

# CSE 306/506 Operating Systems

## Process Management in Linux

YoungMin Kwon

# The Process

- A process includes
  - Code in text section
  - Global variables in data section
  - Resources like open files, pending signals
  - Internal kernel data
  - Processor state
  - Memory address space with memory mappings
  - One or more thread of execution

# The Processes

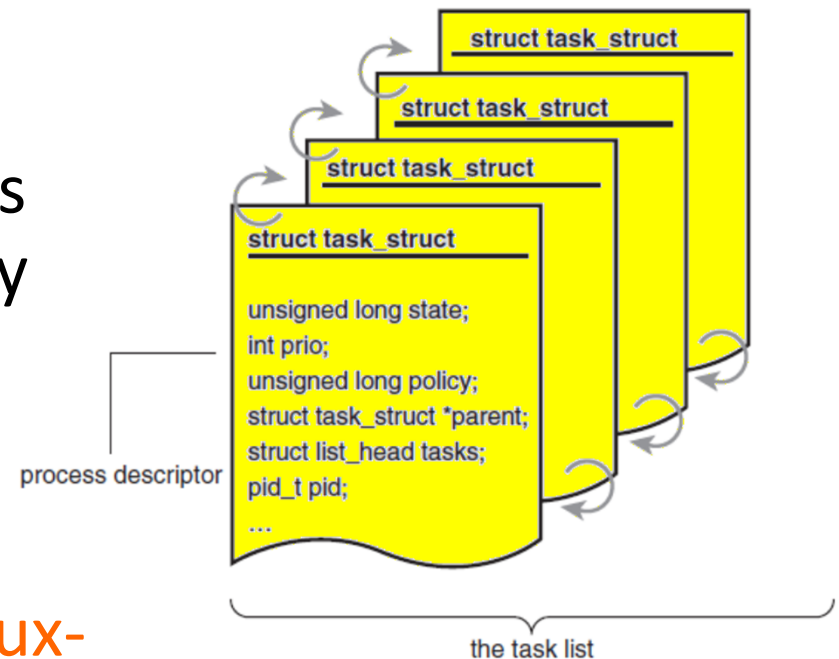
- Processes provide
  - A virtualized processor
    - Gives a process the illusion that it is monopolizing the system
  - A virtual memory
    - Let a process allocate and manage memory as if it alone owned all memory in the system

# The Processes

- In Linux
  - `fork()` creates a process
  - `exec()` family creates new address space and loads a new program
  - `exit()` terminates the process and frees its resources
  - `wait()` inquires the status of a terminated child

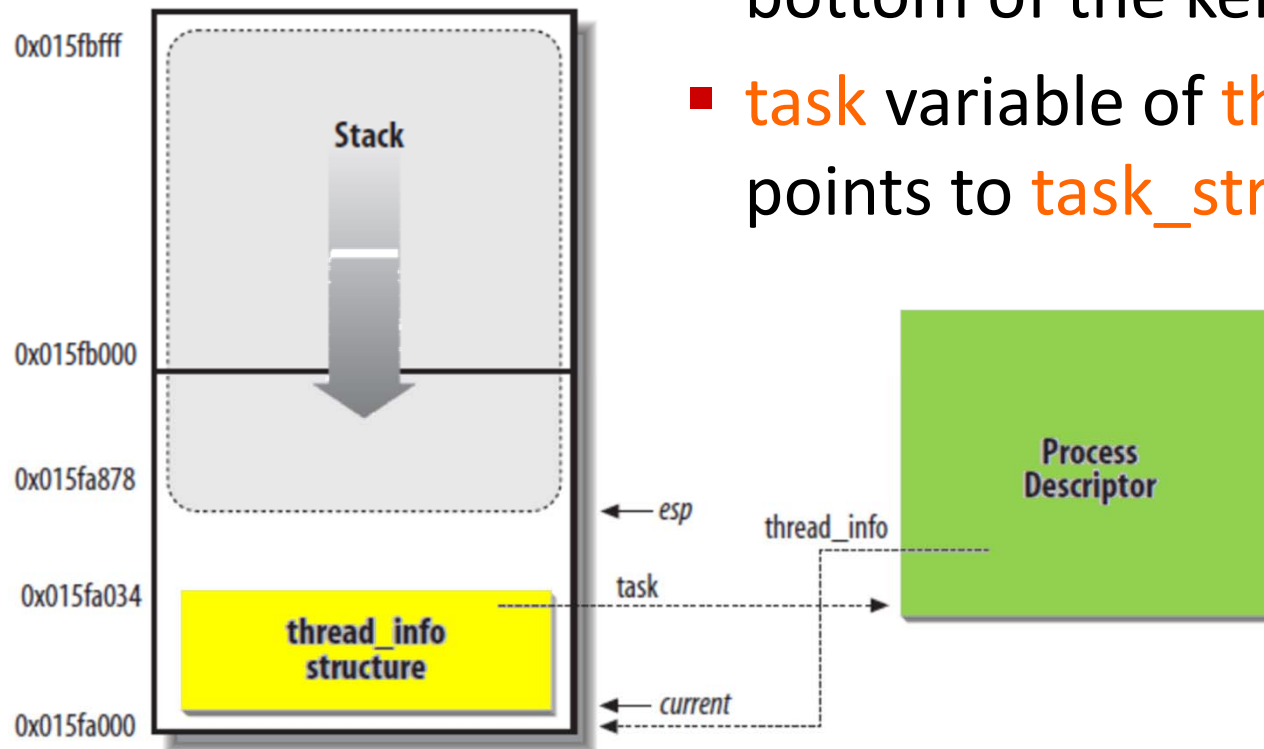
# Process Descriptor

- Task list
  - List of process descriptors stored in a circular doubly linked list
- Process descriptor
  - `task_struct` defined in `linux-5.4.49/include/linux/sched.h`
  - Contains all the information about a process



# Process Descriptor

- Locating process descriptors
  - `thread_info` structure is at the bottom of the kernel stack
  - `task` variable of `thread_info` points to `task_struct`



# Process Descriptor

- `current_thread_info()`

```
movl $-8192 %eax  
andl %esp, %eax
```

-8192 in binary  
1...1 1110 0000 0000 0000

- `current` macro is equivalent to  
`current_thread_info()->task`

```
union thread_union {  
    struct thread_info thread_info;  
    unsigned long stack[THREAD_SIZE / sizeof(long)];  
};
```

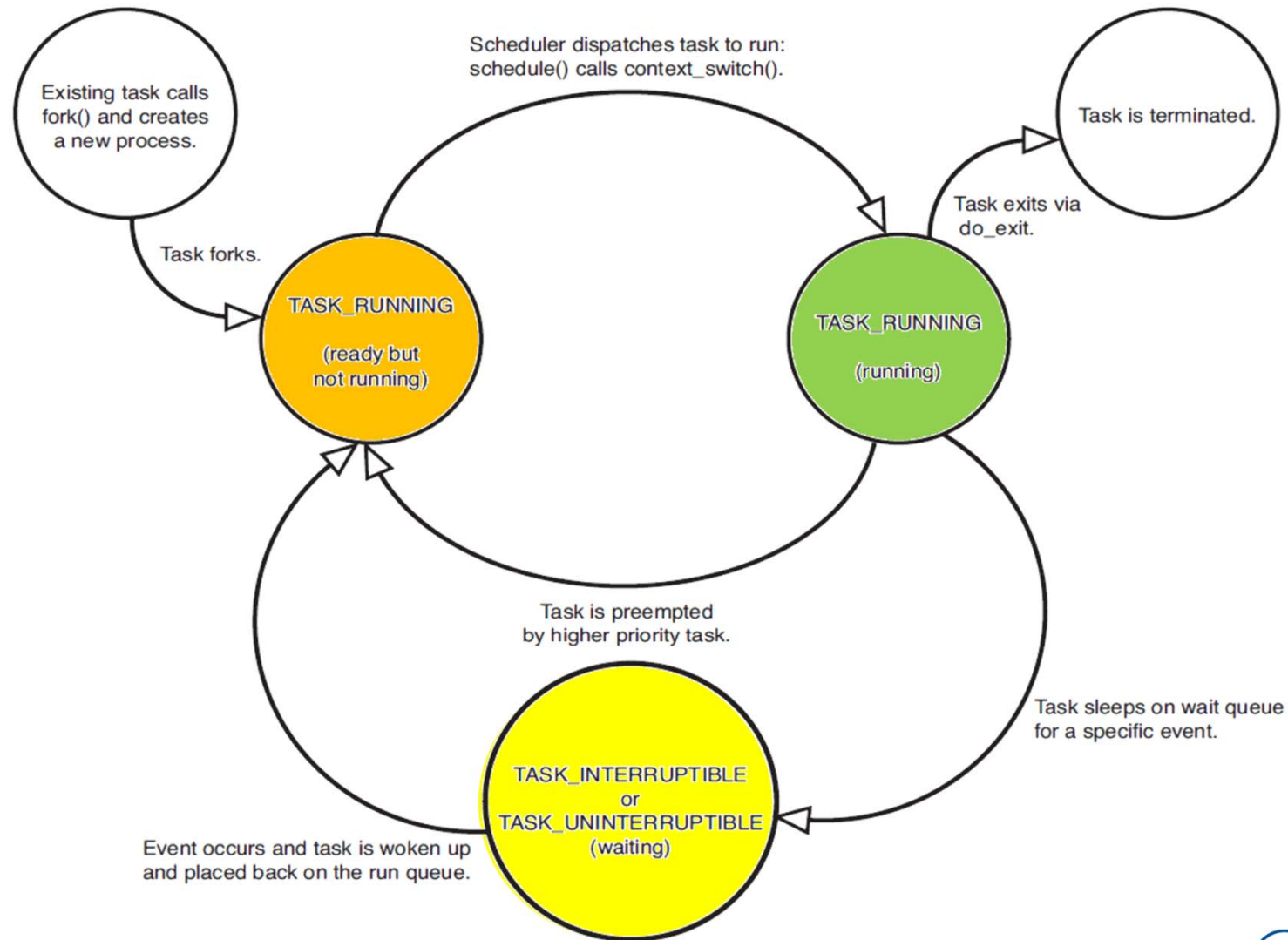
# Process State

- The **state** field of **task\_struct**
  - TASK\_RUNNING: currently running or on a run-queue
  - TASK\_INTERRUPTABLE: blocked and waiting for some condition to exist
    - On receiving a **signal**, can wake up prematurely
  - TASK\_UNINTERRUPTABLE: blocked and waiting for some condition to exist
    - Does not wake up when it receives a **signal**
  - \_\_TASK\_TRACED: the process is being traced by another process (debugger)
  - \_\_TASK\_STOPPED: the process is not running and is not eligible to run

`set_current_state(state)` //changes the state of the current task



# Process State



# Process Context

- Process context
  - Program execution in kernel space on behalf of the process
  - Through a **system call** or an **exception**
  - **current** macro is valid
- Interrupt context
  - Interrupt handler handling an **interrupt**

# Process Family Tree

- **init** process
  - Kernel starts **init** in the last step of the boot process
  - **init** reads initscripts and execute more programs
- Family tree
  - Each process has exactly one parent
  - A process has zero, one, or more children
  - Child processes whose parent are the same are called siblings

# Process Family Tree

- Related links in task\_struct

```
struct task_struct {  
    ...  
    struct task_struct __rcu *real_parent; //real parent process  
    struct task_struct __rcu *parent; //recipient of SIGCHLD, wait4() reports  
  
    struct list_head children; //list of my children  
    struct list_head sibling; //linkage in my parent's children list  
  
    ...  
};
```

- To iterate over child process descriptors

```
struct task_struct *task;  
struct list_head *list;  
list_for_each(list, &current->children) {  
    task = list_entry(list, struct task_struct, sibling);  
    /*task now points to one of current's children */  
}
```

# Process Creation

- `copy_process()` within `fork()` (kernel/fork.c)
  - `dupe_task_struct()`
    - Create `kernel stack`, `thread_info`, `task_struct`
  - Variables of `task_struct` are cleared or initialized
  - Child's `state` is set to `TASK_UNINTERRUPTABLE`
  - Update `flags` member
    - `PF_SUPERPRIV` is cleared, `PF_FORKNOEXEC` is set
  - Assign a `new PID` to the child
  - Duplicate or share
    - Open files, filesystem information, signal handlers, process address space, and namespace

# Process Creation

- Copy-on-Write (COW)
  - A technique to delay or prevent copying of data
  - Rather than duplicate the process address space, parent and child share a single copy
  - When data is actually written, each process receives a unique copy
- Overhead of `fork()`
  - Duplication of `page table`, creation of a `process descriptor (task_struct)`

# Process Termination

- `do_exit()` in `kernel/exit.c`
  - Sets `PF_EXITING` in the `flags` of `task_struct`
  - Calls `del_timer_sync()` to remove any kernel timers
  - Calls `acct_update_integrals()` to write out accounting info
  - Calls `exit_mm()`, `exit_files()`, and `exit_fs()` to release the objects
  - Sets the `exit_code` member of `task_struct`
  - Calls `exit_notify()` to send signals to the task's parent
  - Calls `schedule()` to switch to new process

# Process Termination

- After `do_exit`
  - The `process descriptor` for the terminated process still exists
  - The process is a zombie and is unable to run
  - `wait()` family of functions get the exit code and destroy the process descriptor



# Kernel Threads

- Kernel threads
  - For the kernel to perform operations in the background
  - Kernel threads are a process
    - Schedulable, Preemptable
  - Kernel threads don't have a user address space
    - mm pointer of task\_struct is NULL
    - Operates only in kernel-space

# Kernel Threads

- Some interfaces in linux/kthread.h

```
struct task_struct *kthread_create_on_node(int(*threadfn)(void *data),
    void *data,
    int node,
    const char namefmt[], ...);
```

```
#define kthread_run(threadfn, data, namefmt, ...) \
({ \
    struct task_struct *__k \
        = kthread_create(threadfn, data, namefmt, ## __VA_ARGS__); \
    if (!IS_ERR(__k)) \
        wake_up_process(__k); /*start the task*/ \
    __k; \
})
```

```
int kthread_stop(struct task_struct *k);
```

# Example: exploring task\_structs

```
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/init_task.h>
#include <linux/list.h>
#include <linux/fs.h>
#include <linux/fs_struct.h>
#include <linux/kthread.h>
#include <linux/semaphore.h>

static void print_task(struct task_struct *task, int depth)
{
    char buf[100];

    printk("%*c%5d:%s\n",
           depth*4, '+',
           task->pid,
           d_path(&task->fs->pwd, buf, sizeof(buf)));
}
```

```
static void print_family(struct task_struct *task, int depth)
{
    struct list_head *pos;
    print_task(task, depth);          //self

    list_for_each(pos, &task->children) {
        struct task_struct *t;
        //t = list_entry(pos, struct task_struct, sibling);
        t = container_of(pos, struct task_struct, sibling);
        print_task(t, depth + 1); //child
    }
}
```

```

static int threadfn(void *data) {
    int pid = (long)data;
    struct list_head *head = &init_task.tasks;
    struct list_head *pos;
    for (pos = head; pos->next != head; pos = pos->next) {
        struct task_struct *task = container_of(
            pos->next, struct task_struct, tasks);
        if (task->pid == pid) {
            print_family(task, 0);
            return 0;
        }
    }
    printk("pid %d not found\n", pid);
    return -1;
}

SYSCALL_DEFINE1(print_family, int, pid) {
    kthread_run(threadfn, (void*)(long)pid, "printfamily_%d", 0);
    return 0;
}

```

# User-Space Program

```
//wrapper.c
...
#define __NR_print_family 453
...
long print_family(int pid)
{
    return syscall(__NR_print_family, pid);
}

//print_family.c
...
int main()
{
    long res = print_family(1);
    printf("%ld\n", res);
}
```

```
ykwon4@youngbox2:~/ home$dmesg
```

```
...
```

```
[ 162.737252] + 1:/  
[ 162.737257] + 982:/  
[ 162.737259] + 1000:/  
[ 162.737262] + 1602:/
```

```
...
```

```
[ 162.737270] + 1837:/  
[ 162.737273] + 1843:/var/spool/cron  
[ 162.737275] + 1848:/
```

```
...
```

```
[ 162.737291] + 1979:/  
[ 162.737293] + 2102:/etc/avahi  
[ 162.737295] + 2122:/  
[ 162.737297] + 2180:/  
[ 162.737299] + 2199:/var/lib/lightdm  
[ 162.737301] + 2203:/  
[ 162.737303] + 2213:/var/lib/lightdm  
[ 162.737304] + 2222:/  
[ 162.737306] + 2230:/var/lib/lightdm  
[ 162.737308] + 2232:/var/lib/lightdm  
[ 162.737310] + 2234:/var/lib/lightdm  
[ 162.737313] + 2289:/proc  
[ 162.737315] + 2288:/  
[ 162.737317] + 2312:/  
[ 162.737319] + 2319:/
```

# Processes and Threads

- Two characteristics of a process
  - Resource ownership
    - Virtual address space (program, data, stack, PCB...)
    - Main memory, I/O devices, files
  - Scheduling/execution
    - Execution of a process follows an execution path
    - Execution may be interleaved with that of other processes



# Processes and Threads

- Resource ownership and Scheduling/execution
  - Could be treated independently by the OS
- Thread (lightweight process): the unit of dispatching
- Process (task): the unit of resource ownership

# Why Threads

- Parallel execution
  - Without relying on interrupts, timers, context switches
  - Parallel entities sharing an address space and data
- Easier and faster to create and destroy than processes
  - 10 ~ 100 times faster
- Performance gain
  - Not much for CPU bounded applications
  - Substantial for I/O bounded applications
- Real parallelism with multiple CPUs

# Multithreading

- Threads within a process have
  - Thread execution state (Running, Ready, ...)
  - Saved thread context (PC, registers, ...)
  - Execution stack
  - Per-thread static storage for local variables
  - Access to resources shared with other threads in the process

# Multithreading

- All threads of a process share the states and resources of the process
  - If a thread alters data, other threads will see the change
  - If a thread opens a file with a read privilege, others can also read the file

Per-process items	Per-thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

# POSIX Threads (Pthreads)

- A standard interface for manipulating threads from C programs
- Defines about 60 functions that allow programs
  - to create, kill, and reap threads
  - to share data safely with peer threads
  - to notify peers about changes in the system state

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run

# Threads in Linux

- Threads of a process
  - Share memory address space
  - Share open files and other resources
- Thread in Linux
  - Threads are a process that share certain resources with other processes
  - Each thread has a unique task\_struct and appears to the kernel as a normal process

# Thread in Linux

## ■ Creating a thread

```
clone(CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND, 0);
```

```
#define CLONE_VM      0x00000100 //set if VM shared between processes
#define CLONE_FS      0x00000200 //set if fs info shared between processes
#define CLONE_FILES   0x00000400 //set if open files shared between processes
#define CLONE_SIGHAND 0x00000800 //set if signal handlers and blocked signals shared
...
```

# Assignment 2

- Write a system call that returns the whole family tree from the **init** process

```
SYSCALL_DEFINE2(print_tree, char*, buf, int, buflen);
```

- The output is returned in **buf** as a string
- For each process, the corresponding line in the output string should include
  - PID (**task->pid**)
  - tty name (**task->signal->tty->name**)
  - total elapsed time in sec.msec format  
( **task\_cputime(task, &utime, &stime);**  
    **msec = (utime+stime) / 1000000; )**
  - process name (**task->comm**)
- Each line should be indented by the depth of the node in the tree



# Assignment 2

- A pointer to the `init` task descriptor is `&init_task`
- Visit the child tasks in the depth-first traversal manner
  - Recursion may not work due to the limited kernel stack size

```
static void print_tree(struct task_struct *task, int depth)
{
    struct list_head *pos;
    printk("%*d\n", depth*4, task->pid)

    list_for_each(pos, &task->children) {
        struct task_struct *child;
        child = list_entry(pos, struct task_struct, sibling);

        print_tree(child, depth + 1);
    }
}
```

- Submit the files you changed or added
  - Mark your change with `//CSE306` tag
- Due date: 10/13/2020

# Assignment 2

- To remove recursion, use a stack explicitly

```
struct task_frame {
    struct task_struct *task;
    int depth;
};
static struct task_frame frame_stack[10000];
static int frame_sp = 0;
static void push_frame(struct task_struct *task, int depth);
static void pop_frame(struct task_struct **task, int *depth);
```

- Make print\_tree a thread function

```
struct ptree_param {
    struct task_struct *task;
    struct semaphore *sem;
    int buflen;
    char *buf;
};
static int print_tree(void *data) { //thread function
    struct ptree_param *p = (struct ptree_param*) data;
    ...
```

# Assignment 2

- In `sys_print_tree` (system call handler),
  - Run `print_tree` in a kernel thread
  - Wait for the thread to finish using a semaphore

```
struct semaphore ...;
```

```
void sema_init(struct semaphore *sem, int val);
```

```
void down(struct semaphore *sem);
```

```
void up(struct semaphore *sem);
```

# Assignment 2

- Header files to include

```
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/init_task.h>
#include <linux/list.h>
#include <linux/fs.h>
#include <linux/kthread.h>
#include <linux/semaphore.h>
#include <linux/mm.h>
#include <linux/dcache.h>
#include <linux/sched/cputime.h>
#include <linux/slab.h>
#include <linux/tty.h>
#include <linux/uaccess.h>
```

# Assignment 2 (user-space program)

```
//wrapper.c
#define __NR_print_tree 454

long print_tree (char *buf, int buflen) {
    return syscall(__NR_print_tree, buf, buflen);
}

//print_tree.c
int main() {
    char buf[4096];
    long res = print_tree(buf, sizeof(buf));
    if (res)
        printf("%ld\n", res);
    else
        printf("%ld\n%s\n", res, buf);
}
```

# Assignment 2 (sample output)

```
ykwon4@youngbox2:~/home$ ./a.out
0
+ 0::0.104:swapper/0
+ 1::1.244:systemd
+ 987::0.110:systemd-journal
+ 1829::0.23:avahi-daemon
+ 1842::0.0:avahi-daemon
+ 1832::0.6:acpid
+ 1961::0.21:lightdm
+ 1967:tty7:0.514:Xorg
+ 2053::0.10:lightdm
+ 2108::0.1:lightdm-greeter
+ 2142::0.335:unity-greeter
+ 2194::0.3:lightdm
+ 1966::0.6:systemd-hostnam
+ 1970:tty1:0.3:agetty
+ 1976:ttyS0:0.23:login
+ 2349:ttyS0:0.67:bash
+ 2366:ttyS0:0.1:a.out
+ 2063::0.17:systemd
+ 2077::0.0:(sd-pam)
```