OS Lab 4 - Remake

Omar Harb 900201063
Sara Mohamed 900203032
Mohamed Shaalan 900201539

Roles:

Omar: *shutdown* system call and user program

Sara: *getchildren* system call and user program

Mohamed: *statefilter* system call and user program

System Calls and Why

sys_shutdown

- shuts down machine, by closing all opened files, killing all processes, and then sending a shutdown signal using the `outb` function

- is a system call, as access to files and processes is only available to kernel, and not in user mode

sys_getchildren

- gets and prints children of a process using a given process pid

- is a system call, as access to processes and their attributes is only available in kernel mode, and not in user mode

sys_statefilter

- similar to ps, lists processes and their properties based on status, e.g. sleeping, running, runnable, etc.

- is a system call, as access to processes and their statuses is only available to kernel

User Programs for System Calls

shutdown

- “shutdown” user program, allows user to enter command line argument of number of seconds to wait until shutdown

- e.g. running “shutdown 5” induces shutdown after 5 seconds

getchildren

- “getchildren” user program, allows user to enter command line argument of pid of process in order to output list of children processes

- e.g. running “getchildren sh” lists children of the sh processes

statefilter

- “statefilter” program, allows user to enter command line argument of the desired status to filter processes according to

- e.g. running “statefilter sleeping” displays list of sleeping processes, along with their attributes such as pid, name, and ppid

Changes in Files

```
23  #define SYS_statefilter 22
24  #define SYS_shutdown 23
25  #define SYS_getchildren 24
26
```

new macros in syscall.h

```
106  extern int sys_statefilter(void);
107  extern int sys_shutdown(void);
108  extern int sys_getchildren(void);
```

externs in syscall.c

```
133  [SYS_statefilter] sys_statefilter,
134  [SYS_shutdown] sys_shutdown,
135  [SYS_getchildren] sys_getchildren,
```

function pointers to syscall pointer array in syscall.c

```
32  SYSCALL(statefilter)
33  SYSCALL(shutdown)
34  SYSCALL(getchildren)
```

SYSCALLS for macros in usys.S

```
26  int statefilter(int);
27  int shutdown(void);
28  int getchildren(char* pname);
```

function prototypes in user.h

```
123  int          statefilter(int);
124  int          shutdown(void);
125  int          getchildren(char* pname);
```

function prototypes in defs.h

```
92   int
93   sys_statefilter(void)
94   {
95     int state;
96
97     if (argint(0, &state) >= 0)
98       return statefilter(state);
99     else
100      return -1;
101  }
102  int
103  sys_shutdown(void){
104
105    return shutdown();
106  }
107
108  int
109  sys_getchildren(void)
110  {
111    char* pname;
112
113    if (argstr(0, &pname) >= 0)
114      return getchildren(pname);
115    else
116      return -1;
117  }
```

function calls in sysproc.c

added all function definitions in proc.c, ***pseudocode and description in next slide***

sys_shutdown and shutdown.c - description and pseudocode

in proc.c

```
int shutdown(void)
{
     capture ptable lock;
     for (every process p in the ptable)
          for (every i in 0<=i<NOFILE)
               fileclose p->ofile[i];

          if (p!=currproc && p!=initproc
          && p->state != UNUSED)
               kill p;
     release ptable lock;
     kill currproc and initproc;

     send shutdown signal;
     return -1;
}
```

in shutdown.c

```
int main(int argc, char* argv[])
{
     if (argc > 3)
          print “too many arguments” and exit;
     if (arg == 1)
          print help message and exit;
     if (argv[1] is negative)
          print error message and exit;
     else
          sleep(argv[1]*100);
          shutdown();
     exit();
}
```

sys_getchildren and getchildren.c description and pseudocode

in proc.c

```
int getchildren(int pid)
{
     capture ptable lock; proc* p, q;
     for (every process p in the ptable)
          if (p->id == pid)
               break;

     if (pname not found in ptable)
          return -1;

     for (every process q in the ptable)
          if (q->parent == p)
               print details of q;

     release ptable lock;
     return 0;
}
```

in getchildren.c

```
int main(int argc, char* argv[])
{
     if (no args given)
          print help message and exit;
     else if (argc > 2)
          print error message and exit;
     else if (argv[0] == argv[1])
          //(for the sake of testing)
          create child for current process;
          getchildren(argv[1]);
     else
          getchildren(argv[1]);

     exit();
}
```

sys_statefilter and statefilter.c - description and pseudocode

in proc.c

```
int statefilter(int st)
{
      \\state is saved as an enum in the proc
struct, so can be referred to with an int
      int count = 0;
      capture ptable lock; proc p;
      for (every process p in ptable)
            if(process unused) count++;
            if (p->state == st)
                  print details of p;
                  count++;


      if (count == 0)
            print message;
            return 0;
      else if(st == 0) print message;
      release ptable lock;
      return 0;
}
```

in statefilter.c

```
int main(int argc, char* argv[])
{
      turn argv[1] into lowercase if uppercase
alphabet;
      validate argv[1] among list of possible
states;


      if statements to call statefilter() based
on chosen state filter;

      \\e.g. if (argv[1] == sleeping)
            statefilter(2);


      exit();
}
```

Explanation for Changes

- adding macros in usys.S and syscall.h helps place the proper arguments in relevant registers and issue some kind of trap instruction, by substituting in the definition of SYSCALL(name), according to the values defined in syscall.h

- adding function prototypes in user.h and defs.h allows the functions to be called in our user programs, which include these header files

- In syscall.c, we map the system call number (defined in syscall.h,) to the corresponding kernel function that handles the system call (eg: sys_statefilter), as well as add it to the list of “extern”s which extends the scope of the function to be used elsewhere

- in sysproc.c, we define the system calls, by calling the functions defined in proc.c and declared in defs.h and user.h within them