How to Add a System Call in MINIX 3.2.1

Defining the system call number

In the file `/usr/src/include/minix/callnr.h` there is a list of macro definitions of the form:

```
#define EXIT        1
#define FORK        2
#define READ        3
#define WRITE       4
#define OPEN        5
#define CLOSE       6
#define WAIT        7
#define CREAT       8
#define LINK        9
#define UNLINK     10
...
…
```

These macros define the system call number and symbols (`EXIT, FORK, READ` etc.) that can be used in place of the call number.

To define the new system call number, add the following lines to callnr.h:

```
#define MYCALL   69
```

Adding the Process Manager (PM) server table entry

The file `/usr/src/servers/pm/table.c` includes the following declaration of a table (or array) of system calls implemented by the PM server:

```
int (*call_vec[])(void) = {
     no_sys,        /*  0 = unused */
     do_exit,  /*  1 = exit   */
     do_fork,  /*  2 = fork   */
     no_sys,        /*  3 = read   */
     no_sys,   /*  4 = write  */
     no_sys,        /*  5 = open   */
     no_sys,        /*  6 = close  */
     do_waitpid,    /*  7 = wait   */
     no_sys,        /*  8 = creat  */
     no_sys,        /*  9 = link   */
     no_sys,        /* 10 = unlink */
     ...
     ...
     no_sys,         /* 69 = unused  */
     ...
     ...
};
```

The table is an array of pointers to functions. There is one entry in the table for each system call and the array index position in the table is the system call number.

To add a system call to the PM server, replace the unused table entry at position 69 with the name of the function that will implement the system call (`do_mycall`).

Replace the line:

```
  no_sys,          /* 69 = unused  */
```

with:

```
  do_mycall,     /* 69 = mycall  */
```

MYCALL is now the system call name that maps call number 69 to the entry for `do_mycall` in `servers/pm/table.c`.

Defining the system call prototype

Users of the `do_mycall` system call will need a function prototype in order to invoke `do_mycall`. The function prototype is defined in `servers/pm/proto.h`. Open `proto.h` and you will see a number of function definitions. Add the `do_mycall` function definition after the `misc.c` list of definitions.

Find the following lines in `/usr/src/servers/pm/proto.h`:

```
/* misc.c */
int do_reboot(void);
int do_sysuname(void);
int do_getsysinfo(void);
int do_getprocnr(void);
int do_getepinfo(void);
int do_getepinfo_o(void);
int do_svrctl(void);
int do_getsetpriority(void);
```

And add the following lines:

```
int do_mycall(void);
```

Implementing the system call

You must now provide an implementation of `do_mycall`. The initial implementation of the system call will simply print the message:

```
     Hello World! This is my system call!
```

In the file /usr/src/servers/pm/misc.c create a new C function with the following contents:

```
#include <stdio.h>

int do_mycall() {
    printf("Hello World! This is my system call!\n");
    return 0;
}
```

You now have the definition of a system call number, a system call table entry, the definition of a system call prototype, and an implementation of the system call.

Creating a level library interface for the system call

You can now create a library with a function that to call you implementation of do_mycall.

Create a file named mycalllib.h under /usr/include with the following contents:

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

int mycall(void) {
     message m;
     _syscall(PM_PROC_NR, MYCALL, &m);
     return 0;
}
```

You may use this library on you userspace applications.

Compiling and testing the system call

You compile the system call and include it in the OS boot image by completing the following steps:

In /usr/src/releasetools compile and install services:

```
     # make services
     # make install
```

Sync & Shutdown Minix and start it to run the new version of the OS

Testing your system call

Under `/home` create a file named `test.c`

```
#include <mycalllib.h>

int main (void){
     mycall();
     return 0;
}
```

Complile with and run:

```
$ clang -o test test.c
$ ./test
```