Report : Easy Assignment 3

Disha (2022CS11118)

Flow of the Swapping Mechanism

1. Initialize Swap System

```
// Initialize swap system
void
swapinit(void) {
  int i;
  initlock(&swaplock, "swap");
  for(i = 0; i < NSWAPSLOTS; i++){
    swapslots[i].page_perm = 0;
    swapslots[i].is_free = 1;
  }
}
```

The swapping mechanism begins with the initialization of the swap system. This creates an array of swap slots, where each slot will hold the state (free or used) and permissions of swapped pages.

2. Allocate a Swap Slot

When the operating system needs to swap out a page, it tries to allocate a free swap slot. This is crucial for storing the page data when evicted from memory.

```
int
allocswapslot(void) {
  int i;
  acquire(&swaplock);

  for(i = 0; i < NSWAPSLOTS; i++) {
    if(swapslots[i].is_free) {
      swapslots[i].is_free = 0;
      release(&swaplock);
      return i;
    }
  }
  release(&swaplock);
  return -1;
}
```

3. Select a Victim Process and Page

When memory is low, the system selects a victim process and tries to find a suitable page to evict based on their usage patterns.

```c
struct proc*
find_victim_proc(void)
{
  struct proc *p;
  struct proc *victim = 0;
  int max_rss = -1;

  acquire(&ptable.lock);
  // Loop through process table to find victim
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state == RUNNABLE || p->state == RUNNING || p->state == SLEEPING){
      if(p->pid >= 1 && (p->rss > max_rss || (p->rss == max_rss && victim && p->pid < victim->pid))){
        max_rss = p->rss;
        victim = p;
      }
    }
  }
  release(&ptable.lock);
  return victim;
}
```

4. Swap Out Page

Once a victim page is found, it is written to the allocated swap slot. The permissions of the page are stored, and the page table entry is updated to mark the page as swapped.

```c
// Try to swap out NPG pages
while(npages_swapped < NPG){
  // Find victim page
  victim_va = find_victim_page(victim_proc);
  if(victim_va == 0) {
    break;
  }
  // Get PTE and permissions
  pte = pageswap_walkpgdir(victim_proc->pgdir, (void*)victim_va, 0);
  if(!pte || !(*pte & PTE_P)) {
    continue;
  }
  perm = PTE_FLAGS(*pte);
```

```c
  // Get physical address
  uint pa = PTE_ADDR(*pte);
  if(pa == 0) {
    continue;
  }
  addr = P2V(pa);
  // Swap out to disk
  slotno = swapout_page(addr, perm);
  if(slotno < 0) {
    continue;
  }
```

5. Read Page from Swap Space on Page Fault

If a page fault occurs for a swapped-out page, the system retrieves it from the swap space, updates page table entries, and maps it back into memory.

```c
case T_PGFLT:
    handle_trap_pgfault(tf);
    break;
```

```
// Get PTE for faulting address
uint page_addr = PGROUNDDOWN(fault_addr);
pte = pageswap_walkpgdir(curproc->pgdir, (void*)page_addr, 0);
if(!pte) {
  // cprintf("ERROR: handle_pgfault: No PTE found for address 0x%x\n", page_addr);
  return -1;
}
// Check if this is a swapped page
if(!(*pte & PTE_SWAPPED)) {
  // cprintf("ERROR: handle_pgfault: Page at 0x%x is not swapped (PTE=0x%x)\n", page_addr, *pte);
  return -1;
}
// Extract slot number from PTE
slotno = (*pte >> 12) & 0x3FF;  // Get bits 12-21 (10 bits) for slot number
```

```
// Free a swap slot
void
freeswapslot(int slotno)
{
  if(slotno < 0 || slotno >= NSWAPSLOTS) {
    cprintf("ERROR: freeswapslot: Invalid slot number %d\n", slotno);
    return;
  }
  acquire(&swaplock);
  // Check if slot is already free
  if(swapslots[slotno].is_free) {
    // cprintf("WARNING: freeswapslot: Slot %d is already free\n", slotno);
    release(&swaplock);
    return;
  }
  swapslots[slotno].is_free = 1;
  swapslots[slotno].page_perm = 0;
  release(&swaplock);
  // cprintf("freeswapslot: Freed swap slot %d\n", slotno);
}
```

6. Check and Adjust Memory Usage

The system continuously monitors free memory pages. If the number of free pages drops below a threshold (TH), it invokes the swapout process.

```
void
check_memory(void)
{
  int free_pages = count_free_pages();
  if(free_pages < TH){
    cprintf("Current Threshold = %d, Swapping %d pages\n", TH, NPG);
    swapout();

    TH = (TH * (100 - BETA)) / 100;
    int new_npg = (NPG * (100 + ALPHA)) / 100;
    if(new_npg > LIMIT)
      new_npg = LIMIT;
    NPG = new_npg;
```

```
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      check_memory();
      mem = kalloc();
      if(mem == 0) {
          cprintf("allocuvm out of memory\n");
          deallocuvm(pgdir, newsz, oldsz);
          return 0;
      }
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_
      cprintf("allocuvm out of memory (2)\n");
```

(In function allocuvm (vm.c))

```
      char *v = P2V(pa);
      kfree(v);
      *pte = 0;
      if(myproc() && myproc()->pgdir == pgdir){
        myproc()->rss--;
      }
    }
    else if((*pte & PTE_SWAPPED) != 0){
      int slotno = (*pte >> 12) & 0x3FF;
      if(slotno >= 0 && slotno < NSWAPSLOTS) {
          freeswapslot(slotno);
      } else {
          cprintf("deallocuvm: Invalid swap slot number: %d for address: %d\n", slotno, a);
      }
      *pte = 0;
    }
  }
```

In deallocuvm (vm.c)

7. Free Swap Slots on Process Exit

When a process exits, all the swap slots it used are freed, ensuring resources are reclaimed properly.

```
void
free_process_swap_slots(void)
{
  pte_t *pte;
  uint va;
  struct proc *curproc = myproc();

  if(curproc == 0) { return;}

  // Look through the page table for swapped pages
  for(va = 0; va < KERNBASE; va += PGSIZE){
    pte = pageswap_walkpgdir(curproc->pgdir, (void*)va, 0);

    if(pte && (*pte & PTE_SWAPPED)){
      int slotno = (*pte >> 12) & 0x3FF;  // Extract slot number

      if(slotno >= 0 && slotno < NSWAPSLOTS) {
        freeswapslot(slotno);
        *pte = 0;  // Clear the PTE
      }
```

ALPHA and BETA testing :

The Role of α (ALPHA)

α controls how aggressively the system increases the number of pages to swap out (NPG) after each swapping operation:

- **Formula**: `new_npg = (NPG * (100 + ALPHA)) / 100`
- **Effect**: Increases NPG by ALPHA percent after each swapping operation
- **Higher values** of α cause the system to more quickly escalate the number of pages swapped out each time memory pressure is detected
- **Lower values** make this escalation more gradual

For example, if α = 25 and current NPG = 4:

- Next NPG = (4 * (100 + 25)) / 100 = 5 pages

The Role of β (BETA)

β controls how the memory threshold (TH) decreases after each swapping operation:

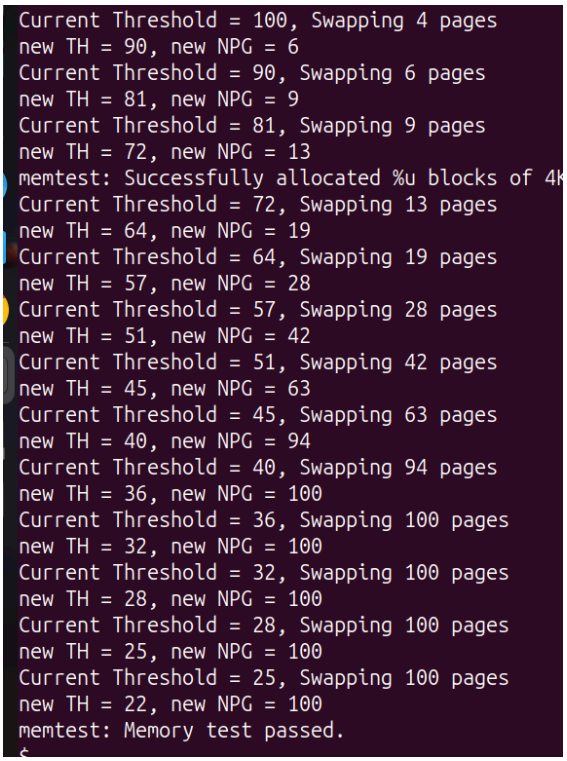
- **Formula**: `TH = (TH * (100 - BETA)) / 100`
- **Effect**: Decreases TH by BETA percent after each swapping operation
- **Higher values** of β cause the threshold to drop more quickly, triggering swapping earlier
- **Lower values** make the threshold reduction more gradual

For example, if β = 10 and current TH = 100:

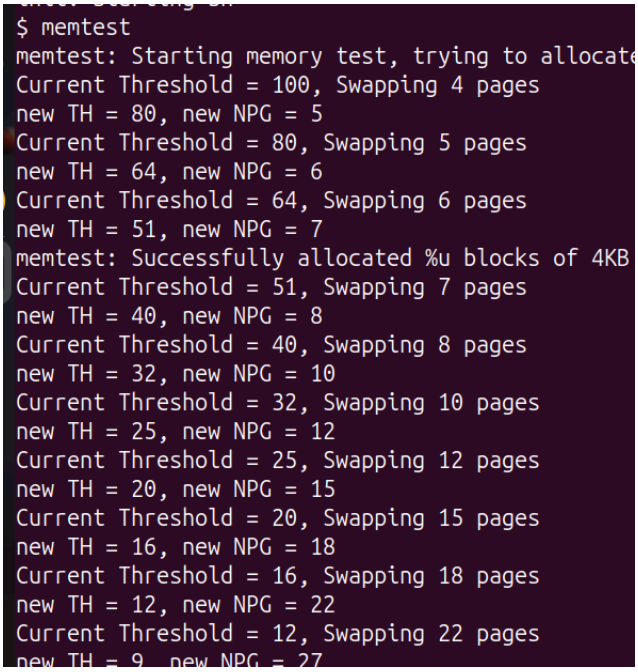
- Next TH = (100 * (100 - 10)) / 100 = 90 pages

```
memtest: Starting memory test, trying to allocate 249036
Current Threshold = 100, Swapping 4 pages
new TH = 90, new NPG = 5
Current Threshold = 90, Swapping 5 pages
new TH = 81, new NPG = 6
Current Threshold = 81, Swapping 6 pages
new TH = 72, new NPG = 7
memtest: Successfully allocated %u blocks of 4KB memory
Current Threshold = 72, Swapping 7 pages
new TH = 64, new NPG = 8
Current Threshold = 64, Swapping 8 pages
new TH = 57, new NPG = 10
Current Threshold = 57, Swapping 10 pages
new TH = 51, new NPG = 12
Current Threshold = 51, Swapping 12 pages
new TH = 45, new NPG = 15
Current Threshold = 45, Swapping 15 pages
new TH = 40, new NPG = 18
Current Threshold = 40, Swapping 18 pages
new TH = 36, new NPG = 22
```

ALPHA = 25, BETA = 10, NPG = 4



ALPHA = 15, BETA = 10, NPG = 4



ALPHA = 25, BETA = 20, NPG = 4

α and β Together Determine System Efficiency: The balance between α and β determines the overall behavior and efficiency of the system:

Response to Memory Pressure

- **High α, Low β**: System responds by aggressively increasing the number of pages swapped out but waits longer between swap operations. This leads to fewer, more intensive swap operations.
- **Low α, High β**: System swaps fewer pages at a time but triggers swapping operations more frequently. This leads to more frequent, less intensive swap operations.
- **High α, High β**: Very aggressive response - quickly increases swapping intensity and frequency. Can lead to "thrashing" where the system spends too much time swapping.
- **Low α, Low β**: Most conservative approach - gradual increase in both swapping intensity and frequency.