How to Add a New System Call in Minix

Created by Mingdong Shang and Wenliang Du (Syracuse University)

(A) Kernel Modification

Step 1: Find a free slot in the `table.c` file. Both File System (fs) and Memory Management (mm) modules have their own `table.c` files: `fs/talbe.c` and `mm/table.c`. Decide where you want to add the new system call; either is ok, but if the new system call is related to file system, put it in `fs/talbe.c`; otherwise put it in `mm/table.c`. In this document, we assume that `fs/table.c` is used.

```
PUBLIC _PROTOTYPE (int (*call_vec[]), (void) ) = {
        no_sys,         /*  0 = unused  */
        do_exit,        /*  1 = exit    */
        do_fork,        /*  2 = fork    */
        …
        no_sys,       /* 57 = unused */
        …
}
```

Next find a free slot for the new system call. For example, if you would like to use entry `57` for your new system call (`mycall`), do the following:

```
Replace
        no_sys,          /* 57 = unused    */
with
        do_mycall,       /* 57 = do_mycall */
```

Step 2: Add a function declaration in `/usr/src/fs/proto.h`

```
    _PROTOTYPE ( int do_mycall, (void) );
```

Step 3: Add `do_mycall()` implementation to either a new file or an existing file under `/usr/src/fs`. If you add to an existing file, you do not need to change `Makefile`; if you prefer to create a new file, you need to modify `Makefile` accordingly (you can use `write.c` as an example to see how to modify `Makefile`). To learn how to write `do_mycall()`, you can use `do_chmod()` in protect.c as an example. You need to know how `do_mycall()` gets the data sent by the user programs. It should be noted that user programs actually use message to pass their data. That is why you should get the data from the `message` data structure.

```
PUBLIC int do_mycall()
{

    int a, b, c;
    /*   Data sent by the application can be retrieved using the global variable m   */
    /*   Variable m is defined in fs/glo.h.   Also see Step 4.   */
    a = m.m1_i1;
    b = m.m1_i2;
    c = m.m1_i3;

    /* Put the actual implementation of the mycall() system call here.*/
     return ok or error;
}
```

(B) Directly Invoke the System Call

Once you have created the system call, you can directly invoke it via `_syscall()`:

```
#include <lib.h>
#include <stdio.h>

/* This function can call the system call, and pass the parameters */
int mycall(a, b, c)
int a, b, c;
{
   message m; /* this structure is used to pass parameters */
   m.m1_i1 = a;
   m.m1_i2 = b;
   m.m1_i3 = c;

   return (_syscall(FS, 57, &m));
}

void main(int argc, char *argv[])
{
    int r;
    r = mycall(10, 5, 1) ;
}
```

(C) Invoke the System Call via a Library Call

If you want other programs to be able to call the system call, you might want to put the mycall() function in the standard library (e.g. `libc.a`). The following step describes how you can achieve this.

Step 1: You can put your implementation in the `/usr/src/lib/posix` directory. The file name should always start with an underscore.

```
File Name: /usr/src/lib/posix/_mycall.c
#include <lib.h>
#include <unistd.h>

/* Suppose you pass three integers to this system call    */
PUBLIC int mycall(a, b, c)
int a, b, c;
{
  message m;

  /* You may pass a buffer pointer here if necessary,
     using m.m1_p1 instead. You can learn from
     /usr/src/lib/posix/_write.c  */
  m.m1_i1 = a;
  m.m1_i2 = b;
  m.m1_i3 = c;
  return(_syscall(FS, MYCALL, &m));
}
```

Note: if you have put your new system call in `mm/table.c`, rather than in `fs/table`.c, you should use MM, instead of `FS,` when calling `_syscall().`

Step 2: Modify `/usr/src/lib/posix/Makefile`: add the following commands to the corresponding places:

```
    $(LIBRARY)(_mycall.o) \

    $(LIBRARY)(_mycall.o):_mycall.c
    <tab key>$(CC1) _mycall.c
```

Step 3: Add the following definition to `/usr/include/minix/callnr.h`

```
       #define MYCALL      57
```

Step 4: Create `/usr/src/lib/syscall/mycall.s`

```
.sect .text
.extern   __mycall
.define   _mycall
.align 2

_mycall:
   jmp __mycall
```

Step 5: Now, you can invoke the system call via the library function mycall(). Here is an example showing how you can use this new system call in an application.

```
......
    int x = 5;
    int y = 10;
    int z = 1;
    mycall(x, y, z);
.........
```