all pages (for PIDs > 650), there were 134,281 contiguous pages, 133,255 non-contiguous pages of a total of 267,536 total pages which is about ~50.19% average contiguous-ness for the virtual memory pages mapped to physical memory.




Above is a graph that shows (%) contiguous memory pages relative to the total number of pages for the process. There's a weak relationship where larger processes, with more pages, tend to have a higher percentage of contiguous memory pages than do small processes with fewer pages. The graph shows this relationship above, though there were many more small processes than larger processes so fitting a line is rough. **We do not see an inverse relationship – which is key!** If a compute server has a very long uptime, one theory says that memory pages will become less contiguous and more fragmented over time.

### Implementation and Constraints

An excellent starting point is provided from this stack overflow post regarding performing virtual to physical page address translations.

http://stackoverflow.com/questions/20868921/traversing-all-the-physical-pages-of-a-process

The code goes as follows:

```
struct vm_area_struct *vma = 0;
unsigned long vpage;
if (task->mm && task->mm->mmap)
    for (vma = task->mm->mmap; vma; vma = vma->vm_next)
        for (vpage = vma->vm_start; vpage < vma->vm_end; vpage += PAGE_SIZE)
            unsigned long phys = virt2phys(task->mm, vpage);
```

```
//...
//Where virt2phys would look like this:
//...

pgd_t *pgd = pgd_offset(mm, virt);
if (pgd_none(*pgd) || pgd_bad(*pgd))
    return 0;
pud = pud_offset(pgd, virt);
if (pud_none(*pud) || pud_bad(*pud))
    return 0;
pmd = pmd_offset(pud, virt);
if (pmd_none(*pmd) || pmd_bad(*pmd))
    return 0;
if (!(pte = pte_offset_map(pmd, virt)))
    return 0;
if (!(page = pte_page(*pte)))
    return 0;
phys = page_to_phys(page);
pte_unmap(pte);
return phys;
```

Essentially above, the virtual memory areas of a process are walked. VMAs are described by the struct vm_area_struct. A linked list is provided which can be walked to obtain the virtual addresses.

**For virt2phys, implement a function based on the code above that takes in a task->mm and a vpage to then return an unsigned long address.**

For this assignment, you should **only** analyze PIDs > 650.

If virt2phys should return 0 at one of the stages, for example while translating either the pgd, pud, pmd, or pte, it is ok *for this assignment* to ignore the 0, and just keep looping. Treat a 0 as an unmapped/untranslatable page entry *for this assignment*.

Linux provides structures for a 4-level page table where pgd_t is the page directory, pud_t is an upper page directory, pmd_t is a middle page directory, and pte_t is a page table entry. There is no guarantee that all of these 4-levels will be physically backed by all HW (CPUs) or all specific compilations of the Linux kernel.

pgd_t is the page directory type
pud_t is the page upper directory type
pmd_t is the page middle directory type
pte_t is the page table entry type

pgd_offset(): returns pointer to the PGD (page directory) entry of an address, given a pointer to the specified mm_struct

pud_offset(): returns pointer to the PUD entry (upper pg directory) entry of an address, given a pointer to the specified PGD entry.

pmd_offset(): returns pointer to the PMD entry (middle pg directory) entry of an address, given a pointer to the specified PUD entry.

pte_page(): pointer to the struct page() entry corresponding to a PTE (page table entry)

pte_offset_map(): Yields an address of the entry in the page table that corresponds to the provided PMD entry. Establishes a temporary kernel mapping which is released using pte_unmap().

Reference slides describing Linux virtual memory areas are here:
http://www.cs.columbia.edu/~krj/os/lectures/L17-LinuxPaging.pdf

For determining contiguous page mappings, just calculate the next address by adding PAGE_SIZE to the current page address. If the next page in the process's virtual memory space is mapped to the current page's physical location plus PAGE_SIZE then this is considered a contiguous page – record a "tick" for a contiguous mapping. IF not, record a tick for a "non-contiguous" mapping.

### Example Hello World Module

A sample kernel module is here:
http://faculty.washington.edu/wlloyd/courses/tcss422/assignments/hello_module.tar.gz

To extract the sample kernel module:
```
tar xzf hello_module.tar.gz
```

To build the sample module:
```
cd hello_module/
make
```

To remove a previously installed the module:
```
sudo rmmod ./helloModule.ko
```

To install a newly built module:
```
sudo insmod ./helloModule.ko
```

This sample kernel module prints messages to the kernel logs.
The "`dmesg`" command provides a command to interface with kernel log messages, but it is simple enough to just trace the output using:

```
sudo tail –fn 50 /var/log/messages
```

***** THE KERNEL MODULE SHOULD BE RENAMED TO "procReport" *****
**FAILURE TO RENAME THE MODULE WILL RESULT IN A 10 point deduction.**

The kernel module should produce output as in the example descried above.

**IF SOME FUNCTIONALITY IS MISSING IN YOUR KERNEL MODULE, PLEASE FOLLOW THE OUTPUT FORMAT AND USE PLACEHOLDERS.**

To support development, it may be helpful to begin by writing code that sends output to the system log files using printk, and then later, go back and refactor to send output to the proc file. The proc file has been deemphasized in importance for this assignment.

Here are some references describing how to create the proc file kernel module interface:

http://www.crashcourse.ca/introduction-linux-kernel-programming/lesson-11-adding-proc-files-your-modules-part-1
http://stackoverflow.com/questions/8516021/proc-create-example-for-kernel-module/

Kernel modules should have a name in the /proc directory.
Please name your module: "**`proc_report`**".

## Grading

This assignment will be scored out of 100* points. (100/100)=100% *-adjusted as needed

Rubric:
100 possible points

| Report Toal: | 65 points | |
|---|---|---|
| 10 points | | Output of the PID for processes with PID > 650 |
| 10 points | | Output of the process name for processes with PID > 650 |
| 10 points | Output of the number of contiguous pages for PIDs | |
| | >>> | *5 points for at least one PID* |
| | >>> | *5 points for all PID > 650* |
| 10 points | Output of the number of non-contiguous pages for PIDs | |
| | >>> | *5 points for at least one PID* |
| | >>> | *5 points for all PID > 650* |
| 10 points | Output of the number of total pages for PIDs | |
| | >>> | *5 points for at least one PID* |
| | >>> | *5 points for all PID > 650* |
| 15 points | Output the total: # of contiguous pages, # of non-contiguous pages, and # of pages for all processes with PID > 650 | |
| | >>> | *5 points for total # of contiguous pages* |
| | >>> | *5 points for total # of non-contiguous pages* |
| | >>> | *5 points for total pages* |

| Output Total: | 10 points | |
|---|---|---|
| 10 points | Report output uses the Linux /proc | |
| | >>> | *5 points - decoupling output routines from report generation* |

| Miscellaneous: | 25 points |
|---|---|
| 10 points | Kernel module builds, installs, uninstalls |
| 5 points | Kernel module does not crash computer |
| 5 points | Coding style, formatting, and comments |
| 5 points | Following the Output requirements as described (even with missing output) |

## What to Submit

For this assignment, submit a tar gzip archive as a single file upload to Canvas.

Tar archive files can be created by going back one directory from the kernel module code with "`cd ..`", then issue the command "`tar czf hello_module.tar.gz hello_module`". Name the file

the same as the directory where the kernel module was developed but with ".tar.gz" appended at the end: `tar czf <module_dir>.tar.gz <module_dir>`.

**Please rename modules to something other than hello_module.**
To rename a directory in Linux use: "`mv hello_module my_proc_module`".

## Pair Programming (optional)

O*ptionally*, this programming assignment can be completed with two person teams.

If choosing to work in pairs, ***only one*** person should submit the team's tar gzip archive to Canvas.

Additionally, ***EACH*** member of a pair programming team must write a summary of their contributions and their teammate's contributions to the overall project. This write up must be written and submitted INDEPENDENTLY by each team member and describe each person's perspective of the teamwork and outcome of the programming assignment. Summaries must describe group contributions in providing functionality for phases 1-4, and requirements 1-4. **Summaries should be a minimum of 200 words.** Contribution summaries should be submitted in confidence to Canvas as a PDF file named: "pair_contributions.pdf". Google docs and modern versions of MS Word provide the ability to save or export a document in PDF format.

Here is an example write up:

1. Jane Smith

   Jane contributed by writing requirement R1 and requirement R2 in the program. Jane also researched several APIs on the web, and was crucial at designing the algorithm to sum matrices. Jane's ideas supported the design of the program output. Jane also discovered and helped code process API routines. Jane was instrumental in testing for deadlock and fixing bugs related to program correctness.

2. John Doe

   John contributed by helping Jane correct a few errors with the implementation of requirement R1 and requirement R2. Additionally, John wrote most of requirement R3. John researched how the Linux system PThread API works for synchronizing threads and exchanging data. John investigated the use of 3 key routines in the API and deciphered how to use them.

Team members may not share their write ups, but should submit them independently in Canvas as a PDF file. Failure of one or both members to submit the contributions report of the required length will result in both members receiving NO GRADE on the assignment… (*considered late until both are submitted*)

<u>Disclaimer regarding pair programming:</u>
The purpose of TCSS 422 is for everyone to gain experience programming in C while working with on projects related to operating systems and parallel programming. Pair programming is provided as an opportunity to harness teamwork to tackle programming challenges. But this does not mean that teams consist of one champion programmer, and a second observer simply watching the champion! The tasks and challenges should be shared as equally as possible.