Operating Systems (234123) - Spring 2015
Home Assignment 1 - Wet

Due date:	Sunday, 17.4.2015, 23:59
Teaching assistant in charge:	Arie Tal

Important: the Q&A for the exercise will take place at a public forum Piazza only. Please note, the forum is a part of the exercise. Important clarifications/corrections will also be published in the FAQ on the site. A number of guidelines to use the forum:

- Read previous Q&A carefully before asking the question; repeated questions will probably go without answers
- Be polite, remember that course staff does this as a service for the students
- You are not allowed to post any kind of solution and/or source code in the forum as a hint for other students; In case you feel that you have to discuss such a matter, please come to the reception hour
- When posting questions regarding hw1, put them in the hw1 folder

Note: Start working on the assignment as soon as possible!!! This assignment involves algorithmic design, code writing, and extensive testing and debugging.

1 Introduction

Managing resources is a common aspect of Operating Systems. Part of managing resources has to do with the allocation of limited resources to users and to applications. One such resource is Processes. In an unbounded system, each process could create as many child processes as memory allows before the system starts to get impacted by it. To avoid such failures either due to programming bugs, or malicious intent, Operating Systems must protect themselves and their other users from such program behavior. For example, on the T2 system each student account is limited to at most 50 processes[1].

However, the limit on the user acccount was found to be too coarse, and a new, finer, approach was suggested. With this new approach, the user may define for each child process how many sub-processes (children, and grand-child, etc.) it may have. With this new approach, a user has the flexibility to determine how many sub-processes each application that they run may have, and thus run multiple applications and limit each application's ability to create too many processes.

To implement the new approach for process resource management, a new API was defined. The following describes the system calls prototype and behavior. The actual implementation of this behavior should be implemented in the wrappers and in the new (and in some existing) system calls.

int set_child_max_proc(int maxp) - system call #243 This system call may be called at any time before a call to fork(). It sets the maximum number of sub-processes that any process created via any subsequent fork() could have at any given time. *Please note that this limit is* **not** *on the number of calls on forks(), but on the number of* **child and grand-child, etc. processes** *(including zombie processes) that a child process may have.*

A child process may not set the child_max_proc of any of its children to a value greater than the value that it has minus 1. Attempting to set maxp to a value higher than is current permitted will cause

[1] You can check your user account's process limit on T2 using the "limit" command in TCSH or the "ulimit -u" in BASH.

this API to return -1 and to set errno to EPERM (Operation not permitted).
A negative parameter to set_child_max_proc() shall reset the limit, i.e. return to the default behavior. Note that in the default behavior, the maximum number of sub-processes that a child process may have may still be limited in practice by the number of sub-processes that all its siblings have if the parent (or some grandparent) process was created with a limit.

int get_max_proc() - system call #244 Returns the current value of max_proc for the calling process.

int get_subproc_count() - system call #245 Returns the total number of current sub-processes that the calling process has (i.e. the total count of the entire sub-tree of processes).

You need to write code wrappers and internal system call implementations for the above system calls (see the slides from Tutorial 2). For each one of the system calls you should implement a code wrapper with the appropriate name (e.g for sys_set_child_max_proc you should implement the wrapper set_child_max_proc). The return value of the system call should be 0, or a positive value for success, or the appropriate negative error code (see above for the exact codes).

2 Example Wrapper

Here is an example of the code wrapper for example_wrapper (see explanation below). Follow this example to write the other three code wrappers:

```
int example_wrapper(int *array, int count) {
   long __res;
   __asm__ volatile (
       "movl $245, %%eax;"
       "movl %1, %%ebx;"
       "movl %2, %%ecx;"
       "int $0x80;"
       "movl %%eax,%0"
       : "=m" (__res)
       : "m" ((long)array), "m" (count)
       : "%eax","%ebx","%ecx"
   );
   if ((unsigned long)(__res) >= (unsigned long)(-125)) {
      errno = -(__res); __res = -1;
   }
   return (int)(__res);
}
```

Explanation of inline assembler:

1. `movl $245, %%eax` copy system call number to register eax.
2. `movl %1, %%ebx` - copy first parameter (array) to register ebx.
3. `movl %2, %%ecx` - copy second parameter (size) to register ecx.
4. `int $0x80` system call invocation via interrupt 0x80.
5. `movl %%eax,%0` copy the value that was returned by the system call to `%0` (which is the first output operand).
6. `: "=m" (__res)` output operand.
7. `: "m" ((long)array), "m" (size)` input operands.
8. `: %eax,%ebx,%ecx` list of clobbered registers. Inform gcc that we use `eax`, `ebx`, `ecx` registers.

You should read the following document for information about the commands used in the preceding code segment: `http://www-106.ibm.com/developerworks/linux/library/l-ia.html`. The code wrappers will not be put as a part of the kernel but in a separate .h file (syscall_maxproc.h). This file shouldn't be put inside the kernel but as a separate file. When you test your system calls you need to include this file in your user mode test program.

3 Notes and Tips

- You are not allowed to use syscall functions to implement code wrappers, or to write the code wrappers for your system calls using the macro _syscall1. You should write the code wrappers according to the example of the code wrapper given above.
- Your solution should be as efficient as possible. Traversing the process tree on every fork() call is not recommended.
- Submit only modified files from the Linux kernel.
- No need to print your code as part of the wet HW paper submission to the course's cell. You need only to explain what you have done in the code.
- Start working on the assignment as soon as possible. The deadline is final, NO postponements will be given, and high load on the VMWare machines will not be accepted as an excuse for late submissions.
- Write your own tests. We will check your assignment also with our test program.
- Linux is case-sensitive. entry.S means entry.S, not Entry.s, Entry.S or entry.s.
- You can assume that the system is with a single CPU.
- In case you decide to allocate memory in the kernel (depends on your implementation), you should use kmalloc and kfree in the kernel in order to allocate and release memory. If kmalloc fails you should return ENOSPC. For the kmalloc function use the flag GFP_KERNEL for the memory for kernel use.
- Pay attention that the process descriptor size is limited. Do not add too many new fields. New fields should be added at the end of the process descriptor struct in order for the code to be backward compatible.
- If a process exits (or dies) while it has child processes, these child processes become children of init by default. For the task of process resource management, you should retain the information on the process's original parent (avoid using p_opptr which is there for debugger purposes) and modify it to point to the grand parent process if the parent process exits.

4 What should you do?

Use VMware, like you learned in the preliminary assignment, in order to make the following changes in the Linux kernel:

1. Put the implementation of the new system calls in the file kernel/syscall_maxproc.c that you will have to create and add to the kernel. Update the makefile in that directory to compile your new file too. (Tip: add it to obj-y).
2. Update sched.h (constant + new fields definition)
3. Update entry.S (add new system call number)
4. Make any necessary changes in the kernel code so the new system calls can be used like any other existing Linux system call. Your changes can include modifying any .c, .h or .S (assembly) file that you find necessary.

5. Put the wrappers functions in syscall_maxproc.h
6. Recompile and run the new kernel like you did in the preliminary assignment.
7. Boot with your new Linux, and try to compile and run the test program to make sure the new system calls work as expected.

Did it all? Good work, Submit your assignment. ☺

5 Submission

The submission of this assignment has two parts:

- An electronic submission - you should create a zip file (use zip only, not gzip, tar, rar, 7z or anything else) containing the following files:
  1. A tar ball named kernel.tar.gz containing all the files in the kernel that you created or modified (including any source, assembly or makefile files). To create the tarball run (inside VMWare):

     ```
     cd /usr/src/linux-2.4.18-14custom
     tar -czf kernel.tar.gz <list of modified or added files>
     ```

     For example, if the only files you changed are arch/i386/kernel/entry.S and kernel/syscall_maxproc.c you should run:

     ```
     cd /usr/src/linux-2.4.18-14custom
     tar czf kernel.tar.gz arch/i386/kernel/entry.S kernel/syscall_maxproc.c
     ```

     Make sure you don't forget any file and that you use relative paths in the tar command, i.e., use kernel/syscall_maxproc.c and not /usr/src/linux-2.4.18-14custom/kernel/syscall_maxproc.c
  2. A file named submitters.txt which includes the ID, name and email of the participating students. The following format should be used:

     ```
     Bill Gates bill@t2.technion.ac.il 123456789
     Linus Torvalds linus@gmail.com 234567890
     Steve Jobs jobs@os_is_best.com 345678901
     ```

  3. A file named syscall_maxproc.h that contains the implementation of your wrapper functions.

  Important Note: Make the outlined zip structure exactly. In particular, the zip should contain only the following files (no subdirectories):

  ```
  zipfile  -+
           |
           +- kernel.tar.gz
           |
           +- submitters.txt
           |
           +- syscall_maxproc.h
  ```

- A printed submission. You should write a short (1 page) summary explaining what you have done in this assignment, which changes to the kernel data structures you have done and which functions have been modified and how. This need to be attached with the relevant cover page (separate from the dry) and submitted to the course cell.

Good Luck!
The Course Staff