```
shiree@shiree-VirtualBox:~$ diff -uw cs450_pa3/xv6/user.h xv6/user.h
--- cs450_pa3/xv6/user.h        2017-11-02 05:06:53.792878422 -0500
+++ xv6/user.h  2017-11-03 22:18:35.577686308 -0500
@@ -23,7 +23,6 @@
 char* sbrk(int);
 int sleep(int);
 int uptime(void);
-int u_v2p(char*);

 // ulib.c
 int stat(char*, struct stat*);
```

Figure 1: user.h u_v2p def diff

```
4   // system calls
5   int fork(void);
6   int exit(void) __attribute__((noreturn));
26  int u_v2p(char*);
```

Figure 2: user.h u_v2p def

Defining u_v2p as a system call, callable by user.

```
shiree@shiree-VirtualBox:~$ diff -uw cs450_pa3/xv6/usys.S xv6/usys.S
--- cs450_pa3/xv6/usys.S        2017-11-02 04:54:46.333892007 -0500
+++ xv6/usys.S  2017-11-03 22:18:35.577686308 -0500
@@ -29,4 +29,3 @@
 SYSCALL(sbrk)
 SYSCALL(sleep)
 SYSCALL(uptime)
-SYSCALL(u_v2p)
```

Figure 3: usys.S u_v2p SYSCALL diff

```
1   #include "syscall.h"
2   #include "traps.h"
3
4   #define SYSCALL(name) \
5     .globl name; \
6     name: \
7       movl $SYS_ ## name, %eax; \
8       int $T_SYSCALL; \
9       ret
10
11  SYSCALL(fork)
12  SYSCALL(exit)
32  SYSCALL(u_v2p)
```

Figure 4: usys.S u_v2p SYSCALL

Adding u_v2p as a system call.

```
shiree@shiree-VirtualBox:~$ diff -uw cs450_pa3/xv6/syscall.h xv6/syscall.h
--- cs450_pa3/xv6/syscall.h     2017-11-02 04:54:46.325892149 -0500
+++ xv6/syscall.h       2017-11-03 22:18:35.577686308 -0500
@@ -20,4 +20,3 @@
 #define SYS_link   19
 #define SYS_mkdir  20
 #define SYS_close  21
-#define SYS_u_v2p  22
```

Figure 5: syscall.h SYS_u_v2p diff

```
1  // System call numbers
2  #define SYS_fork    1
3  #define SYS_exit    2
23 #define SYS_u_v2p  22
```

Figure 6: syscall.h SYS_u_v2p

Defining the system call number.

```
shiree@shiree-VirtualBox:~$ diff -uw cs450_pa3/xv6/syscall.c xv6/syscall.c
--- cs450_pa3/xv6/syscall.c     2017-11-02 04:54:46.325892149 -0500
+++ xv6/syscall.c       2017-11-03 22:18:35.577686308 -0500
@@ -103,7 +103,6 @@
 extern int sys_wait(void);
 extern int sys_write(void);
 extern int sys_uptime(void);
-extern int sys_u_v2p(void);

 static int (*syscalls[])(void) = {
 [SYS_fork]    sys_fork,
@@ -127,7 +126,6 @@
 [SYS_link]    sys_link,
 [SYS_mkdir]   sys_mkdir,
 [SYS_close]   sys_close,
-[SYS_u_v2p]    sys_u_v2p,
 };

 void
```

Figure 7: syscall.c sys_u_v2p diff

```
105  extern int sys_uptime(void);
106  extern int sys_u_v2p(void);
107
108  static int (*syscalls[])(void) = {
109  [SYS_fork]    sys_fork,
110  [SYS_exit]    sys_exit,
130  [SYS_u_v2p] sys_u_v2p,
131  };
```

Figure 8: syscall.c sys_u_v2p

Adding the system call to the list and the associated function.

```
shiree@shiree-VirtualBox:~$ diff -uw cs450_pa3/xv6/sysproc.c xv6/sysproc.c
--- cs450_pa3/xv6/sysproc.c     2017-11-02 06:19:45.150963318 -0500
+++ xv6/sysproc.c       2017-11-03 22:18:35.577686308 -0500
@@ -6,7 +6,6 @@
 #include "memlayout.h"
 #include "mmu.h"
 #include "proc.h"
-#include "stdint.h"

 int
 sys_fork(void)
@@ -90,30 +89,3 @@
   release(&tickslock);
   return xticks;
 }
-
-//HOMEWORK 3 IMPLEMENTATION
-//Takes input using argint, uses v2p to convert to physical, and then returns the physical address
-int
-sys_u_v2p(char *a)
-{
-       int v;
-       //argint(0, &v);//gets syscall arg
-  char *hex;
-  argstr(0,&hex);
-  uint32_t val = 0;
-  while(*hex){
-    uint8_t byte = *hex++;
-    if(byte >= '0'&& byte <= '9') byte = byte - '0';
-    else if (byte >= 'a' && byte <= 'f') byte = byte - 'a' + 10;
-    else if (byte >= 'A' && byte <= 'F') byte = byte - 'A' + 10;
-    val = (val << 4) | (byte & 0xF);
-  }
-  v = val;
-       int p = V2P(v);//converts
-       if (p >= (PHYSTOP) || p < 0) {//end of physical memory
-               cprintf("invalid ");
-               return p;//error
-       }
-       //p = (uint) (p);
-       return p;//returns physical address
-}
```

Figure 9: sysproc.c sys_u_v2p diff

```
94  //HOMEWORK 3 IMPLEMENTATION
95  //Takes input using argint, uses v2p to convert to physical, and then returns the physical
96  int
97  sys_u_v2p(char *a)
98  {
99    int v;
100   //argint(0, &v);//gets syscall arg
101   char *hex;
102   argstr(0,&hex);
103   uint32_t val = 0;
104   while(*hex){
105     uint8_t byte = *hex++;
106     if(byte >= '0'&& byte <= '9') byte = byte - '0';
107     else if (byte >= 'a' && byte <= 'f') byte = byte - 'a' + 10;
108     else if (byte >= 'A' && byte <= 'F') byte = byte - 'A' + 10;
109     val = (val << 4) | (byte & 0xF);
110   }
111   v = val;
112   int p = V2P(v);//converts
113   if (p >= (PHYSTOP) || p < 0) {//end of physical memory
114     cprintf("invalid ");
115     return p;//error
116   }
117   //p = (uint) (p);
118   return p;//returns physical address
119 }
```

Figure 5: sysproc.c sys_u_v2p

Implementation of u_v2p. Takes the syscall argument as a hex encoded string, converts into an integer and passes it as an argument to V2P. The returned value is checked between the bounds of 0 to PHYSTOP, in which if true states invalid. The returned value is the integer of the address.

```c
//TEST FILE FOR HOMEWORK 3 u_v2p
// I am using
//this cat.c which already compiles. Sorry for the confusing
//name but this is where the test for u_v2p is
#include "types.h"
#include "stat.h"
#include "user.h"
#include "stdint.h"

int
main(int argc, char *argv[])
{
  printf(1, "physical address: 0x%x\n", u_v2p(argv[1]));
  exit();
}
```

Figure 11: uv2p.c

Execution of u_v2p, in which takes the argument and returns the hex translated address.