

# CSL301

## 12340220

### Assignment: TLB and Page Fault Measurement in XV6

---

## Part 1 – Preparing the Syscall

### Step 1: Add `page_faults` field in `proc` struct

File: `proc.h`

```
// Per-process state
struct proc {
    .....
    int page_faults;
    .....
};
```

---

### Step 2: Initialize `page_faults` in `allocproc()`

File: `proc.c`

```
// Inside allocproc()
// Initialize page fault counter
p->page_faults = 0;
```

Modify `growproc()` to only bump size:

```
if (n > 0){
    sz += n;
}
else if (n < 0){
    sz = deallocvm(curproc->pgdir, sz, sz + n);
}
```

---

### Step 3: Update `syscall.h`

File: `syscall.h`

```
#define SYS_getpagefaults 31
```

---

## Step 4: Update **usys.S**

File: **usys.S**

```
SYSCALL(getpagefaults)
```

---

## Step 5: Add syscall implementation

File: **sysproc.c**

```
int
sys_getpagefaults(void)
{
    struct proc *p = myproc();
    return p->page_faults;
}
```

---

## Step 6: Add declaration in **user.h**

File: **user.h**

```
int getpagefaults(void);
```

---

## Step 7: Update syscall table

File: **syscall.c**

```
extern int sys_getpagefaults(void);
```

```
static int (*syscalls[]) (void) = {
    ....
    [SYS_getpagefaults] sys_getpagefaults,
};
```

---

# Part 2 – Lazy Allocation (VM Modification)

## Step 1: Add prototype in `defs.h`

File: `defs.h`

```
int vmfault(pde_t *pgdir, int va, int write);
```

---

## Step 2: Implement `vmfault()`

```
int vmfault(pde_t *pgdir, int va, int write)
{
    struct proc *p = myproc();
    void *mem;
    if (va >= p->sz)
        return -1;
    va = PGROUNDDOWN(va); // already mapped? then nothing to do
    pte_t *pte = walkpgdir(pgdir, (char *)va, 0);
    if (pte && (*pte & PTE_P))
        return 0; // already mapped

    mem = kalloc();
    if (mem == 0)
        return -1;
    memset(mem, 0, PGSIZE);
    if (mappages(pgdir, (char *)va, PGSIZE, V2P(mem),
                PTE_W | PTE_U) < 0)
    {
        kfree(mem);
        return -1;
    }
    return 0;
}
```

---

# Part 3 – Modify Page Fault Handling

## Step 1: Update `trap.c` in `usertrap()`

File: `trap.c`

```
// Add a new case for pagefault
case T_PGFLT:
    // page fault in user process
    struct proc *p = myproc();
    if (p) {
```

```

uint faultaddr = rcr2();
int is_write = (tf->err & 0x2) ? 1 : 0;
p->page_faults++;

if(vmfault(p->pgdir, faultaddr, is_write) == 0){
    return;
} else {
    p->killed = 1;
}
}
break;

```

---

## Part 4 – User Programs

### Step 1: **tlbrun.c**

File: **tlbrun.c**

```

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

#define PAGE_SIZE 4096
#define MAXPAGES 1024

int
main(int argc, char *argv[])
{
    int jump = PAGE_SIZE / sizeof(int);

    for (int numpages = 1; numpages <= MAXPAGES; numpages *= 2) {
        int trials = 5000000;

        int faults_before = getpagefaults();
        uint start = uptime();

        int *arr = (int *) sbrk(numpages * PAGE_SIZE);
        if (arr == (void *) -1) {
            printf(1, "sbrk failed for %d pages\n", numpages);
            exit();
        }

        for (int t = 0; t < trials; t++) {
            for (int i = 0; i < (numpages/2) * jump; i += jump) {
                arr[i] += 1;
            }
        }
    }
}

```

```

    }
}

uint end = uptime();
int faults_after = getpagefaults();

printf(1, "PageCount: %d    Trials: %d    Ticks: %d    PageFaults:
%d\n",
    numpages, trials, end - start,
    faults_after - faults_before);
}

exit();
}

```

---

## Step 2: **tlbtest.c**

File: **tlbtest.c**

```

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

#define PAGESIZE 4096
#define MAXPAGES 1024

int
main(int argc, char *argv[])
{
    if (argc < 3) {
        printf(1, "Usage: tlbtest <numpages> <trials>\n");
        exit();
    }

    int numpages = atoi(argv[1]);
    int trials = atoi(argv[2]);

    if (numpages < 1 || numpages > MAXPAGES) {
        printf(1, "numpages out of range (1..%d)\n", MAXPAGES);
        exit();
    }
}

```

```

int jump = PAGE_SIZE / sizeof(int);

// allocate
int *arr = (int *) sbrk(numpages * PAGE_SIZE);
if (arr == (void*) -1) {
    printf(1, "sbrk failed for %d pages\n", numpages);
    exit();
}

int pf_before = getpagefaults();
uint start_ticks = uptime();

// trigger lazy allocation
for (int t = 0; t < trials; t++) {
    for (int i = 0; i < (numpages/2) * jump; i += jump) {
        arr[i] += 1;
    }
}

uint end_ticks = uptime();
int pf_after = getpagefaults();

printf(1, "Pages: %d    Trials: %d    Ticks: %d    PageFaults: %d\n",
        numpages, trials, end_ticks - start_ticks,
        pf_after - pf_before);

exit();
}

```

---

## Step 3: Update Makefile

**File:** Makefile

```

UPROGS=\
    ....
    _tlbrun\
    _tlbtest\

```

For tlbrun

```
Machine View

Booting from Hard Disk...
cpu0: starting 0

sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
12340220$ 12340220$ tlbttest 16 5000000
Pages: 16      Trials: 5000000      Ticks: 22      PageFaults: 8
12340220$ tlbrun
PageCount: 1    Trials: 5000000      Ticks: 1      PageFaults: 0
PageCount: 2    Trials: 5000000      Ticks: 1      PageFaults: 1
PageCount: 4    Trials: 5000000      Ticks: 3      PageFaults: 2
PageCount: 8    Trials: 5000000      Ticks: 4      PageFaults: 4
PageCount: 16   Trials: 5000000      Ticks: 7      PageFaults: 8
PageCount: 32   Trials: 5000000      Ticks: 24     PageFaults: 16
PageCount: 64   Trials: 5000000      Ticks: 94     PageFaults: 32
PageCount: 128  Trials: 5000000      Ticks: 187    PageFaults: 64
PageCount: 256  Trials: 5000000      Ticks: 376    PageFaults: 128
PageCount: 512  Trials: 5000000      Ticks: 757    PageFaults: 256
PageCount: 1024 Trials: 5000000      Ticks: 2349   PageFaults: 512
12340220$ tlbttest 16 5000000
Pages: 16      Trials: 5000000      Ticks: 7      PageFaults: 8
12340220$ _
```