1 Change History

1.0 Initial version.

2 HW1 Dry

2.1 General

2.1.1 What tutorials are required for this assignment?
This assignment references material from tutorial 2 (r2_linux) through tutorial 4 (r4_context_switch).

2.1.2 Can the answers be written in Hebrew?
The dry assignment was written in English for the teacher’s convenience. You may write your answers in English or Hebrew.

2.2 Question 1

2.3 Can values, except for %esp be stored in other registers?
Yes. In general, any value can be stored anywhere as in a regular program. The only limitation is that of storing %esp in some other register (as is usually done).

2.4 Do we assume the code will be hand-crafted to overcome the limitation?
In your answer you may assume that whatever process you define to overcome this limitation when accessing local variables and parameters, the compiler would be smart enough to generate code as per your specifications. Just make sure your specification is feasible.

2.5 Question 4

2.5.1 Is the answer to Question 4 (b) supposed to be numerical?
In general, if the limit for a process is numerical and known (and easy to calculate), then you should give it. If the limit is not known or is a function of several constraints, then you should explain what these constraints are and how they affect the limit.
3 HW1 Wet

3.1 Default behavior

3.1.1 What is the default behavior for a process as far as the max_proc limit?

The default behavior is "unlimited". If you retain a max_proc field in the task_struct, you may use a value of -1 to store that default. The specific value is not checked by any of the APIs in the assignment, see below.

3.1.2 What should get_max_proc() return in the default case?

The wrapper should return -1 (for error) and set errno = EINVAL (i.e. the system call should return -EINVAL).

3.1.3 What is the default behavior for fork or clone?

If your retain a child_max_proc field in your task_struct, then that field should be set with the default value (-1, i.e. "unlimited") when the task is created (the value is not inherited from the parent). Any calls to fork() or clone() would use that value for the child process that the process creates. Once a call is made to set_child_max_proc() with a value greater than -1, that value would be used for any subsequent fork() or clone() calls made by the process. In other words, any fork() or clone() call will set the max_proc field of the new process to the value of the child_max_proc field in the parent process at the time of the call to fork() or clone(). Note that the max_proc value does not change throughout the life of the process.

3.2 fork() and clone() error values

3.2.1 What is the return value from fork() or clone() in case the process limit is reached?

The behavior of these system calls is the same as when they reach any other limit in the system. That is, they return -EAGAIN.

3.3 Unrecognized errno symbol in your syscall_maxproc.h

You may add #include <errno.h> in your syscall_maxproc.h file, which will make the compiler recognize errno in your wrapper and in your test programs.

3.4 get_subproc_count behavior

3.4.1 Does get_subproc_count count the parent process as well?

No, this system call counts only the child (and grandchild, and grand-grand-child, etc.) processes of the calling process, so for a process that has no child process this system call returns 0. Please note that as many other system calls, this system call should be efficient. That is, try avoiding traversing the entire sub-process tree on this system call (see below).

3.5 What does traversing the sub-process tree mean?

The process tree is composed of many nodes (processes), that have children and their children may have children as well. Traversing that entire tree means traversing each and every node in the tree.

On the other hand, if you need to scan from a node (process) to its grand-grand-parent, i.e. you are traversing only up the tree (limited in the depth of the tree), then that is not referred to as traversing the entire tree, and is permitted when needed.
3.6 Clarifications about child process limits

Each process in the process tree may define a limit on its sub-process tree size. That limit is set by its parent’s child_max_proc value when the process is first created. The limit adds a constraint on the sub-process tree.

If a process that adds such a constraint creates sub-processes (up to its limit) without setting any limits on the sub-processes, and then exits (either finishes, or gets killed) and is removed from the process tree, its former sub-processes become exposed to the limits set by its former parent process. In essence, the grand-parent process becomes the limiting process for the sub-process tree.

However, in terms of the operating system, the child processes of a deceased parent become orphaned and are adopted by the child_reaper process (a.k.a. init), so that init may call wait() on them. Note that in the general case the grand-parent process is unequipped to deal with adopted child processes (it does not know to wait() for them). What this means is that you should handle this "book keeping" by adding a member in the task struct that points to the bookkeeping parent.

3.7 Cloning and threads

We haven’t studied threads yet, so the requirements in the assignment are relaxed as far as threads are concerned. When a process or thread creates another thread (with the flags that change the thread’s parent to be the process’s parent), the value of child_max_proc in the thread calling clone() is the one that sets the value of max_proc in the created thread.

When a thread in a thread group leaves, and a different thread is located to take its place as the parent of its sub-processes (this is the only exception to orphaned child processes becoming adopted by init), the child processes become subjected to the new thread’s constraints. That is, the adopting thread also becomes the bookkeeping parent for the adopted child processes. In case the number of adopted child processes is greater than the limit allowed by the adopting thread, the adopted child processes would not be killed (even though the constraint is violated), but new calls to fork() or clone() by the adopting thread would be rejected until its number of sub-processes is less than it’s limit. Note that in this case only the constraint on the adopting thread is temporarily violated, but the constraint on its parent isn’t.

3.8 Changes to kernel source files, e.g. sched.c

As stated in the assignment, you may change any source file in the kernel necessary for your implementation. Make sure you submit all your changes in your kernel.tar.gz file, so that we are able to reproduce and test your implementation with our test programs.

3.9 I can’t find function xyz() in the kernel source tree, where is it?

Besides saying that grep -r is your friend, you may use other (more useful) tools as well. For example, you may use ctags, which is installed in your Linux image. If you do choose to use it, you may do as follows:

In the kernel source directory, run the command

```
> ctags -R -h.c,.h,.s,.S
```

This will create a "tags" file in your kernel source directory. Next you may open vim to view/edit files. In vim you may use the following:

```
:ta <symbol> This command will look for the declaration or definition of the symbol <symbol> in the source tree. For example, use :ta do_fork.

:tn This command would skip to the next matching tag of the last :ta command.

:tp This command would skip to the previous matching tag of the last :ta command.
```

Ctrl+] When viewing/editing a source file, this key combination would look for the symbol that is currently under the cursor. For example, if you see a reference to do_fork in the code, move the cursor over it and press this key combination; this will be the equivalent of the :ta do_fork command.
Ctrl+T The tag commands act like a stack, so that you may traverse down a call tree using the Ctrl+] combo. You may use Ctrl+T to "pop the stack" and go to the last position that you were in before you pressed Ctrl+]..

The above commands work across files and directories in your source tree. For it to work properly you need to start vim in the same directory where the "tags" file was created, or it would not be able to find it. There are other commands that work with tags, but these are the most common ones.

Good Luck!
The Course Staff