Mini Project 1

The purpose of this first project was to add a system call that counts and outputs the number of times a process calls the getPID( ) system call. To start off we had to edit syscall.h, where all the system calls and their reserved call numbers are. So we had to add our system call at the end of the list of system calls and add the call number 22.
This way it can be accessed from the syscall tables.

```
#define _SYSCALL_H_

// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_write   5
#define SYS_read    6
#define SYS_close   7
#define SYS_kill    8
#define SYS_exec    9
#define SYS_open   10
#define SYS_mknod  11
#define SYS_unlink 12
#define SYS_fstat  13
#define SYS_link   14
#define SYS_mkdir  15
#define SYS_chdir  16
#define SYS_dup    17
#define SYS_getpid 18
#define SYS_sbrk   19
#define SYS_sleep  20
#define SYS_uptime 21
#define SYS_getid  22

#endif // _SYSCALL_H_
```

Then we had to add it into the syscall.c file, where there's an array of pointers. When we add our system call, the pointer will connect to the function we implement. We also had to add a prototype of the function to this file.

```
extern int sys_getid(void);

// array of function pointers to handlers for all the syscalls
static int (*syscalls[])(void) = {
[SYS_chdir]   sys_chdir,
[SYS_close]   sys_close,
[SYS_dup]     sys_dup,
[SYS_exec]    sys_exec,
[SYS_exit]    sys_exit,
[SYS_fork]    sys_fork,
[SYS_fstat]   sys_fstat,
[SYS_getpid]  sys_getpid,
[SYS_kill]    sys_kill,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_mknod]   sys_mknod,
[SYS_open]    sys_open,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_unlink]  sys_unlink,
[SYS_wait]    sys_wait,
[SYS_write]   sys_write,
[SYS_uptime]  sys_uptime,
[SYS_getid]   sys_getid,
```

Then we finally implement the function in sysproc.c. We did this by adding a global variable, i, that gets incremented whenever getpid( ) is called. Then when getid is called, it returns that variable.

```
int i = 0;
int
sys_getpid(void)
{
  i += 1;
  return proc->pid;
}

int
sys_getid(void)
{
  return i;
}
```

.

Then we have to allow our user program to access the new system call, which was done by adding it to the usys.S file.

```
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getid)
```

Then the system call is added to user.h, which is what the user program will call. We're not implementing it here, so instead it's mapped to the specific call number we assigned to it earlier (22).

user.h ×

```
int open(char*, int);
int mknod(char*, short, short);
int unlink(char*);
int fstat(int fd, struct stat*);
int link(char*, char*);
int mkdir(char*);
int chdir(char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int getid(void);
```

Finally we wrote a user program. In this program we just print out a statement and then call our new syscall to print out our number.

proj.c

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"

int
main(int argc, char *argv[])
{
  printf(1, "getpid has run %d\n", getpid());
  exit();
}
```

Now to enable our program to work in qemu. To do this we are going to go into the makefile.mk in the user directory. In here we add our program to the list of visible programs in qemu.

```
        usertests\
        wc\
        zombie\
        proj\

EXTRA=\
mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
proj.c\
printf.c umalloc.c\
README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
.gdbinit.tmpl gdbutil\
```

After running our syscall we will be given the amount of times getPid was called. Every subsequent time out program is run that number will increment by 1.

```
sergio@sergio-VirtualBox:~/Desktop/xv6_patched$ make qemu-nox
Ctrl+a h for help
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2
xv6...
lapicinit: 1 0xfee00000
cpu1: starting
cpu0: starting
init: starting sh
$ proj
getpid has run 3
$ proj
getpid has run 4
$ proj
getpid has run 5
$ proj
getpid has run 6
$ proj
getpid has run 7
$
```

Documentation

Name	Code	Report	Documentation	Presentation
Stefan Smoke	x	x		x
Sergio Torres	x	x		x
Kate Wood		x	x	x

File	Line #	Code	Reason
syscall.h	26	#define SYS_getid 22	Reserve 22 as our system call number
syscall.c	83	extern int sys_getid(void)	Prototype of our function
syscall.c	108	[SYS_getid] sys_getid	Pointer to implementation of our function
sysproc.c	38	int i = 0;	Create global variable.
sysproc.c	42	i += 1;	Increment global variable when getpid( ) is called

sysproc.c	46
	47
	48
	49
	50
		int
		sys_getid(void)
		{
		return i;
		}
			Main function of our system call. Returns the number stored in global variable

usys.S	32	SYSCALL(getid)	Allows user to access new system call
user.h	28	int getid(void);	Allows the user program to connect to the function
makefile.mk	25	proj.c\	Enables program to work in qemu