CSL 301 - Operating System
XV6-public Lab: TLB and Page Fault Measurement

Objective

Understand TLB behavior, page faults, and lazy allocation in `xv6-public` by modifying kernel and user code. You will build incrementally with guided steps.

Part 1: Add a Page Fault Counter in `proc` and Create Syscall `getpagefaults()`

Step 1: Add a counter field `page_faults` to `struct proc`. File to modify: `proc.h`

```
// Add an integer field in struct proc named page_faults.
// #WRITE YOUR CODE HERE
```

Step 2: Initialize it in `allocproc() (proc.c)`. Write the line that sets page faults counter initially to zero.

```
// #WRITE YOUR CODE HERE
```

Step 3: Implement the syscall to fetch page faults count.

- Assign a syscall number for `getpagefaults` in `syscall.h`.
- Declare the syscall handler in `sysproc.c`. You may use the following hint to implement it:

```
int sys_getpagefaults(void) {
        // Hint: return the page_faults field of current process.
        // #WRITE YOUR CODE HERE
}
```

- Register your syscall in `syscall.c` and add user-space prototypes in `user.h`.
- Add the syscall stub in `usys.S`.

Part 2: Lazy Page Allocation

Step 1: Modify the `vmfault` function to allocate pages lazily. The full code is provided below; study it carefully and understand its working:

```
   int vmfault(pde_t *pgdir, uint va, int write) {
           struct proc *p = myproc();
           char *mem;
           if (va >= p->sz)
           return -1;
           va = PGROUNDDOWN(va);
           if (walkpgdir(pgdir, (void *)va, 0))
           return 0;
           mem = kalloc();
           if (mem == 0)
           return -1;
           memset(mem, 0, PGSIZE);
           if (mappages(pgdir, (void *)va, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0)
{
                   kfree(mem);
                   return -1;
           }
           return 0;
   }
```

Step 2: Modify the trap handler to increase page faults count and handle faults by calling `vmfault()`. Replace the relevant code section with:

```
        // Inside trap() for page faults
        p->page_faults++;
        if (vmfault(p->pgdir, rcr2(), tf->err & 2) < 0)
        p->killed = 1;
```

Part 3: User Programs to Measure Page Faults

Task: Complete the user program `tlbrun.c` to: - allocate increasing number of pages - access pages many times to generate faults - print page counts, trials, ticks, and faults

Fill the missing code marked below:

```
    #include "types.h"
    #include "user.h"

    #define PAGESIZE 4096
    #define MAXPAGES 1024

    int main() {
            int jump = PAGESIZE/sizeof(int);
            printf(1, ‘‘PageCount\tTrials\tTicks\tPageFaults\n’’);
            for (int numpages = 1; numpages <= MAXPAGES; numpages *= 2) {
                    int trials = 5000000;
                    int faults_before = getpagefaults();
                    int start = uptime();
                    int *arr = (int*) sbrk(numpages * PAGESIZE);
                    if (arr == (void*) -1)
                    exit();
                    for (int t = 0; t < trials; t++) {
                            for (int i = 0; i < (numpages/2)*jump; i += jump) {
                                    // #WRITE YOUR CODE HERE: Access the page
to trigger faults
                            }
                    }
                    int end = uptime();
                    int faults_after = getpagefaults();
                    printf(1, ‘‘%d\t%d\t%d\t%d\n’’, numpages, trials, end-start
, faults_after - faults_before);
            }
            exit();
    }
```

Similarly, complete `tlbtest.c` to accept page count and trials from command line and print results. Use the above as guide.

Part 4: Integration and Testing

Add the programs `_tlbrun` and `_tlbtest` to the `UPROGS` in Makefile. Build and run tests.

Submission Checklist

- Kernel source changes with comments.
- User programs (`tlbrun.c`, `tlbtest.c`) with explanations of your approach.
- Sample output demonstrating page fault counts and timing.