Deimantas Gilys #A20434583
Francisco Barba Cuellar #A20121767
Nathan Cook #A20458336

Adding whereIs, isWritable, notWritable, yesWritable

All common attributes

1. Add entries in kernel > syscall.h
   a.
```
23  #define SYS_whereIs 22
24  #define SYS_isWritable 23
25  #define SYS_notWritable 24
26  #define SYS_yesWritable 25
```
2. Add entries in user > usys.S
   a.
```
xv6-riscv > user > usys.S
  1   # generated by usys.pl -
  2   #include "kernel/syscall
  3   .global whereIs
  4   whereIs:
  5    li a7, SYS_whereIs
  6    ecall
  7    ret
  8   .global isWritable
  9   isWritable:
 10    li a7, SYS_isWritable
 11    ecall
 12    ret
 13   .global notWritable
 14   notWritable:
 15    li a7, SYS_notWritable
 16    ecall
 17    ret
 18   .global yesWritable
 19   yesWritable:
 20    li a7, SYS_yesWritable
 21    ecall
 22    ret
```
3. Add entries in user > usys.pl
   a.
```
39  entry("alsoNice");
40  entry("whereIs");
41  entry("isWritable");
42  entry("notWritable");
43  entry("yesWritable");
```
4. Add definition in kernel > syscall.c

Deimantas Gilys #A20434583
Francisco Barba Cuellar #A20121767
Nathan Cook #A20458336

a.
```
105  extern uint64 sys_whereIs(void);
106  extern uint64 sys_isWritable(void);
107  extern uint64 sys_notWritable(void);
108  extern uint64 sys_yesWritable(void);
```

5. Add entries in kernel > sycall.c syscalls[]

a.
```
135  [SYS_whereIs] sys_whereIs,
136  [SYS_isWritable]  sys_isWritable,
137  [SYS_notWritable] sys_notWritable,
138  [SYS_yesWritable] sys_yesWritable
```

6. Add entries in user > user.h

a.
```
25  int whereIs(void*);
26  int isWritable(void*);
27  int notWritable(void*);
28  int yesWritable(void*);
```

7. Add entries to kernel > defs.h in syscall.c section

a.
```
136  // syscall.c
137  int             whereIs(void*);
138  int             isWritable(void*);
139  int             notWritable(void*);
140  int             yesWritable(void*);
141  void            argint(int, int*);
142  int             argstr(int, char*, int);
143  void            argaddr(int, uint64 *);
144  int             fetchstr(uint64, char*, int);
145  int             fetchaddr(uint64, uint64*);
146  void            syscall();
147
```

Deimantas Gilys #A20434583
Francisco Barba Cuellar #A20121767
Nathan Cook #A20458336

Implementation of 4 system calls:

1. Implement in sysproc.c
   a.
```
 9  int sys_whereIs(void)
10  {
11    //added
12    char va;
13
14    if(argstr(0, &va, sizeof(char*)) < 0)
15      return -1;
16
17    struct proc* p = myproc();
18
19    pte_t* pte = walk(p->pagetable, (uint64)va, 0);
20
21    if(pte == 0 || (*pte & PTE_V) == 0)
22      return -1;
23
24    return PTE2PA(*pte);
25  }
```
      i. Using PTE2PA, we can fetch the PFN of the given VA
      ii. We first consider the error cases where the provided va size does not match
      iii. We also return -1 when walk cannot find the PTE or if PTE_V is not set

Deimantas Gilys #A20434583
Francisco Barba Cuellar #A20121767
Nathan Cook #A20458336

b.

```
27  int sys_iswritable(void)
28  {
29    char va;
30    struct proc* p = myproc();
31    if(argstr(0, &va, sizeof(char *)) < 0)
32      return -1;
33
34    pte_t* pte = walk(p->pagetable, (uint64)va, 0);
35    if(pte == 0)
36      return -1;
37
38    if((*pte & PTE_W) == 0)
39      return 0;
40    else
41      return 1;
42  }
```

i. isWritable also returns -1 if the PTE cannot be found or VA size does not match. If it passes these conditions, the function returns 0 if the writable flag PTE_W is 0 and 1 otherwise.

c.

```
44  int sys_notwritable(void)
45  {
46    char va;
47    struct proc* p = myproc();
48    if(argstr(0, &va, sizeof(char *)) < 0)
49      return -1;
50
51    pte_t* pte = walk(p->pagetable, (uint64)va, 0);
52    if(pte == 0)
53      return -1;
54
55    if((*pte & PTE_W) == 0)
56      return -1;
57
58    *pte &= ~PTE_W;
59    return 0;
60  }
```

i. We approach this function in a very similar manner, including checking if the address is already not writable. If the cases pass, the function returns 0 and the writable flag PTE_W is updated.

Deimantas Gilys #A20434583
Francisco Barba Cuellar #A20121767
Nathan Cook #A20458336

d.

```
62  int sys_yeswritable(void)
63  {
64    char va;
65    struct proc* p = myproc();
66    if(argstr(0, &va, sizeof(char *)) < 0)
67      return -1;
68
69    pte_t* pte = walk(p->pagetable, (uint64)va, 0);
70    if(pte == 0)
71      return -1;
72
73    if((*pte & PTE_W) != 0)
74      return -1;
75
76    *pte |= PTE_W;
77    return 0;
78  }
```

i. Once again, we check all of the necessary cases, but this time rather than check if the address is not writable, we check if it is already writable. If it is not writable, it will be updated and the PTE_W flag will then be writable and 0 is returned.