Displaying Memory Information

OS Lab UG TAs

December 14, 2024

1 `numvp()`

To count the number of virtual pages in the user part of the address space, we follow these steps:

1. Find the size of the process (given by `myproc()->sz`).
2. Round that size up to the nearest page boundary (using `PGROUNDUP`).
3. Divide the result by the page size (using `PGSIZE`).

Thus, the final formula for the total number of virtual pages is:

$$\text{Number of Virtual Pages} = \frac{\texttt{PGROUNDUP}(\texttt{myproc}()->\texttt{sz})}{\texttt{PGSIZE}}$$

2 `numpp()`

To find the number of physical frames used by the user part of the address space, we need to perform a **page table walk** over the entire user address space. This process involves:

1. Iterating over the range from 0 to `PGROUNDUP(myproc()->sz)` with a stride of `PGSIZE`.
2. Obtaining the **page table entry (PTE)** of each page, which can be done using:

   ```
   walkpgdir(myproc()->pgdir, (char*)page, 0)
   ```

3. Checking the following conditions for each PTE:
   - The PTE exists: `pte` $\neq 0$.
   - The page is present: (`*pte` & `PTE_P`) $\neq 0$.
   - The page is mapped to userspace: (`*pte` & `PTE_U`) $\neq 0$.
4. Incrementing a counter if all the above conditions are satisfied.

This counter will represent the number of physical frames used by the user part of the address space.

3 Files to Change and Their Modifications

3.1 `user.h`

We define our syscalls in `user.h`. Both `numvp()` and `numpp()` return integers and take no arguments. Add the following lines:

```
1 int numvp();
2 int numpp();
```

3.2 `usys.s`

Add the definitions for syscalls:

```
1 SYSCALL(numvp)
2 SYSCALL(numpp)
```

3.3 syscall.h

Define syscall numbers:

```
#define SYS_numvp 22
#define SYS_numpp 23
```

3.4 syscall.c

We have to declare that our syscalls for `numvp()` and `numpp()` are defined externally. Then, we must map the numbers defined in `syscall.h` to `numvp()` and `numpp()` respectively.

```
extern int sys_numvp();
extern int sys_numpp();

static int (*syscalls[])(void) = {
    ...
    [SYS_numvp] sys_numvp,
    [SYS_numpp] sys_numpp,
};
```

3.5 sysproc.c

For `numvp()`, you can define the syscall itself here. Simply return:

$$\frac{\texttt{PGROUNDUP(myproc()->sz)}}{\texttt{PGSIZE}}$$

For `numpp()`, we cannot invoke the `walkpgdir` function directly from inside `sysproc.c`[1], thus we must define a function that will be invoked from `sysproc.c`. We define it in `vm.c`, as then we can invoke `walkpgdir` from inside it. Let us call this new function (defined in `vm.c`) `int getNumPP()`. Now, for `numpp()`, we simply return `getNumPP()`.[2]

3.6 defs.h

Declare the new function:

```
int getNumPP();
```

3.7 vm.c

Implement the function to return the number of physical frames, following the logic in Section 2:

```
int getNumPP() {
    int numPP = 0;
    pte_t* pte;
    for (int i = 0; i < PGROUNDUP(myproc()->sz); i += PGSIZE) {
        pte = walkpgdir(myproc()->pgdir, (char*)i, 0);
        if (pte == 0) {
            continue;
        } else if (*pte & PTE_P && *pte & PTE_U) {
            numPP++;
        }
    }
    return numPP;
}
```

[1] There is a way to do this, by modifying `defs.h`, which is also a valid solution of this lab. In fact, we will have to modify `defs.h` anyway.

[2] The provided solution differs slightly here: in the provided solution, `getNumPP()` takes the page directory as an argument. However, this is not necessary as one can access the page directory from inside the `getNumPP()` function directly using `myproc()->pgdir`.