Juan Jauregui
2017-02-04
COP4600: Operating Systems
Project 1 Report

Project 1 Report

Project 1 involved creating three new elements in the Minix OS:

1. A new property in the process manager's process table called **tag,** and
2. A new system call to get or set a process's **tag**
3. A new set of library calls, get_tag(int) and set_tag(int, int), to allow C programs to easily make the system call.

The following sections summarize the process to accomplish these tasks.

Part 1: Creating the **tag** process property

Creating the **tag** property involves modifying the process manager service in two locations.

1. /usr/src/servers/pm/mproc.h

   mproc.h holds the struct that keeps track of each process's information in the process manager. **Necessary modification:** Add an **int tag** property to the struct.

2. /usr/src/servers/pm/forkexit.c

   forkexit.c holds the function which is responsible for spawning new processes, **do_fork();** This function handles creating a new entry in **mproc,** which it refers to by the struct pointer **rmc.** The function initializes the properties of **rmc** like mp_pid. **Necessary modification:** Add an expression setting **rmc->tag = 0.** This will set each new process's entry in the process table to have a tag of 0.

Part 2: Creating the **system call** to modify the **tag** of a process

Creating a new system call involves registering it in several places and then implementing it:

1. /usr/src/servers/pm/table.c

   table.c holds the list of system calls implemented by the process management server. **Necessary modifications:** Find an unused system call entry. I used table entry **44** to register new system call tag_stuff.

```
int (*call_vec[])(void) = {
    no_sys,             /*  0 = unused   */
    do_exit,            /*  1 = exit     */
    ...
    tag_stuff,          /* 44 = tag_stuff  */
    ...
};
```

Juan Jauregui
2017-02-04
COP4600: Operating Systems
Project 1 Report

2. /usr/src/include/minix/callnr.h

callnr.h holds a set of macro definitions that map system calls to their specific C implementations. **Necessary modifications:** Define TAG_STUFF to *44*.

```
#define EXIT        1
#define FORK        2
#define READ        3
...
#define TAG_STUFF   44
...
```

3. /usr/src/servers/pm/proto.h**:**

proto.h holds a set of the function prototype declarations for system calls. **Necessary modifications:** 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);
```

And add the following line:

```
int tag_stuff(void);
```

4. **Implementing the system call: In** /usr/src/servers/pm, I created a new C file, **tag_stuff.c.** In it, I defined the function **int tag_stuff()** that I defined in proto.h. I used the Minix **message** interface's **m7** struct for message passing to the system call because it supports enough int parameters to pass all the requisite components to determine what the caller intended the call to achieve, and whether they have the permissions to do so: **Caller's PID, caller's UID, PID in question, whether it's meant to get or set the tag (1=set, -1=get), and the new tag (or -1 if not setting):**

```
int caller_pid = m_in.m7_i1;
int caller_uid = m_in.m7_i2;
int pid_in_question = m_in.m7_i3;
int set_or_nah = m_in.m7_i4;     //1 or -1
int new_tag = m_in.m7_i5;


//If caller PID is same as PID in question, they can get the tag
int can_get_it = 0;
if((caller_pid == pid_in_question) || caller_uid == 0)
   can_get_it = 1;

//If the caller's UID is 0 (root), they can set the tag
int can_set_it = 0;
if(caller_uid == 0){
   can_set_it = 1;
```

Accessing the process table in order go get or set the tag is as simple as calling **find_proc(pid_in_question)**, which returns either **null** (if no such process exists) or a pointer to an mproc struct, which can then simply be queried using **mproc->tag** to either get it or set it.

Note that this requires

```
#include "pm.h"      //For global message passing
#include "mproc.h"  // for access to process table mproc
```

At this point, any program can make this system call using the following syntax:

```
message m;
m.m7_i1 = (int)getpid();            //Send own PID to check permissions
m.m7_i2 = (int)getuid();             //Send UID to check if 0 (root)
m.m7_i3 = pid_in_question;          //Send PID of interest
m.m7_i4 = -1;                        //-1 = Not setting a new tag
m.m7_i5 = -1;                       //No new_pid
int ret_val = _syscall(PM_PROC_NR, TAG_STUFF, &m);
return ret_val;
```

Finally, I added **tag_stuff.c** to the **SRCS** section of the PM server's **MAKEFILE**, so that it gets recompiled.

5. **Making a user library function to make the system call:**

- In /usr/src/include/unistd.h, I added the function prototypes for **get_tag(int)** and **set_tag(int,int)** immediately before__END_DECLS

- In /usr/src/lib/libc/sys-minix, I created a C file called **tag_stuff.c** to define the implementation of both functions. These functions take integer parameters, form the message structure, make the system call, and return its result. The code near the end of step #4 comes from get_tag(int pid_in_question).

- Add **tag_stuff.c** to the list of SRCS in /usr/src/lib/libc/sys-minix/Makefile.inc

5. **Building everything**

In /usr/src/releasetools,

make services

make install

Juan Jauregui
2017-02-04
COP4600: Operating Systems
Project 1 Report

This compiles all the changes to the process manager service (the **tag** in mproc.h, the new expression in do_fork(), and everything involving the system call)

In /usr/src/include:

make dependall install

In /usr/src/lib/libc

make dependall install

in /usr/src/releasetools

make hdboot

This compiles the changes to the user library.

After a reboot, it is now possible to compile and run the test programs provided with the assignment.