

Paul Kummer

Dr. Hanku Lee

Operating Systems CSIS 430-01

11/10/2021

## Lab 08: hello() as a system call with arguments

### Purpose

This assignment is to modify the implementation of the system call hello. Previously, the system call for hello would only display "Hello World". Now the system call will either print "Hello World" if the second argument is null or all the arguments after the command. The end goal of this assignment is to have the students create a system call with an 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 has now been modified to print all the arguments.

*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 or arguments, then all the arguments after the command will be printed. Otherwise hello will be called with "Hello World" if the second argument is null.

```
C: > Users > pakum > Desktop > operatingSystems > lab08 >
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4
5  int main(int argc, char *argv[])
6  {
7      if(argc > 1)
8      {
9          for(int x=1; x<argc; x++)
10         {
11             hello(argv[x]);
12             hello(" ");
13         }
14         hello("\n");
15     }
16     else
17     {
18         hello("Hello World\n");
19     }
20
21     exit();
22 }
```

*defs.h*

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

```

104 //PAGEBREAK: 16
105 // proc.c
106 int      cpuid(void);
107 void     exit(void);
108 int      fork(void);
109 int      growproc(int);
110 int      kill(int);
111 struct cpu* mycpu(void);
112 struct proc* myproc();
113 void     pinit(void);
114 void     procdump(void);
115 void     scheduler(void) __attribute__((noreturn));
116 void     sched(void);
117 void     setproc(struct proc*);
118 void     sleep(void*, struct spinlock*);
119 void     userinit(void);
120 int      wait(void);
121 void     wakeup(void*);
122 void     yield(void);
123 int      cps(void);
124 int      hello(char*);

```

*proc.c*

The definition of the hello will print whatever parameter is passed into it.

```

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

```

*user.h*

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

```
C: > Users > pakum > Desktop > operatingSystems > assign06 > code > C user.h
1  struct stat;
2  struct rtdat;
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
1  // System call numbers
2  #define SYS_fork 1
3  #define SYS_exit 2
4  #define SYS_wait 3
5  #define SYS_pipe 4
6  #define SYS_read 5
7  #define SYS_kill 6
8  #define SYS_exec 7
9  #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_hello 22
24 #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.

```

85  extern int sys_chdir(void);
86  extern int sys_close(void);
87  extern int sys_dup(void);
88  extern int sys_exec(void);
89  extern int sys_exit(void);
90  extern int sys_fork(void);
91  extern int sys_fstat(void);
92  extern int sys_getpid(void);
93  extern int sys_kill(void);
94  extern int sys_link(void);
95  extern int sys_mkdir(void);
96  extern int sys_mknod(void);
97  extern int sys_open(void);
98  extern int sys_pipe(void);
99  extern int sys_read(void);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_hello(void);
107 extern int sys_cps(void);

```

```

110 static int (*syscalls[])(void) =
111 {
112     [SYS_fork]    sys_fork,
113     [SYS_exit]    sys_exit,
114     [SYS_wait]    sys_wait,
115     [SYS_pipe]    sys_pipe,
116     [SYS_read]    sys_read,
117     [SYS_kill]    sys_kill,
118     [SYS_exec]    sys_exec,
119     [SYS_fstat]   sys_fstat,
120     [SYS_chdir]   sys_chdir,
121     [SYS_dup]     sys_dup,
122     [SYS_getpid]  sys_getpid,
123     [SYS_sbrk]    sys_sbrk,
124     [SYS_sleep]   sys_sleep,
125     [SYS_uptime]  sys_uptime,
126     [SYS_open]    sys_open,
127     [SYS_write]   sys_write,
128     [SYS_mknod]   sys_mknod,
129     [SYS_unlink]  sys_unlink,
130     [SYS_link]    sys_link,
131     [SYS_mkdir]   sys_mkdir,
132     [SYS_close]   sys_close,
133     [SYS_hello]   sys_hello,
134     [SYS_cps]     sys_cps,
135 };

```

## usys.S

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

```

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)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(hello)
33 SYSCALL(cps)

```

*sysproc.c*

The system process function `sys_hello` will 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.

```

93  //hello
94  int sys_hello(void)
95  {
96      char *ptr = "";
97
98      argstr(0, &ptr);
99
100     return hello(ptr);
101 }

```

*Makefile*

The makefile includes the user command called `testcase`.

<pre> 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\ </pre>	<pre> 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\ 261 </pre>
--	---

*testcase.c*

This file tests the hello function. It will iterate five times and make ten calls to the hello function with two different predefined parameters.

```
C: > Users > pakum > Desktop > operatingSystems > lab08 > 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 = "This is the result from Lab08\n";
8      char *strTwo = "CSIS 430: Online Lecture Video\n\n";
9
10     for(int x=0; x<5; x++)
11     {
12         hello(strOne);
13         hello(strTwo);
14     }
15
16     exit();
17 }
```

**Conclusion**

This lab was the same as assignment 6. However, to make it a little more interesting I made the hello method print out all the command line arguments. I chose to do this within the hello.c file and use a for loop to print out all the arguments. My design is almost identical to the echo.c file. However, I use a system call to hello whereas the echo.c file simply outputs to the terminal.