Overview

- This Lecture
  - Processes
  - Source: ULK ch 3, ch 7 & ch 9

Why processes?

- Isolation
  - Each process has a private memory area for code, stack, and data
- Protection
  - Each process can't read/write outside its address space
- Sharing is allowed
- Maintain multiple concurrent tasks

Process management

- Related system calls
  - fork(), exec()
  - exit(): release the resources and send its parent a SIGCHLD signal
- Zombie processes
  - Terminated process, but its parent hasn't called wait() yet to collect the process' status
- Process groups and login sessions

User/kernel modes

- How modes changed?
  - System call, interrupt, exception
- Registers saved at context switch
  - PC and SP
  - General purpose registers
  - Floating point registers
  - Processor status word
  - Memory management registers
- Process address space
  - Stack, code, data (can be shared using mmap)

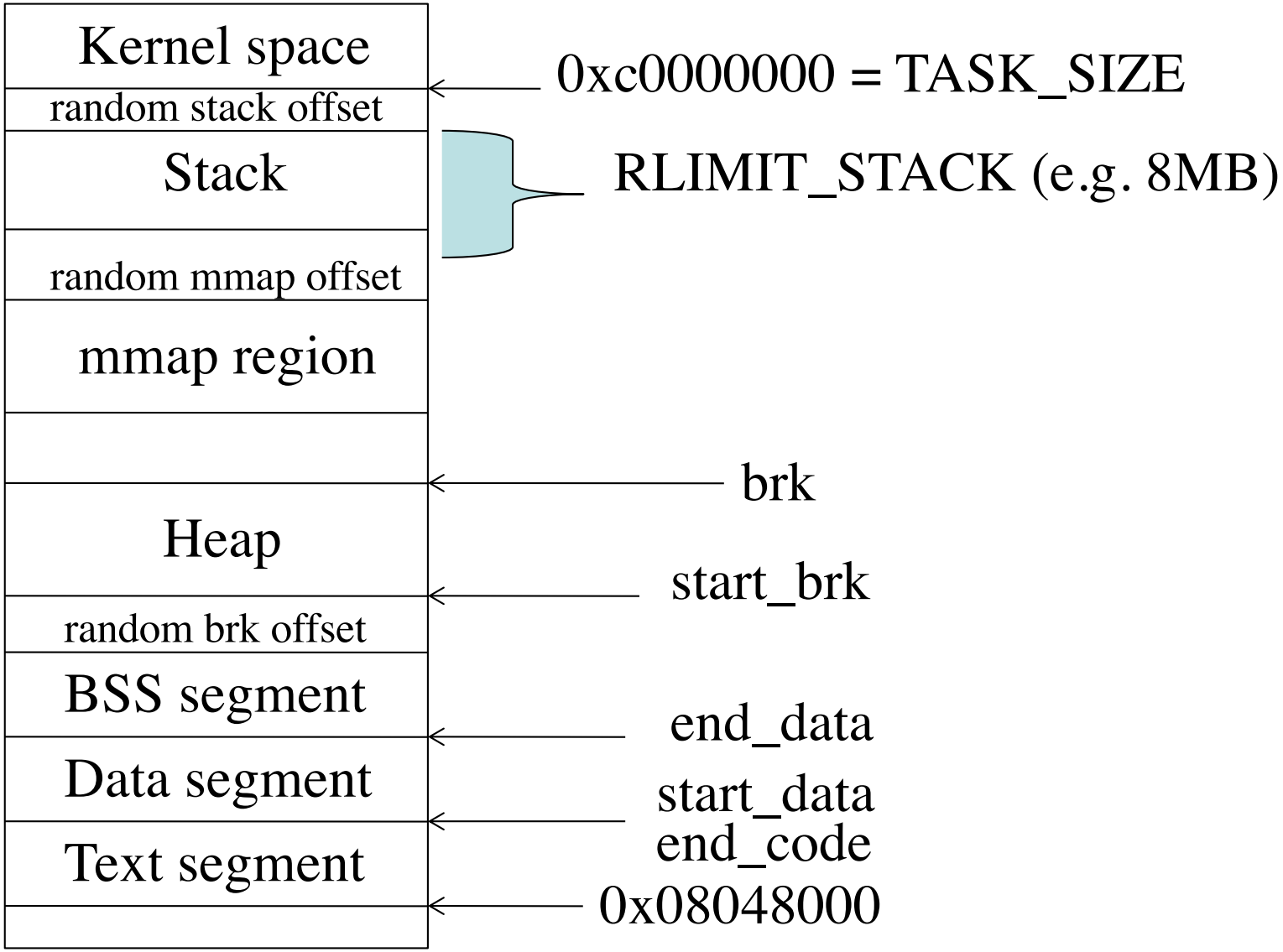
Process implementation

- Implemented by both OS and hardware
- OS manages processes
  - Allocate physical memory
  - Keep track of the process status
  - Schedule/switch between processes
- Hardware
  - Perform address translation, protection
  - Assist user/kernel transfer (syscalls, interrupts)

Process virtual address space

- A list of memory area descriptors
  - Program code
  - Initialized data
  - Uninitialized data
  - The heap
  - Code and data of shared libraries
  - Program stack
- Demand paging and swapping
  - No physical pages allocated to a process initially.
- Copy-On-Write(COW) for process forking

Layout of process address space



Process address space

- A memory descriptor is used to describe the address space
- Process address space consists of memory regions
- Memory regions are organized as red-black trees
  - Access rights are applicable to individual regions

Memory regions (mma)

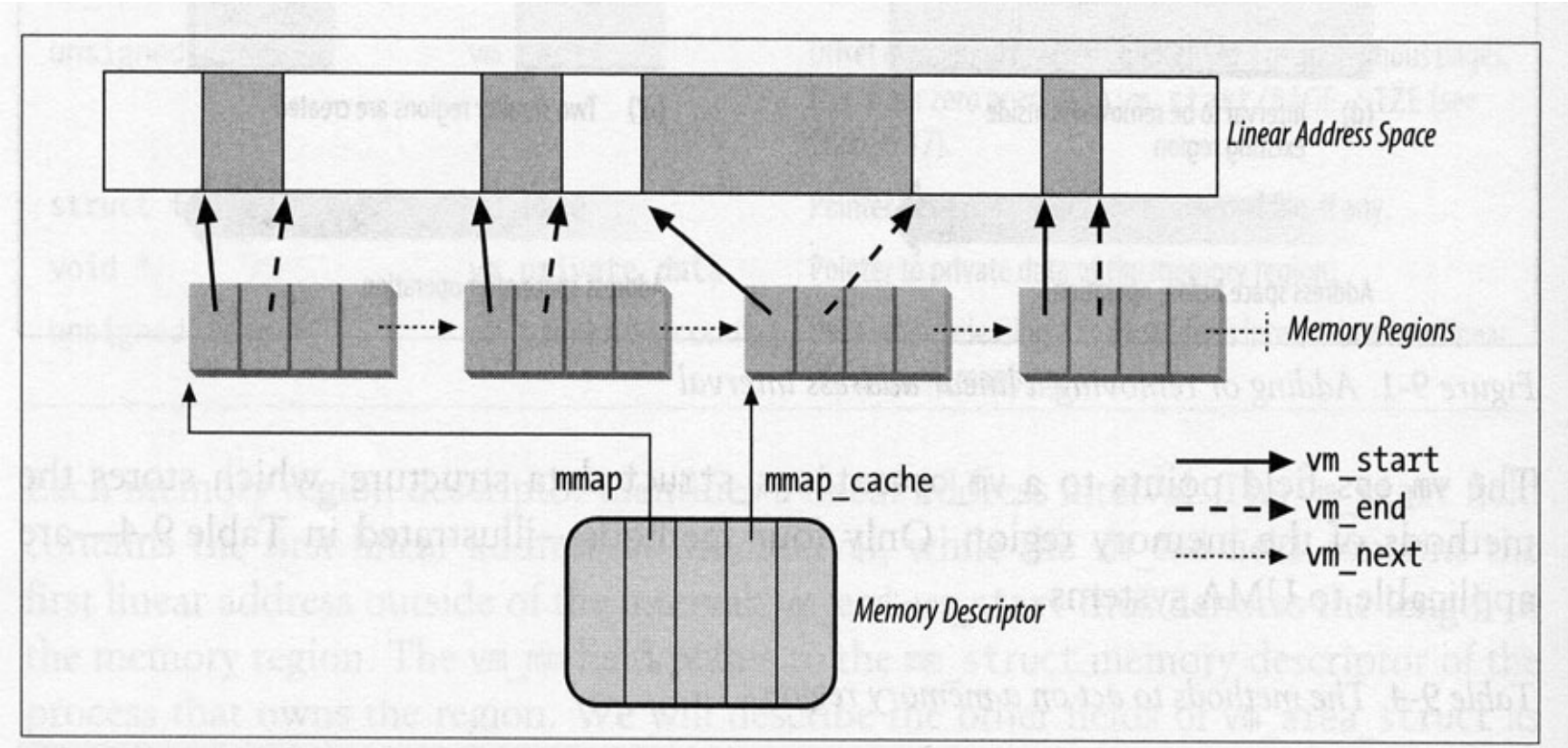


Figure 9-2. Descriptors related to the address space of a process

mmap

- What happens if a memory region is mapped to a device (or file)?
  - mmap method of the device driver is called
  - Operations to act on a memory region
    - open, close, nopage
  - Very useful for writing a RAM device driver

Memory management

- Virtual memory
  - A logical layer between application memory requests and MMU
- Why using virtual memory?
- After kernel initialization, the available memory is used by virtual memory system
  - Kernel buffers, process memory requests, caches, etc
- Memory fragmentation
- Kernel Memory Allocator (KMA)
  - Based on allocating algorithms

Protection for address space

- x86 has two mechanisms for protection
  - Segmentation
  - Paging
- Linux uses simple segmentation, but supports paging extensively
- Linux has its own linear address space (top 1G in 32-bit), but can access each process' address space (lower 3G in 32-bit).
  - Now 32K GB as kernel address space on 64-bit

Processes and threads

- Processes
- Threads
  - An execution flow of a process
  - Share everything with the process except the stack
- Light-weight processes
  - A process sharing some data structures with its parent. The extent of sharing can be decided by options

Process Descriptor (PD)

- Also called process control block
- Fields in process descriptor
  - Process state: TASK_RUNNING, TASK_INTERRUPTIBLE, …
  - Process id, thread id
  - Memory area descriptors
  - File descriptors
  - Signals
  - Terminal
  - Various links

Linux PD

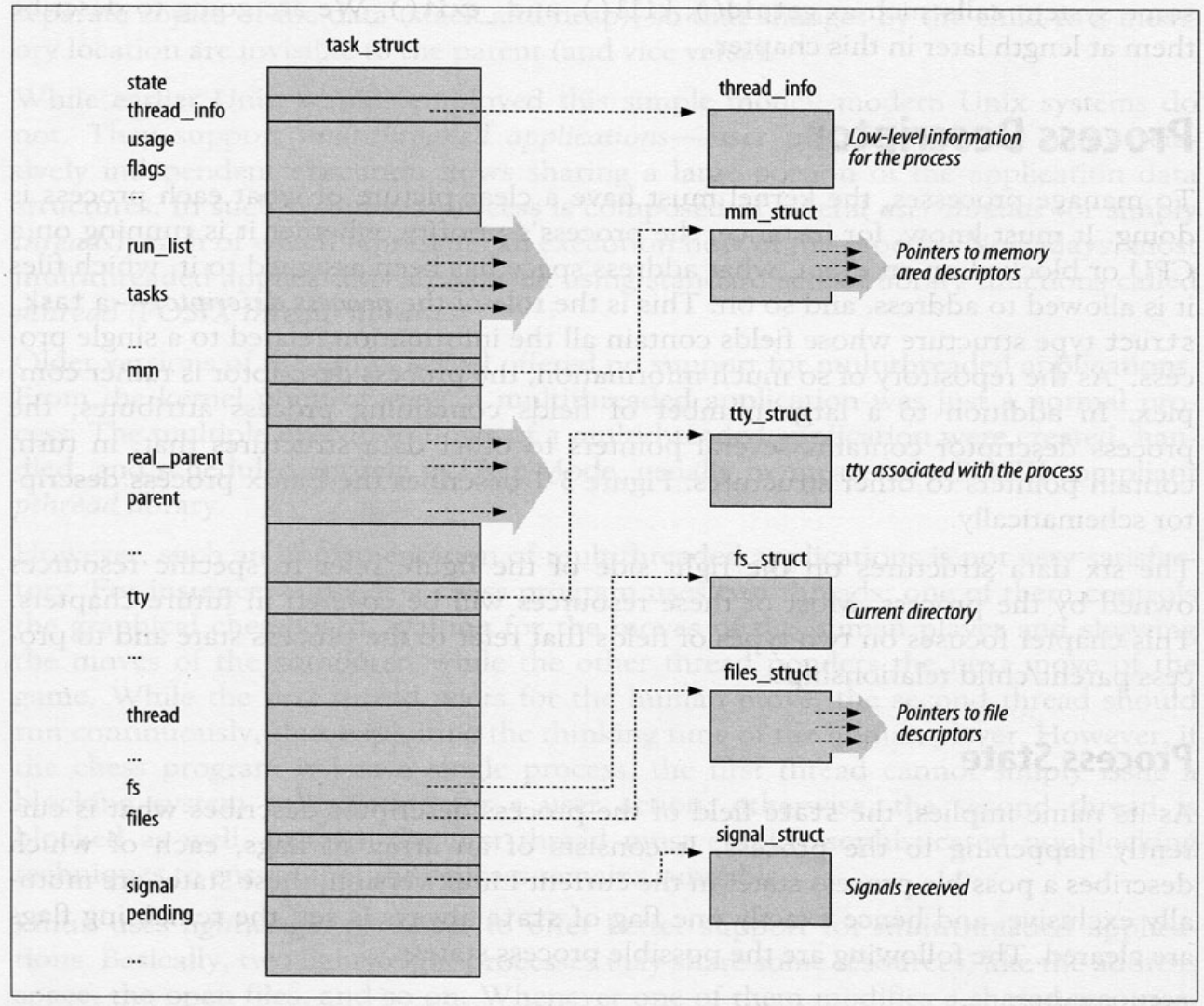


Figure 3-1. The Linux process descriptor

Process descriptor handling

- 8KB (2 pages) memory block (aligned with 8KB)
- How to get a process descriptor's pointer?
- Layout

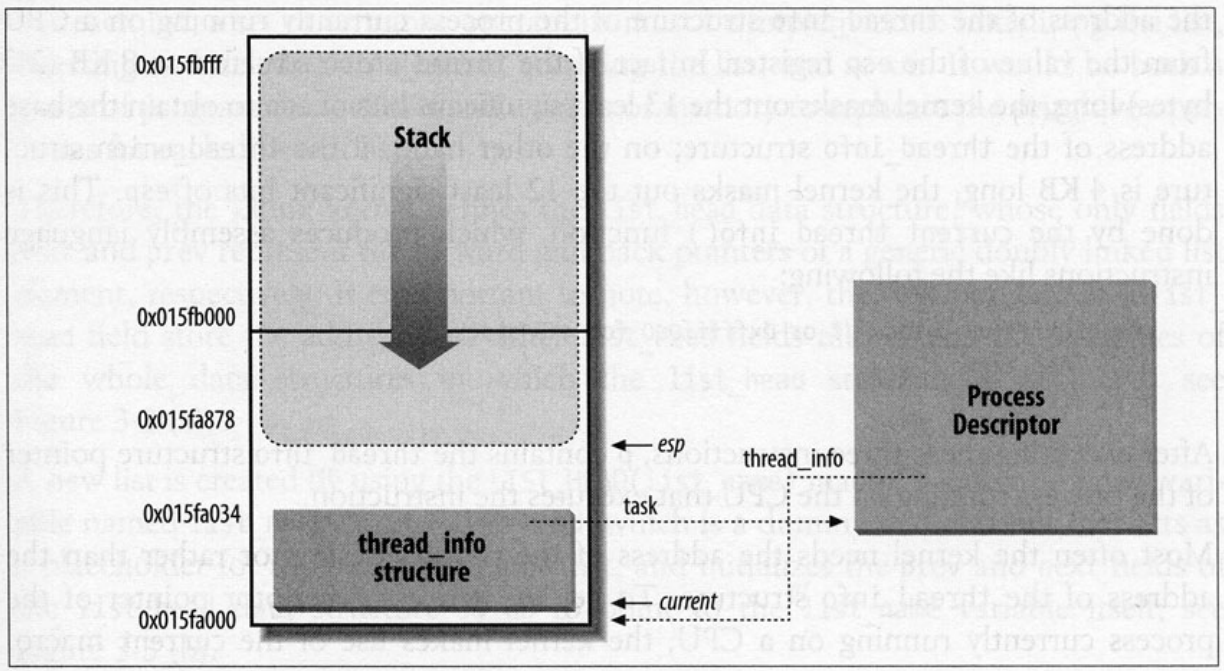


Figure 3-2. Storing the thread_info structure and the process kernel stack in two page frames

Process list

- Doubly linked list
  - list_head structure: next, prev
  - Related functions and macros

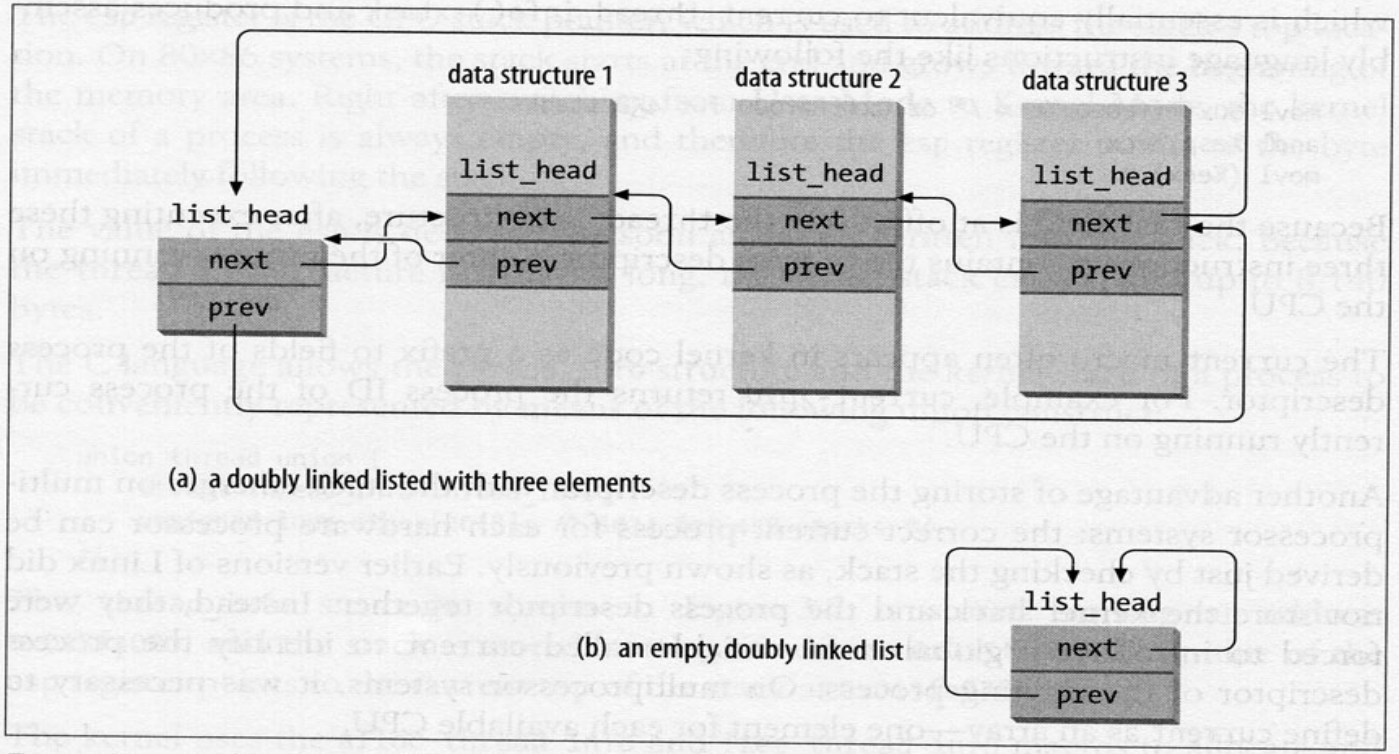


Figure 3-3. Doubly linked lists built with list_head data structures

Handling process list

- List of TASK_RUNNING processes
  - The queue is called *runqueue*
  - The list head is the *init_task* PD
  - The *run_list* field in the PD is used
- How a process is scheduled to run?
  - Related functions: wake_up_process
- How to quickly find a PD with a pid?
  - pidhash table

Parenthood relationships

- Useful pointers
  - Real parent, Parent, Child, Younger sibling, Older sibling

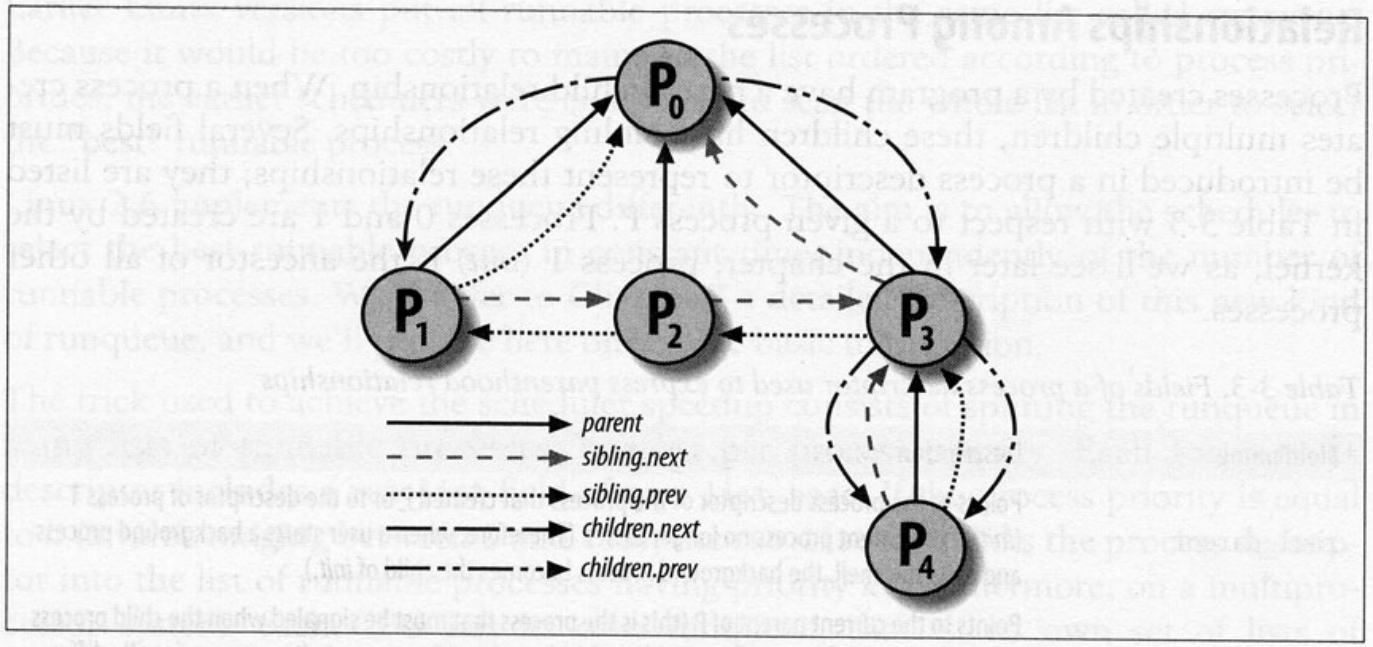


Figure 3-4. Parenthood relationships among five processes

COS

Wait queues

- Used to suspend processes for many purposes
- Related data structures
  - Queue head
  - Link element
- Exclusive and non-exclusive processes
- Related functions (very important)
  - *wait_event()*, *wake_up()*, etc

How a process wait for a device?

```
void sleep_on(wait_queue_head_t *wq)
{
    wait_queue_t wait;
    init_waitqueue_entry(&wait, current);
    current->state = TASK_UNINTERRUPTIBLE;
    add_wait_queue(wq,&wait); /*wq points to the wait queue head*/
    schedule();
    remove_wait_queue(wq, &wait);
}
```

Process resource limits

- Various resource limits
  - Maximum address space
  - Maximum core dump file size
  - Maximum CPU time
  - Maximum heap size
  - Maximum file size
  - Maximum stack size
  - Maximum number of page frames
  - …

Process switch

- Also called task switch or context switch
  - Suspend the execution of the process running on the CPU and resume the execution of some other process previously suspended
- Hardware context
  - A set of registers

Process handling

- Process creation
  - clone(), fork(), vfork()
  - Understanding do_fork()
- Process termination
  - exit()
  - Understanding do_exit()
- Process removal
  - How zombies are removed?

Scheduling

- Many theoretical work done,
  - Strict priority, round-robin, shortest job first, minimum guarantee with admission control
- Scheduling seems uninteresting
  - When resources are not in shortage
  - Except the web servers and large-scale networks that cannot handle peak demand or some scheduling decisions have non-linear effects on overall system behavior (read the *Eliminating Receive Livelock* article for today)

Key problems in scheduling

- Gap between desired policy and implementing mechanism
  - Scheduler can approximate policy
- Conflicting goals
  - Low latency, high throughput, fairness, etc
- Interaction between difference schedulers
  - Only optimizing CPU scheduler may have little impact on overall performance

Scheduling in Linux

- Scheduling policy
  - Preemptable between processes
  - Kernel threads have higher priority
  - NAPI for network tasks as in the *livelock* paper
- Quantum
  - How long must be a Quantum last?
- The *schedule* function

Kernel programming

- Application vs kernel programming
  - Applications use libc and other libraries
  - Kernel code can only use kernel functions
- Reentrant
  - The same function called by different processes/threads
  - Better no global variables
- Data race and mutual exclusion
  - Interrupt disabling
  - Semaphores: down() and up()
  - Spin locks
  - deadlock

The *livelock* paper

- Background
  - Uniprocessor, no spinlocks
  - Different contexts
    - Process (user half, kernel half)
    - Soft interrupts (bottom half)
    - Device (hard) interrupts
  - How network frames were handled?
  - What solution was proposed in the paper?