

CSIS0230A Principles of Operating Systems (Class A)

Tutorial 4

Process Structure

In this tutorial, you have to add two system calls—*getchild* and *getname*. The **specifications** of the system calls are as follows:

int *getchild*(*pid_t* *pid*, *pid_t** *child*, **int** *size*)

Given a specific process *pid*, fill the process IDs of all its children in the *child* array of size *size*. At most that many entries in *child* will be used by the system call. It returns the number of children of *pid*. Thus a caller can check whether the buffer is large enough by inspecting the return value.

Input:

pid—the specified process ID
child—for storing the children process IDs
size—the size of *child* array

Return value: the number of children of *pid*

Possible errors:

ESRCH—process cannot be found
EFAULT—memory specified in *child* is not writable

int *getname*(*pid_t* *pid*, **char*** *name*)

Given a process *pid*, put the name of the program executed by the process to *name*.

Input:

pid—the specified process ID
name—a buffer for storing the *name* of the process

Return value: 0 if the request is served successfully.

Possible errors:

ESRCH—process cannot be found
EFAULT—memory specified in *name* is not writable

After you finish writing these system calls, write a user program to test whether the system calls work. A pid should be read from the command line, and the program should print all the pids and names of all its children.

For example, one can execute the program as

```
./a.out 541
```

Output:

```
The children of pid 541 are:  
9305: in.telnetd  
9146: in.telnetd
```

One possible structure of the user program:

```
#include <syscall.h>
#include <errno.h>
#include <sys/types.h>
#include <stdlib.h>
/* Define __NR_getchild and __NR_getname here */
_syscall3(int, getchild, pid_t, pid, pid_t*, child, int, size);
_syscall2(int, getname, pid_t, pid, char*, name);
int main(int argc, char** argv) {
    pid_t pid;
    pid = atol(argv[1]);
    /* find number of children of pid (calling getchild with size=0) */
    /* allocate memory for children array */
    /* find all pid's children process IDs */

    /* for each child process ID
       find the name of process
       print the process ID and name */
}
```

But remember to check the for errors returned by the user program. To ease debugging, we recommend that you print out the error message in cases of error.

Hints:

1. There are some data types or functions that you may need to use:

function or type	include
<i>pid_t</i>	<linux/types.h>
struct <i>task_struct</i>	<linux/sched.h>
int <i>copy_from_user</i> (void* to, void* from, int size)	<asm/uaccess.h>
int <i>copy_to_user</i> (void* to, void* from, int size)	<asm/uaccess.h>
struct <i>task_struct</i> * <i>find_task_by_pid</i> (pid_t)	<linux/sched.h>

2. Using the user address space without checking will leave a really big security hole in your kernel. So check them by using *copy_to_user* and *copy_from_user*.
3. Use *printk* for debugging. To see the messages from *printk*, run your program from a text-mode console rather than in an X terminal.
4. Don't forget to modify *entry.S* and *Makefile*.