CS450 Operating Systems (Spring 2018)

Josiah Hunt

jhunt5@hawk.iit.edu

A20350987

Mayank Bansal

mbansal5@hawk.iit.edu

A20392482

Programming Assignment 3

Part 2 (System call to display memory usage of a process)

MANUAL

NAME: myMemory() – get memory usage for each process

SYNOPSIS

```
#include “user.h”
int myMemory(void);
```

DESCRIPTION

myMemory() runs through the process tables of processes and then checks the memory usage based on how many pages are being used

RETURN VALUE

0

ERRORS

None noted

NOTES

This implementation is inefficient because as the number of process increase, the number of page table walks will increase. Each process has to loop through a worse case of 1,048,576 entries. This will cause a lot of overhead for highly multi-threaded processes.

BUGS

None noted

DESIGN & IMPLEMENTATION

The basic program is in the sysfile.c file

```
int myMemory(void) {

   struct proc *curproc = myproc();
   pte_t *pgtab;
   pde_t *pgdir = curproc->pgdir;

   int countPTEUse = 0;
   int countPTEPre = 0;
   int countPTEAcc = 0;

   for (int i = 0; i < NPDENTRIES; i++) {
      pgtab = (pte_t *) P2V(PTE_ADDR(pgdir[i]));

      if (pgdir[i] & PTE_U && pgdir[i] & PTE_P && pgdir[i] & PTE_W)
{

         for (int j = 0; j < NPTENTRIES; j++) {
            // PTE Present
            if (pgtab[j] & PTE_P) {
               countPTEPre++;
            }
            // PTE User Accessible
            if (pgtab[j] & PTE_P && pgtab[j] & PTE_U) {
               countPTEAcc++;
            }
            // PTE User Writable
            if (pgtab[j] & PTE_P && pgtab[j] & PTE_U && pgtab[j] &
PTE_W ) {
               countPTEUse++;
            }
         }
      }
   }

   cprintf("\n     **********************************\n");
   cprintf(  "     *  MEMORY USAGE (In Pages)       \n");
   cprintf(  "     **********************************\n");
   cprintf(  "     * - User Writable   :%d     \n", countPTEUse);
   cprintf(  "     * - Present         :%d     \n", countPTEPre);
   cprintf(  "     * - User Accessible :%d     \n", countPTEAcc);
   cprintf(  "     **********************************\n");
```

```
   return 0;
}
```

Here we initialize all all the initial counts of the page table entries as 0

```
int countPTEUse = 0;
int countPTEPre = 0;
int countPTEAcc = 0;
```

We get the current process using

```
struct proc *curproc = myproc();
pte_t *pgtab;
pde_t *pgdir = curproc->pgdir;
```

Now that we have the page directory for this individual process, we can then parse the page table directory for this individual process. This allows us to have multiple levels of processes and have the memory usage available for all of them.

```
for (int i = 0; i < NPDENTRIES; i++) {
   pgtab = (pte_t *) P2V(PTE_ADDR(pgdir[i]));

       // parse table

}
```

Now we go through the whole page table directory, and get the address of page table each index points to.

```
if (pgdir[i] & PTE_U && pgdir[i] & PTE_P && pgdir[i] & PTE_W) {

   }
```

here we check only if the pgdir entry is valid, because the ones that aren't will point to an empty page table.

Now, for each page table

```
for (int j = 0; j < NPTENTRIES; j++) {
   // PTE Present
   if (pgtab[j] & PTE_P) {
      countPTEPre++;
   }
   // PTE User Accessible
   if (pgtab[j] & PTE_P && pgtab[j] & PTE_U) {
      countPTEAcc++;
   }
   // PTE User Writable
   if (pgtab[j] & PTE_P && pgtab[j] & PTE_U && pgtab[j] & PTE_W ) {
      countPTEUse++;
   }
}
```

we loop through all the page table entries to find ones that are present, user writable and user accessible.

We use the

```
PTE_U
PTE_P
PTE_W
```

Flags to determine which bits are set in the address

The system call then pretty prints the results that it finds:

```
cprintf("\n     **********************************\n");
cprintf(  "     *  MEMORY USAGE (In Pages)       \n");
cprintf(  "     **********************************\n");
cprintf(  "     * - User Writable   :%d     \n", countPTEUse);
cprintf(  "     * - Present         :%d     \n", countPTEPre);
cprintf(  "     * - User Accessible :%d     \n", countPTEAcc);
cprintf(  "     **********************************\n");
```

Note: changes made to xv6 for implementing syscall haven't been mentioned