Paul Kummer

Dr. Hanku Lee

Operating Systems CSIS 430-01

10/13/2021

Implementing hello(str) in xv6

Purpose

This assignment is to modify the implementation of the system call hello. Previously, the system call for hello would only display “Hello World”. How the system call will either print “Hello World” if the second argument is null or whatever the string argument is following the system call. The end goal of this assignment is to have the students create a system call with and argument that very closely mimics the command “echo”. To prove that the new system call works, a user command of testcase will pass string arguments into the new system call.

Code

Much of the code from the previous hello system call is reused or only changed slightly. The hello function now accepts a character pointer as a parameter.

hello.c

This file was changed to call hello two different ways depending on if there is an argument for the hello call. If there is an argument, the argument is printed. Otherwise, the traditional “Hello World” is displayed. The exit function call is required. If it is not used, error 5 will occur.

C: > Users > pakum > Desktop > operatingSystems > assign06 > code > C hello.c

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

int main(int argc, char *argv[])
{
    //int returnVal;

    if(argc > 1)
    {
        hello(argv[1]);
    }
    else
    {
        hello("Hello World");
    }

    exit();
}
```

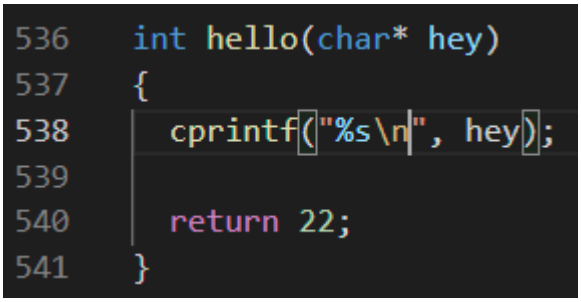
defs.h

The declaration of the hello function for processes was changed to handle a character array as a parameter. Strings will be allowed by using a character array.

```c
//PAGEBREAK: 16
// proc.c
int             cpuid(void);
void            exit(void);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             cps(void);
int             hello(char*);
```

proc.c

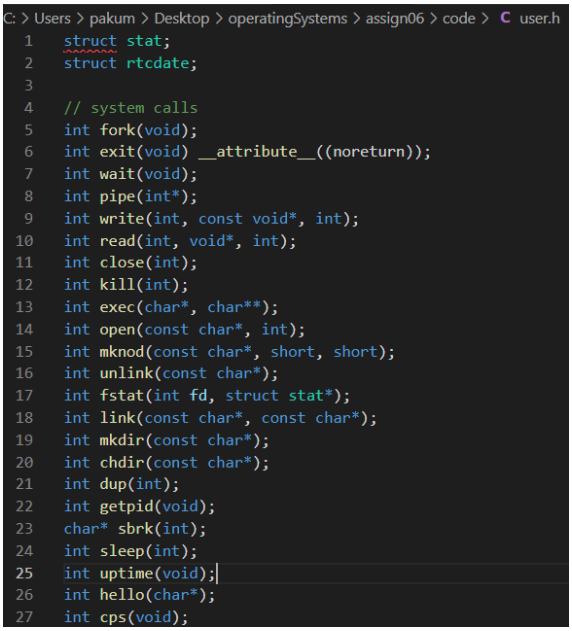
The definition of the hello was added, and now will print the string and add a newline when hello is called. Remember, if no argument is used, then the traditional “Hello World” is displayed.



```
536  int hello(char* hey)
537  {
538    cprintf("%s\n", hey);
539
540    return 22;
541  }
```

user.h

This definition for hello has the character array added as a parameter.



C: > Users > pakum > Desktop > operatingSystems > assign06 > code > C user.h

```
1   struct stat;
2   struct rtcdate;
3
4   // system calls
5   int fork(void);
6   int exit(void) __attribute__((noreturn));
7   int wait(void);
8   int pipe(int*);
9   int write(int, const void*, int);
10  int read(int, void*, int);
11  int close(int);
12  int kill(int);
13  int exec(char*, char**);
14  int open(const char*, int);
15  int mknod(const char*, short, short);
16  int unlink(const char*);
17  int fstat(int fd, struct stat*);
18  int link(const char*, const char*);
19  int mkdir(const char*);
20  int chdir(const char*);
21  int dup(int);
22  int getpid(void);
23  char* sbrk(int);
24  int sleep(int);
25  int uptime(void);
26  int hello(char*);
27  int cps(void);
```

syscall.h

The system call declaration for the function sys_hello does not change.

C: > Users > pakum > Desktop > operatingSystems > assign06 > code > C syscall.h

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_hello  22
#define SYS_cps    23
```

syscall.c

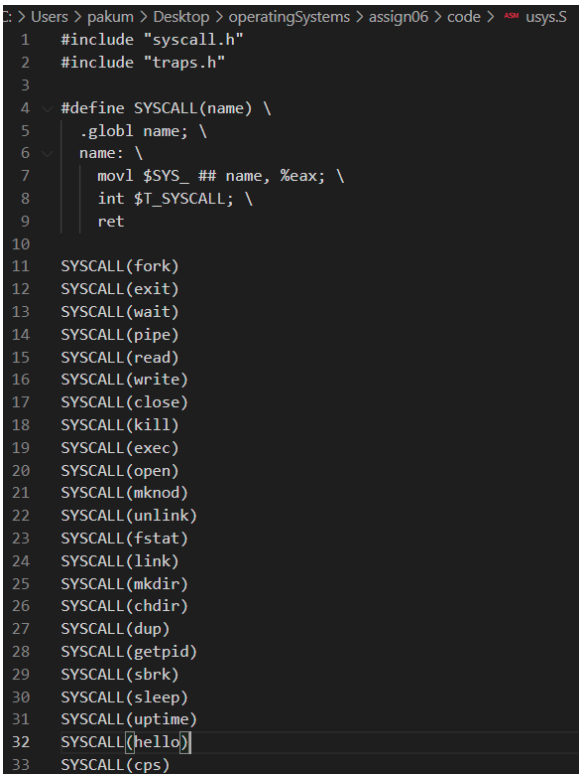
The system call for sys_hello is unchanged from the version that didn't accept arguments. It still just says that the sys_hello is defined elsewhere and is added to the list of system calls.

```
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_hello(void);
extern int sys_cps(void);
```

```
static int (*syscalls[])(void) =
{
  [SYS_fork]    sys_fork,
  [SYS_exit]    sys_exit,
  [SYS_wait]    sys_wait,
  [SYS_pipe]    sys_pipe,
  [SYS_read]    sys_read,
  [SYS_kill]    sys_kill,
  [SYS_exec]    sys_exec,
  [SYS_fstat]   sys_fstat,
  [SYS_chdir]   sys_chdir,
  [SYS_dup]     sys_dup,
  [SYS_getpid]  sys_getpid,
  [SYS_sbrk]    sys_sbrk,
  [SYS_sleep]   sys_sleep,
  [SYS_uptime]  sys_uptime,
  [SYS_open]    sys_open,
  [SYS_write]   sys_write,
  [SYS_mknod]   sys_mknod,
  [SYS_unlink]  sys_unlink,
  [SYS_link]    sys_link,
  [SYS_mkdir]   sys_mkdir,
  [SYS_close]   sys_close,
  [SYS_hello]   sys_hello,
  [SYS_cps]     sys_cps,
};
```

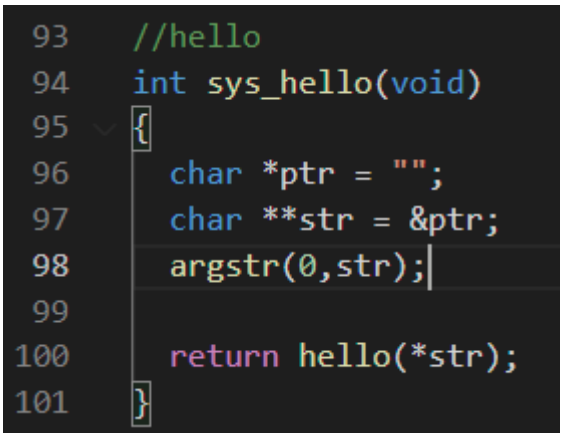
usys.S

the source code file remains unchanged, and still specifies hello as a system call.



sysproc.c

The system process function sys_hello is changed to call hello with a string as a parameter. Additionally, the parameter for the hello function is extracted from a command line argument by using the function argstr. With this function, the argument at index zero will be extracted. Since argstr extracts a pointer to a pointer, the extracted value must be dereferenced, resulting in a pointer to a character array.



Makefile

The makefile is updated to include a new user command called testcase. Otherwise, the other files and commands remains unchanged.

```
168  UPROGS=\
169      _cat\
170      _echo\
171      _forktest\
172      _grep\
173      _init\
174      _kill\
175      _ln\
176      _ls\
177      _mkdir\
178      _rm\
179      _sh\
180      _stressfs\
181      _usertests\
182      _wc\
183      _zombie\
184      _hello\
185      _cp\
186      _ps\
187      _testcase\
```

```
254  EXTRA=\
255      mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
256      ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
257      hello.c cp.c ps.c testcase.c\
258      printf.c umalloc.c\
259      README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
260      .gdbinit.tmpl gdbutil\
```

testcase.c

This file is added to test the newly modified hello function. It will iterate five times and make ten calls to the hello function with two different predefined parameters.

```
: > Users > pakum > Desktop > operatingSystems > assign06 > code > C testcase.c
1   #include "types.h"
2   #include "stat.h"
3   #include "user.h"
4
5   int main(void)
6   {
7       char *strOne = "CSIS430's Xv6 Lab Session";
8       char *strTwo = "Calling hello() with an argument\n";
9
10      for(int x=0; x<5; x++)
11      {
12          hello(strOne);
13          hello(strTwo);
14      }
15
16      exit();
17  }
```

Conclusion

This assignment taught me how system calls can call other functions and pass parameters. It was very useful to learn how to extract command line arguments with the argstr function. The system calls work together with many other parts of the operating system, and are more work to implement, where a user command is very easy to add. I also learned that in some of the c files, I need to end the function with exit() instead of returning the integer zero. I believe that the exit() function is terminating the process correctly and returning zero is now.