415.340/1 Operating Systems

Lecture 19 Handout

Linux task structure (or Process Control Block)

Like other UNIX systems the Linux PCB includes information which is not essential for keeping track of the process itself, such as the links to keep this process in lists.

You aren't expected to understand this structure, but it does show you how complicated process control blocks can be.

```
struct task_struct {
/* these are hardcoded - don't touch */
    volatile long state;  /* -1 unrunnable, 0 runnable, >0 stopped */
    long counter;
    long priority;
    unsigned long signal;
    unsigned long blocked;      /* bitmap of masked signals */
    unsigned long flags;        /* per process flags, defined below */
    int errno;
    long debugreg[8];  /* Hardware debugging registers */
    struct exec_domain *exec_domain;
/* various fields */
    struct linux_binfmt *binfmt;
    struct task_struct *next_task, *prev_task;
    struct task_struct *next_run,  *prev_run;
    unsigned long saved_kernel_stack;
    unsigned long kernel_stack_page;
    int exit_code, exit_signal;
    /* ??? */
    unsigned long personality;
    int dumpable:1;
    int did_exec:1;
    /* shouldn't this be pid_t? */
    int pid;
    int pgrp;
    int tty_old_pgrp;
    int session;
    /* boolean value for session group leader */
    int leader;
    int   groups[NGROUPS];
    /*
     * pointers to (original) parent process, youngest child, younger sibling,
     * older sibling, respectively.  (p->father can be replaced with
     *  p->p_pptr->pid)
     */
    struct task_struct *p_opptr, *p_pptr, *p_cptr, *p_ysptr, *p_osptr;
    struct wait_queue *wait_chldexit;       /* for wait4() */
    unsigned short uid,euid,suid,fsuid;
    unsigned short gid,egid,sgid,fsgid;
    unsigned long timeout, policy, rt_priority;
    unsigned long it_real_value, it_prof_value, it_virt_value;
    unsigned long it_real_incr, it_prof_incr, it_virt_incr;
    struct timer_list real_timer;
```



```
    long utime, stime, cutime, cstime, start_time;
/* mm fault and swap info: this can arguably be seen as either mm-specific or thread-specific */
    unsigned long min_flt, maj_flt, nswap, cmin_flt, cmaj_flt, cnswap;
    int swappable:1;
    unsigned long swap_address;
    unsigned long old_maj_flt;      /* old value of maj_flt */
    unsigned long dec_flt;          /* page fault count of the last time */
    unsigned long swap_cnt;         /* number of pages to swap on next pass */
/*  limits  */
    struct rlimit rlim[RLIM_NLIMITS];
    unsigned short used_math;
    char comm[16];
/* file system info */
    int link_count;
    struct tty_struct *tty; /* NULL if no tty */
/* ipc stuff */
    struct sem_undo *semundo;
    struct sem_queue *semsleeping;
/* ldt for this task - used by Wine.  If NULL, default_ldt is used */
    struct desc_struct *ldt;
/* tss for this task */
    struct thread_struct tss;
/* filesystem information */
    struct fs_struct *fs;
/* memory management info */
    struct mm_struct *mm;
/* open file information */
    struct files_struct *files;
/* signal handlers */
    struct signal_struct *sig;
#ifdef __SMP__
    int processor;
    int last_processor;
    int lock_depth;     /* Lock depth. We can context switch in and out of holding a syscall kernel lock...
* /
#endif
};
```

Where the thread_struct contains the saved registers with other per thread information:

```
struct thread_struct {
    unsigned short   back_link,__blh;
    unsigned long esp0;
    unsigned short   ss0,__ss0h;
    unsigned long esp1;
    unsigned short   ss1,__ss1h;
    unsigned long esp2;
    unsigned short   ss2,__ss2h;
    unsigned long cr3;
    unsigned long eip;
    unsigned long eflags;
    unsigned long eax,ecx,edx,ebx;
    unsigned long esp;
    unsigned long ebp;
    unsigned long esi;
    unsigned long edi;
    unsigned short   es, __esh;
    unsigned short   cs, __csh;
    unsigned short   ss, __ssh;
    unsigned short   ds, __dsh;
    unsigned short   fs, __fsh;
    unsigned short   gs, __gsh;
```

```
    unsigned short    ldt, __ldth;
    unsigned short    trace, bitmap;
    unsigned long io_bitmap[IO_BITMAP_SIZE+1];
    unsigned long t r;
    unsigned long cr2, trap_no, error_code;
/* floating point info */
    union i387_union i387;
/* virtual 86 mode info */
    struct vm86_struct * vm86_info;
    unsigned long screen_bitmap;
    unsigned long v86flags, v86mask, v86mode;
};
```

And now for something completely different. This is a small program which gives you information on the threads running in a Java program.

```
// This example is from the book _Java in a Nutshell_ by David Flanagan.
// Written by David Flanagan.  Copyright (c) 1996 O'Reilly & Associates.
// You may study, use, modify, and distribute this example for any purpose.
// This example is provided WITHOUT WARRANTY either expressed or implied.

import java.io.*;

public class ThreadLister {
    // Display info about a thread.
    private static void print_thread_info(PrintStream out, Thread t,
                                          String indent) {
        if (t == null) return;
        out.println(indent + "Thread: " + t.getName() +
                    "  Priority: " + t.getPriority() +
                    (t.isDaemon()?" Daemon":"") +
                    (t.isAlive()?"":" Not Alive"));
    }

    // Display info about a thread group and its threads and groups
    private static void list_group(PrintStream out, ThreadGroup g,
                                   String indent) {
        if (g == null) return;
        int num_threads = g.activeCount();
        int num_groups = g.activeGroupCount();
        Thread[] threads = new Thread[num_threads];
        ThreadGroup[] groups = new ThreadGroup[num_groups];

        g.enumerate(threads, false);
        g.enumerate(groups, false);

        out.println(indent + "Thread Group: " + g.getName() +
                    "  Max Priority: " + g.getMaxPriority() +
                    (g.isDaemon()?" Daemon":""));

        for(int i = 0; i < num_threads; i++)
            print_thread_info(out, threads[i], indent + "    ");
        for(int i = 0; i < num_groups; i++)
            list_group(out, groups[i], indent + "    ");
    }

    // Find the root thread group and list it recursively
    public static void listAllThreads(PrintStream out) {
        ThreadGroup current_thread_group;
        ThreadGroup root_thread_group;
        ThreadGroup parent;
```



```
        // Get the current thread group
        current_thread_group = Thread.currentThread().getThreadGroup();

        // Now go find the root thread group
        root_thread_group = current_thread_group;
        parent = root_thread_group.getParent();
        while(parent != null) {
            root_thread_group = parent;
            parent = parent.getParent();
        }

        // And list it, recursively
        list_group(out, root_thread_group, "");
    }


    public static void main(String[] args) {
        ThreadLister.listAllThreads(System.out);
    }
}
```