CS 444 Writing Assignment 1

Jared Wasinger

I. Comparing FreeBSD with Linux

Processes in FreeBSD are similar to Linux in many ways. The relationship between processes and threads in Linux and FreeBSD systems are hierarchical. When a system is booted, a special ‘init‘ process is started to complete system initialization. All subsequent processes exist as children of the ‘init‘ process. The system call ‘clone‘ is used to create a new process (same?).

In Linux processes are implemented in the kernel via a doubly-linked list known as the ‘task list‘. Each entry in the ‘task list‘ contains information about the state of a given process on the system.

```
struct thread_info {
        struct task_struct
*task;
        struct exec_domain
*exec_domain;
        unsigned long
flags;
        unsigned long
status;
        __u32                 cpu;
        __s32
preempt_count;
        mm_segment_t
addr_limit;
        struct restart_block
restart_block;
        unsigned long
previous_esp;
        __u8
supervisor_stack[0];
};
```

Both Linux and FreeBSD threads maintain a kernel stack. This is so that Kernel operations on user-space process can be done without exposing sensitive Kernel information to the user-space.

The implementation of processes and threads in FreeBSD differs from Linux in several important ways. FreeBSD makes a clear distinction between threads and processes. Threads must be spawned by a process. In their lightest form, threads can share nearly all resources with their parent process (including PID).

Both Linux and FreeBSD conform to the POSIX Threads standard. This was probably chosen because having multiple operating systems support a common interface for concurrency is mutually benefical to both FreeBSD and Linux. This is because it is easier to write programs that can be cross-compiled between FreeBSD and Linux operating systems.

In FreeBSD CPU resources are given to threads based on ‘scheduling class‘ and ‘scheduling priority‘(97). FreeBSD has two kernel scheduling classes and three user scheduling classes (97):

- (0-47) bottom-half kernel (interrupt)
- (48-79) real-time user
- (80-119) top-half kernel
- (120-223) time-sharing user
- (224-255) idle user

In FreeBSD, threads in the bottom-half kernel class will always be given preference. Tasks in this class are kernel-intterupt threads that need to be run as soon as possible (97). Threads in the real-time and idle classes can be set by the application using the ‘rtprio‘ system call.

FreeBSD thread scheduling is similar to Linux in its goal. FreeBSD thread scheduling is implemented in the form of a round-robin allocated thread system through assignment of processor time slices to threads. Like Linux, FreeBSD scheduling gives priority to interactive jobs by increasing the scheduling priority of threads that are blocked on I/O operations for 1 or more second (125-126). I assume that this feature was implemented on both operating systems in order to make them more viable for use on desktop machines which typically run more interactive programs than traditional mainframe computers.

II. Comparing Windows with Linux