pOS Project Report

Theme B
Virtual Memory

Group 9

Tapish Raniwal 120050023

Ayush Deothia 120050025

Syamantak Naskar 120050016

Pintulal Meena 1205018

Rajesh Roshan Behera 120050079

Naveen Sagar 120050026

Bhupendra Bhuarya 120050040

Introduction :

This will tell us how we implemented virtual memory and one handed clock.

Swap Space :

Swap Space is an important part to implement virtual memory. When the RAM requirement of all process together exceeds the actual physical memory present, then the OS provides the illusion of large memory through swap space. Inactive process are swapped from RAM into this swap space and blocked process are brought in when the RAM is free

Since this project deals with simulation of virtual memory on the Geeki OS, we would choose to allocate a constant swap space on the Sim disk to each process. This can be handled based on the number of processes that are going to be simulated, based on the initial configuration file).

Page Table :

The page table is an auxiliary data structure that helps identify which pages are in the main memory and in which page frames they are located in. It is actually a mapping from the ‘virtual’ address to an actual physical address on the disk or the RAM. A page table is required for each process to check where its pages are allocated memory.

The problem with keeping one page table for each process is that the memory requirement for storing page tables themselves exceeds the current disk capacities. We therefore use multi-level page tables.

Here, a mapping is stored from physical page addresses to virtual page addresses. For this, a single linear shared array suffices. Physical page numbers are not stored.Process IDs are stored, which would be used when a linear search is done to get the physical address mapping of a virtual address. This is time consuming since searching through the whole array for each memory access is expensive.

To solve that problem we use hashed multi-level page table. An additional level, in the form of a hash table is added that maps process IDs and virtual page numbers to page table entries. Chaining will need to be done to resolve collisions. Hash tables search in constant time.

Page Replacement Strategy :

When a page needs to be swapped in, a proper page replacement strategy needs to be used. In this project we will be using the one handed clock algorithm.
In clock algorithm, pages don't have to be constantly pushed to the back of the list. The clock algorithm keeps a circular list of pages in memory, with the "hand" (iterator) pointing to the last examined page frame in the list. When a page fault occurs and no empty frames exist, then the R (referenced) bit is inspected at the hand's location. If R is 0, the new page is put in place of the page the "hand" points to, otherwise the R bit is cleared. Then, the clock hand is incremented and the process is repeated until a page is replaced.

Current Status :

The following things have been done as of today:
• Read and understood the internals of the original Geek OS code
• Finalized the design and decided the work distribution
• Coded the VM implementation
- Init_VM() : buildsb kernel directory and page tables and installs a page_fault_handeller interrupt.
- Init_Paging() : creates swap space and create a bitset

• Implemented Multi-level Page Table

• Page_in() :
- it does page_in operation when a page_fault occurs.

Page_fault_handler()
- it decides whether the page_fault address is valid or not then does the appropriate action.

Changes to existing code :

1. In paging.c - added code for InitVM(), Init_Paging(), Page_Fault_Handler(), Page_in(), Write_To_PagingFile(),Read_to_PagingFile etc.
2. in paging.h - added virtual memory info i.e USER_VM_START, USER_VM_SIZE ,USER_VM_END .
3. in uservm.c - private functions added and given functions are changed.
4. in user.h - added stackLimit and exeSize to userContext struct.
5. in makefile.common - replace userseg.c to uservm.c .

Test-case:

test-cases are added to GeekOS/src/user . there are four test cases example1.c, example2.c,example3.c and example4.c in which no.of page_fault varies.