Part 4:

Basic terminology:

WSS: WSS is the estimate of the pages those are frequently accessed by a process. RSS signifies the number of pages present in the memory, which doesn't give an indication on how many of pages has been accessed very frequently. WSS gives an estimate on how many number of pages has been accessed by the process during a particular time window.

To check this we try to load a user process which try to access a region in memory. Now as we proceed, after each given time slot, we set the _PAGE_PROTNONE bit to 1 for all the pages accessed by that process and reset the _PAGE_PRESENT bit. Hence after a time window, we are forcing the user process to invoke a page fault for all the page accesses it made after that time frame. The page fault will only occur for the first access to the page.

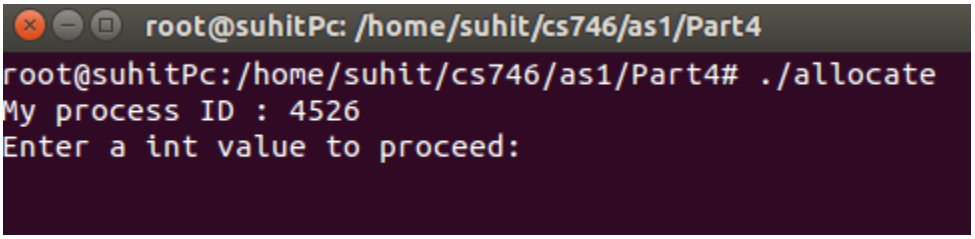
Now we want to estimate the number of page fault in each time slots for that process. It is the WSS for that process, as WSS estimates the frequently accessed pages of a process. If the process tries to access a region frequently, then after each time slot, a loadable kernel module will set the _PAGE_PROTNONE bit to 1 and _PAGE_PRESENT to 0. Now when after that time slots, the process accesses one of the pages, a fore page fault occurs. Then we change the _PAGE_PROTNONE bit to 0 and _PAGE_PRESENT bit to 1. Hence until the next time slot, this page will not be page faulted forcefully.

Pseudo Code:

- Run a user process which tries to access a memory continuously.
- Hook in function **_do_page_fault()** in Linux/arch/x86/mm/fault.c
- This hook function will be invoked whenever a page fault incurs.
- This hook checks whether the _PAGE_PROTNONE bit is set to 1 and _PAGE_PRESENT is set to 0.
- If the above condition is true and the process is with user process pid, then it increases the count of WSS by 1 and change the _PAGE_PROTNONE bit to 0 and _PAGE_PRESENT bit to 1.
- Perform the above process for multiple time slots.

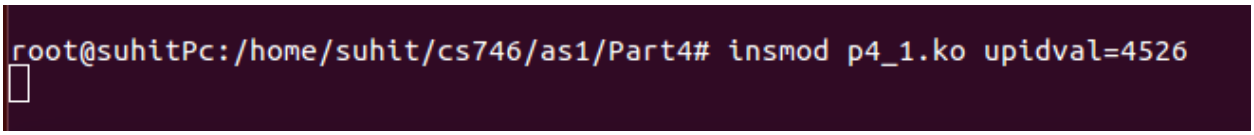
Results and Experiment:

Run the user process:

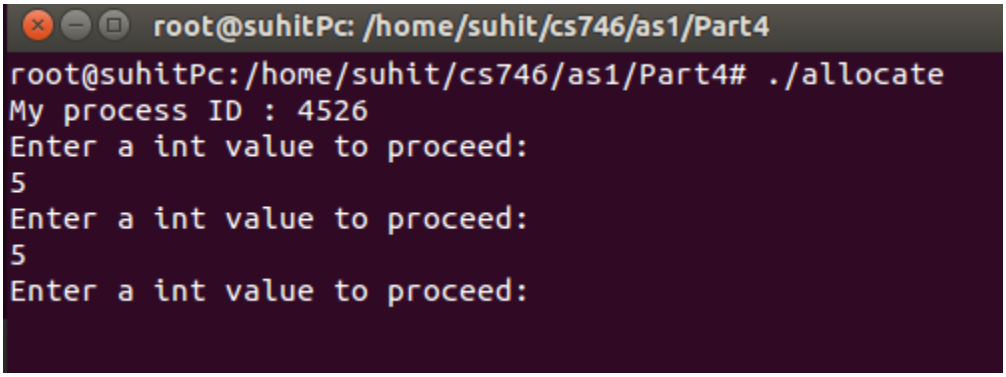


User process will access after each input stroke by user.

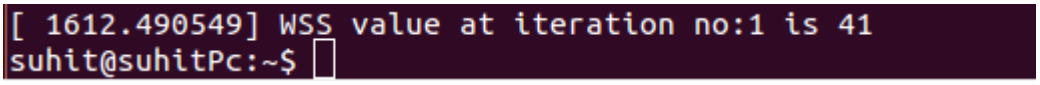
Start the loadable module with the user process PID:



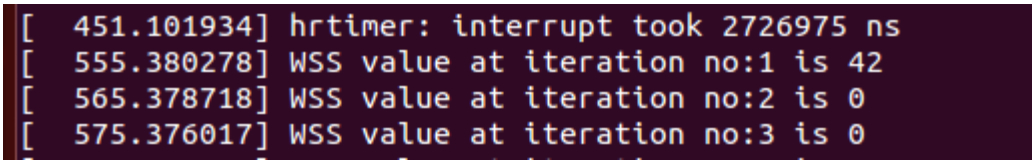
Iterate the user process a number of times:



The WSS value for 1st time interval:



Now next few time interval we didn't iterate the user process. Hence, there is no memory access. WSS should be 0:



After some time, we iterate the user process again and we found some WSS values:

```
[ 1210.233321] Exiting module
[ 1612.490549] WSS value at iteration no:1 is 41
[ 1622.487935] WSS value at iteration no:2 is 0
[ 1632.485294] WSS value at iteration no:3 is 0
[ 1642.482788] WSS value at iteration no:4 is 0
[ 1652.480058] WSS value at iteration no:5 is 0
[ 1662.477538] WSS value at iteration no:6 is 0
[ 1672.476101] WSS value at iteration no:7 is 0
[ 1682.473319] WSS value at iteration no:8 is 0
[ 1692.471182] WSS value at iteration no:9 is 0
[ 1702.468457] WSS value at iteration no:10 is 43
```