

# CSIS0230A Principles of Operating Systems (Class A)

## Tutorial 7

### A look at the Page Table

In this tutorial, you are required to add a system call, defined as follows:

```
int getpt(pid_t pid, unsigned long addr, char *buf);
```

Given a specific process *pid* and a linear (i.e., virtual) address *addr*, fill *buf* with the page table of *pid* that contains the address *addr*.

#### Input:

*pid*—the specified process ID  
*addr*—the specified linear address  
*buf*—the buffer provided by user process to store the page table. Its size should be `PAGE_SIZE = 4KiB`.

#### Return:

–*ESRCH*—process can not be found.  
–*EFAULT*—memory specified by *buf* is not writable.  
1—the page table containing the linear address *addr* is not in memory.  
0—success

The following program tests your system call (already in the computer, compiled). Understand it and run it using the PID of another running process as argument. By inspecting the content of `/proc/pid/maps` (where *pid* is the process id of the running process; run `ps a` to see a full list of them), try to get the page table of some valid and invalid addresses. Explain what is meant by the numbers printed by the program. Also try to use an address around `c0000000`. Explain what are meant by the numbers shown.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <syscall.h>
#include <sys/types.h>

#define __NR_getpt 259
__syscall3(int, getpt, pid_t, p, unsigned long, laddr, int *, buf);

void dump(int buf[1024]) {
    int i, j;
    for (i = 0; i < 1024; i += 8) {
        if (i > 0 && i % (16 * 8) == 0) {
            printf("Press enter");
            while (getchar() != '\n'); /* skip line */
        }
        printf("%04x: ", i);
        for (j = 0; j < 8; ++j)
            printf(" %8x", buf[i+j]);
        printf("\n");
    }
}
```

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s pid\n", argv[0]);
        return 1;
    }
    pid_t pid = atoi(argv[1]);
    for (;;) {
        unsigned long addr;
        printf("Address in hex? (-1 ends) ");
        if (scanf("%lx", &addr) != 1 || addr == (unsigned long)-1)
            return 0;
        while (getchar() != '\n'); /* skip line */
        int buf[1024];
        switch (getpt(pid, addr, buf)) {
            case -1:
                fprintf(stderr, "Error: %s\n", strerror(errno));
                return 1;
            case 1:
                printf("Page not present.\n");
                break;
            case 0:
                dump(buf);
        }
    }
}
```

**Remark:** A typical `/proc/NNN/map` file looks as follows:

```
08048000-0804f000 r-xp 00000000 03:03 318762 /sbin/init
0804f000-08050000 rw-p 00006000 03:03 318762 /sbin/init
08050000-08051000 rwxp 00000000 00:00 0
40000000-40011000 r-xp 00000000 03:03 385131 /lib/ld-2.3.1.so
40011000-40012000 rw-p 00011000 03:03 385131 /lib/ld-2.3.1.so
4001d000-40126000 r-xp 00000000 03:03 385134 /lib/libc-2.3.1.so
40126000-4012b000 rw-p 00109000 03:03 385134 /lib/libc-2.3.1.so
4012b000-40131000 rw-p 00000000 00:00 0
bffff000-c0000000 rwxp 00000000 00:00 0
```

Each row represent a region of virtual memory used by the process. The meaning of the columns are: the starting and ending (virtual) address of the memory region, whether the region can be read, written and executed, and whether other processes share the memory with it. The fields after that is meaningful only if the region maps to a disk file. In that case they are the starting offset of the file which maps to the region, the major device number, minor device number and the inode number of the file mapped (which uniquely identify a file), and the filename of the file mapped.