COMSW4118-6-1 9/21/2017

- Linux kernel **3.10** ~ **10MM** lines of code.
- **System calls** - API that the kernel exposes to user space. User space asks the kernel to perform some action on its behalf.
- **syscall** - function in the standard C library to invoke a system call by number (normally system calls are invoked by wrappers in the std. C lib)
- What to do to implement a system call in the Linux kernel?

  ① Define a system call number: "unistd" is usually name for POSIX stuff.
  /arch/arm/include/uapi/asm/**unistd.h**

  ② Add an entry to the system call table.
  /arch/arm/kernel/**calls.S**

  **entry.S** not used anymore here, **calls.S** is used

  ③ Add to the generic headers to the kernel the new function prototype:
  /include/linux/**syscalls.h** ← non-architecture specific stuff

  Ex: **asmlinkage** long sys_open(const char **__user** *filename, int flags, umode_t mode);

  meaning args are on stack, not in registers

  following argument is a variable sourced from user memory, i.e. NOT to be trusted by the kernel (usually copied to kernel space asap).

  ④ Add system call implementation:
  Ex: /kernel/**exit.c** ← note this is diff than definition:

  SYSCALL_DEFINE1(exit, int, code) { .... }

  # of arguments

  **do_exit**(code) ← convention to actually implement in do*

  ⑤ Modify build Makefile to include new *.c implementation file
  Add *.o to obj-y.

- Kernel conventions:
  - ▶ Linux kernel does not use standard C library (too large, written for user space calls syscalls) — no printf, no malloc, etc.
  - ▶ stack and heap in userspace have a lot of memory
    In the kernel, the stack is very small - never use recursion
  - ▶ kernel versions of some common functions: kmalloc (with arg where to allocate, ex. GFP_KERNEL) kfree, printk, kprintf

---

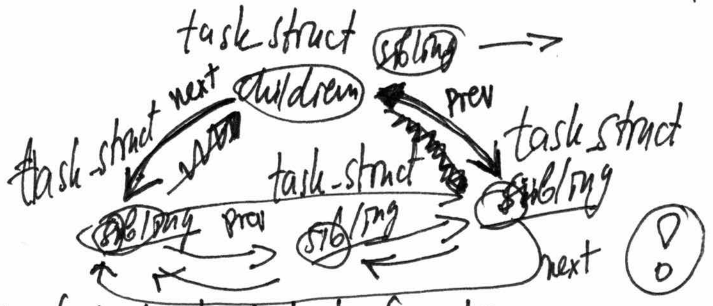
- **Processes** ← exists in the Linux kernel
  - ▶ Has a data structure, associated with it:
    **task_struct** (400+ lines!)

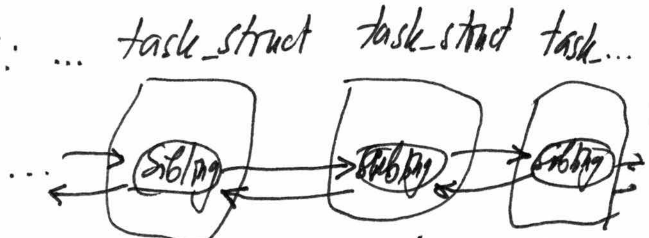
COMSW4118 - 6 - 2 9/21/2017

real_parent is who spawned, parent is current

▶ **task_struct**: parent, children, sibling

NOTE: List-head is a member of the list itself:





Parents children is in the leftmost child's sibling circle

• Some useful linked-list functions:

LIST_HEAD(list) for static or local vars, INIT_LIST_HEAD(&list) for dynamic vars (list needs to be declared separ)

list_entry(ptr, type, member), list_empty(head), list_is_last(ptr, head), list_first_entry(ptr, type, member)

list_for_each[-prev], list_for_each_entry_safe
list_for_each_entry, list_for_each_safe.

▶ How does fork() work?
- kernel makes a copy of invoking process and executes it as new process.
- check out the do_fork implementation: invokes copy_process
  note some macros used like ERR_PTR ← sets errno
- /include/uapi/asm-generic/errno-base.h ← lists errno values.

▶ task_struct list is guarded by tasklist_lock
init_task is the root task

■