Lab #1: System Call Implementation Report

Group Members

- Bhagyesh Ravindra Gaikwad (bgaik001@ucr.edu, NetID: bgaik001)
- Dipro Chakraborty (dchak006@ucr.edu, NetID: dchak006)

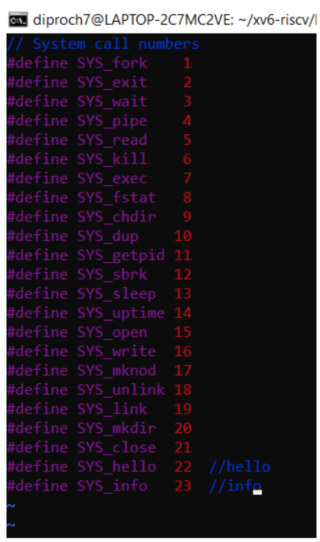
Objective

To add a new system call *int info(int param)* which takes an integer parameter as an input and takes the values 1,2 or 3. Based on the value, it returns :

1. A count of the number of processes in the system.
2. A count of the total number of system calls that the current process has made so far.
3. The number of memory pages the current process is using.

Setting up the System Call

- We first define a new system call number in the *kernel/syscall.h* file. //info is an indication of the code added for the new system call.



```
diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv/
// 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  //hello
#define SYS_info   23  //info
~
~
```

- This is followed by updating the system call table in the *kernel/syscall.c* file. This involves declaring the *sys_info* function and adding the *sys_info* syscall entry.

diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv/kernel

```c
extern uint64 sys_chdir(void);
extern uint64 sys_close(void);
extern uint64 sys_dup(void);
extern uint64 sys_exec(void);
extern uint64 sys_exit(void);
extern uint64 sys_fork(void);
extern uint64 sys_fstat(void);
extern uint64 sys_getpid(void);
extern uint64 sys_kill(void);
extern uint64 sys_link(void);
extern uint64 sys_mkdir(void);
extern uint64 sys_mknod(void);
extern uint64 sys_open(void);
extern uint64 sys_pipe(void);
extern uint64 sys_read(void);
extern uint64 sys_sbrk(void);
extern uint64 sys_sleep(void);
extern uint64 sys_unlink(void);
extern uint64 sys_wait(void);
extern uint64 sys_write(void);
extern uint64 sys_uptime(void);
extern uint64 sys_hello(void);  //hello: declaration
extern uint64 sys_info(void); //info: declaration

static uint64 (*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,  //hello:syscall entry
[SYS_info]    sys_info,  //info:syscall entry
};
```

- We define the syscall in the *kernel/sysproc.c* file. It creates the kernel function print_info() which takes an integer n as the parameter.

```c
//info syscall definition
uint64
sys_info(void)
{
  int n;
  argint(0, &n);
  print_info(n);
  return 0;
}
```

- Now, we define the kernel function print_info() in the file *kernel/proc.c* . The explanation of the code is in the next section.

diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv/kernel

```c
// parents are not lost. helps obey the
// memory model when using p->parent.
// must be acquired before any p->lock.
struct spinlock wait_lock;

int numSystemCalls = -1; // Initialize number of system calls

//hello: print hello message
void
print_hello(int n) {
  printf("Hello from the kernel space %d\n",n);
}

//info: print given info
void
print_info(int n) {

  //Case 1: Count the number of processes in the system
  if(n == 1) {
        struct proc *p;
        int count = 0;
        for(p = proc; p < &proc[NPROC]; p++){
                if(p->state != UNUSED) count++;

        }
        printf("Number of processes running in the system : %d\n",count);
  }
  //Case 2: Count the total number of system calls made by the current process so far
  else if (n == 2) {
        if(numSystemCalls == -1) printf("Total number of system calls made by the current process so far : %d\n",numSystemCalls);
        else {
                numSystemCalls = numSystemCalls + 1;
                printf("Total number of system calls made by the current process so far : %d\n",numSystemCalls);
        }
  }
  //Case 3: Count the total number of memory pages used by the current process
  else if (n == 3) {
        printf("Total number of memory pages used by current process : %d\n",(proc->sz/PGSIZE));
  }
  else {
        printf("Invalid input choice\n");
  }
}
```

- The final step in the kernel-space syscall interface is to add the void print_info(int) function definition in the *kernel/defs.h* file.

diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv/kernel

```c
// proc.c
int             cpuid(void);
void            exit(int);
int             fork(void);
int             growproc(int);
void            proc_mapstacks(pagetable_t);
pagetable_t     proc_pagetable(struct proc *);
void            proc_freepagetable(pagetable_t, uint64);
int             kill(int);
struct cpu*     mycpu(void);
struct cpu*     getmycpu(void);
struct proc*    myproc();
void            procinit(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(uint64);
void            wakeup(void*);
void            yield(void);
int             either_copyout(int user_dst, uint64 dst, void *src, uint64 len);
int             either_copyin(void *dst, int user_src, uint64 src, uint64 len);
void            procdump(void);
void            print_hello(int); //hello
void            print_info(int); //info
```

- After updating the kernel-space interface, the next important step is to update the user-space syscall interface. This is done by adding the *info* function entry into *user/usys.pl* and defining *int hello(int)* in the *user/user.h* function.

diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv/user

```
#!/usr/bin/perl -w

# Generate usys.S, the stubs for syscalls.

print "# generated by usys.pl - do not edit\n";

print "#include \"kernel/syscall.h\"\n";

sub entry {
    my $name = shift;
    print ".global $name\n";
    print "${name}:\n";
    print " li a7, SYS_${name}\n";
    print " ecall\n";
    print " ret\n";
}

entry("fork");
entry("exit");
entry("wait");
entry("pipe");
entry("read");
entry("write");
entry("close");
entry("kill");
entry("exec");
entry("open");
entry("mknod");
entry("unlink");
entry("fstat");
entry("link");
entry("mkdir");
entry("chdir");
entry("dup");
entry("getpid");
entry("sbrk");
entry("sleep");
entry("uptime");
#hello syscall for user
entry("hello");
#info syscall for user
entry("info");
~
~
~
~
~
~
~
"usys.pl" 42L, 728C
```

diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv/user

```
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(int) __attribute__((noreturn));
int wait(int*);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int hello(int);  //hello
int info(int);  //info

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void fprintf(int, const char*, ...);
void printf(const char*, ...);
char* gets(char*, int max);
uint strlen(const char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
int memcmp(const void *, const void *, uint);
void *memcpy(void *, const void *, uint);
~
~
~
~
~
"user.h" 44L, 1089C
```

Testing the System Call

- To test the system call created, we write a new program *info.c* in the *user* directory of *xv6-riscv*. It takes an integer *param,* a command line argument as the input and invokes the *info(param)* function with *param* as the parameter.

diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv/user

```
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

int main(int argc, char *argv[]) {
  int param=0;
  if(argc >= 2) param = atoi(argv[1]);

  printf("You opted for option %d \n",param);
  info(param);
  exit(0);
}
~
```

- Lastly, we edit the *Makefile* and append the instruction “$U/_info\” to UPROGS.

```
diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv
UPROGS=\
	$U/_cat\
	$U/_echo\
	$U/_forktest\
	$U/_grep\
	$U/_init\
	$U/_kill\
	$U/_ln\
	$U/_ls\
	$U/_mkdir\
	$U/_rm\
	$U/_sh\
	$U/_stressfs\
	$U/_usertests\
	$U/_grind\
	$U/_wc\
	$U/_zombie\
	$U/_test\
	$U/_info\
```

Executing the System Call

To run the code, we type the command *make qemu* at the *xv6-riscv* directory. This boots up the xv6 system. In the command line we type *info* with our desired input to get the results as below.

```
diproch7@LAPTOP-2C7MC2VE: ~/xv6-riscv
diproch7@LAPTOP-2C7MC2VE:~/xv6-riscv$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 1 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ info 1
You opted for option 1
Number of processes running in the system : 3
$ info 2
You opted for option 2
Total number of system calls made by the current process so far : 100
$ info 3
You opted for option 3
Total number of memory pages used by current process : 3
$ info 2
You opted for option 2
Total number of system calls made by the current process so far : 177
$ info 4
You opted for option 4
Invalid input choice
$ QEMU 4.2.1 monitor - type 'help' for more information
(qemu) quit
diproch7@LAPTOP-2C7MC2VE:~/xv6-riscv$
```

Explanation of the Code

Counting the number of processes in the system

In this case, each process is defined in the *kernel/proc.h* file. In the *proc.c* file, the pointer p acts as a reference to each process, and iterates over the NPROC processes in the system, to check which of the processes are active and not in the UNUSED state. The variable *count* counts the number of active processes in the system, and the final value of *count* gives us the number of processes running in the system.

```
//Case 1: Count the number of processes in the system
if(n == 1) {
      struct proc *p;
      int count = 0;
      for(p = proc; p < &proc[NPROC]; p++){
             if(p->state != UNUSED) count++;

      }
      printf("Number of processes running in the system : %d\n",count);
}
```

Counting the total number of system calls made by the current process

To return the count of the total number of system calls made by the current process, we need to set a variable that stores this count. In the file *kernel/syscall.c*, the function *void syscall (void)* is called when a system call is done. So, we add the counter (*numSystemCalls++*) within the if statement that the system call is valid, so we define *extern int numSystemCalls* to store the count. To print the value we initialize the variable in the *kernel/proc.c* file which checks if there are any system calls made, after which it prints out the result.

```
extern int numSystemCalls; //define system calls variable

void
syscall(void)
{
  int num;
  struct proc *p = myproc();

  num = p->trapframe->a7;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    p->trapframe->a0 = syscalls[num]();
    numSystemCalls++;
  } else {
    printf("%d %s: unknown sys call %d\n",
            p->pid, p->name, num);
    p->trapframe->a0 = -1;
  }
}
```

```
//Case 2: Count the total number of system calls made by the current process so far
else if (n == 2) {
      if(numSystemCalls == -1) printf("Total number of system calls made by the current process so far : %d\n",numSystemCalls);
      else {
             numSystemCalls = numSystemCalls + 1;
             printf("Total number of system calls made by the current process so far : %d\n",numSystemCalls);
      }
}
```

Counting the number of memory pages the current process is using

In the definition of a process in the *kernel/proc.h* file, the structure *proc* has a parameter uint64 *sz*, which is the size of the process memory (in bytes). Also the constant *PGSIZE* stores the value of the size of a page, so the number of memory pages used = (*proc->sz*) / *PGSIZE*.

```
}
//Case 3: Count the total number of memory pages used by the current process
else if (n == 3) {
      printf("Total number of memory pages used by current process : %d\n",(proc->sz/PGSIZE));
}
else {
```

****