Linux Case Study

OPS Lecture 17, G53OPS/G52OSC

Geert De Maere
(Jason Atkin – OSC)
Geert.DeMaere@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2015

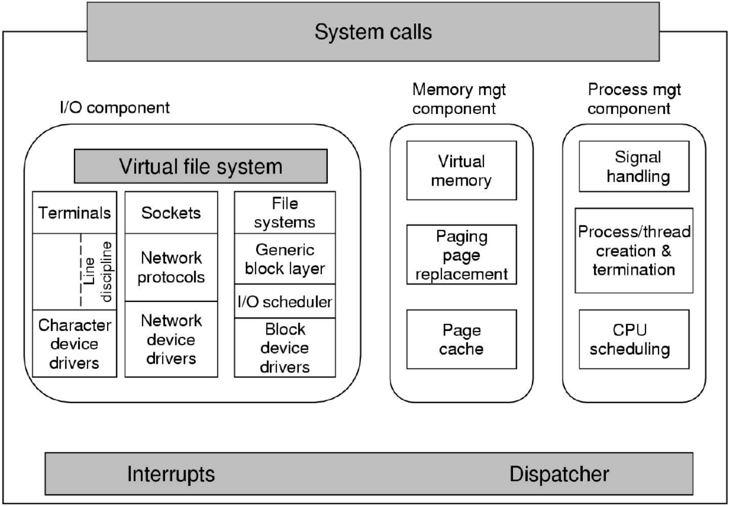

Figure: Structure of the Linux Kernel (Tanenbaum)

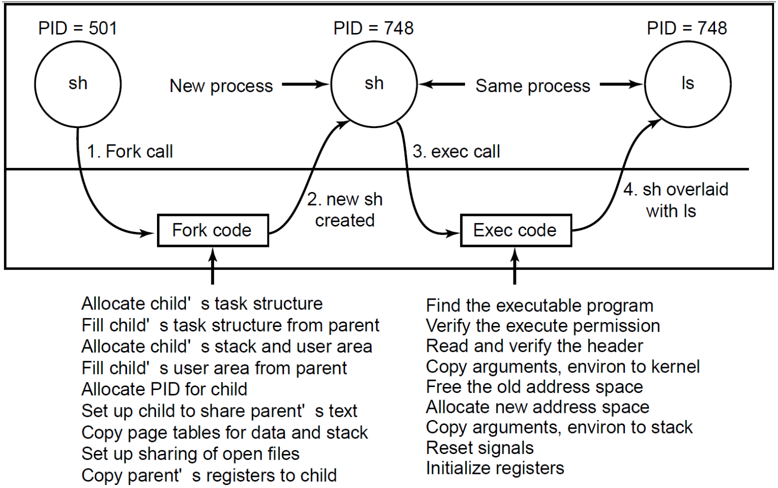

Figure: Process Creation in Linux (Tanenbaum)

- **Two approaches** exist for a parent process to **create** a child process
  - The traditional **Unix approach** using `fork()` and `exec()`
  - The **Linux specific approach** using `clone()` which blurs the boundaries between processes and threads by allowing **fine grained control** over memory, files, etc. and
- **Processes** and **threads** are both called **tasks** in Linux

- The process table (containing task structures) is implemented as a **doubly linked list** with a **mapping** based on the PID
- A **task structure** in Linux is used for processes as well as threads, and includes:
  - They contain, e.g., process/thread **scheduling information**, open **file table**, **memory information**, registers, etc.
  - Some of this information is contained in **separate structures** to which the task structure holds a pointer

- Traditionally, **threads share resources** (address space, global variables, files, etc.) with the process, but have their **own program counter**, **registers**, **stack**, **states**, and **local variables**
- In contrast to `fork()`, `clone()` allows **fine grained control** over which aspects the thread shares
  - This is possible because the process control block is based on **several sub-blocks**, e.g. for the file descriptors

Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID
CLONE_PARENT	New thread has same parent as caller	New thread's parent is caller

Figure: Control flags for the clone call (Tanenbaum)

- Process scheduling has evolved over different versions of Linux to account for **multiple processors / cores**, processor **affinity**, and **load balancing** between cores
- Linux distinguishes between two types of tasks for scheduling:
  - **Real time tasks** (to be POSIX compliant), divided into:
    - Real time FIFO tasks
    - Real time Round Robin tasks
  - **Time sharing tasks** using a **preemptive multitasking** approach (**variable** in Windows)
- The most recent scheduling algorithm in Linux is the "**completely fair scheduler**" (CFS, before the 2.6 kernel, this was an $O(1)$ scheduler)

- **Real time FIFO** tasks have the **highest priority** and are scheduled using a FCFS approach, using **preemption if a higher priority** job shows up
- **Real time round robin tasks** are preemptable by **clock interrupts** and have a **time slice** associated with them
- Both approaches **cannot guarantee hard deadlines**

- The CFS **divides the CPU time** between all processes
- If all N processes have the **same priority**:
  - They will be allocated a "time slice" equal to $\frac{1}{N}$ times the available CPU time
  - I.e., if N equals 5, every process will receive 20% of the processor's time
- The length of the **time slice** and the "available CPU time" are based on the **targeted latency** ($\Rightarrow$ every process should **run at least once** during this interval)

- A **weighting scheme** is used to take different priorities into account
- If process have **different priorities**:
  - Every process i is allocated a **weight** w_i that reflects its priority
  - The “time slice” allocated to process i is then **proportional to** $\frac{w_i}{\sum_{j \in N} w_j}$

- The tasks with the **lowest amount** of "used **CPU time**" are **selected first**
- If N **is very large**, the **context switch time will be dominant**, hence a lower bound on the "time slice" is imposed by the minimum granularity
  - A process's time slice can be **no less** than the **minimum granularity** (response time will deteriorate)

Memory Management

Process's Memory Image

- Process/program contains **three segments**:
  - **Program text** containing machine instructions produced by the compiler and is usually **read only**
  - **Data segment** containing storage for the program's variables containing **initialised and uninitialised** data and can vary in size (using the system call `brk`)
  - The **stack segment**, usually starting at the top of the address space
- Text segments can be **shared** between multiple processes running the same program

Memory Management

Process's Memory Image

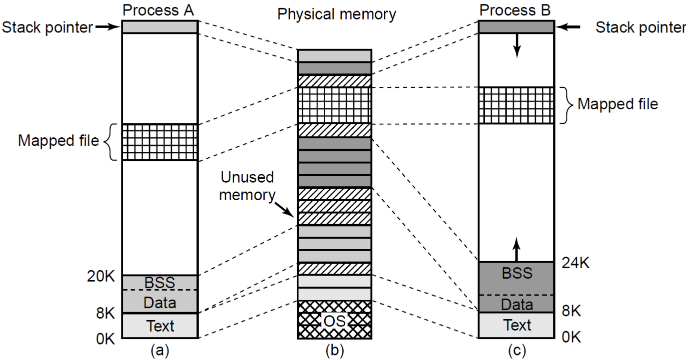

Figure: Page Replacement (Stallings)

Memory Management

Memory Zones in Linux

- Physical memory is divided into **three zones**:
  - ZONE_DMA for DMA access of old legacy devices
  - ZONE_NORMAL
  - ZONE_HIGHMEM
- A **memory map** is maintained by the kernel to administer the use of physical memory

Memory Management

The Kernel Memory Map

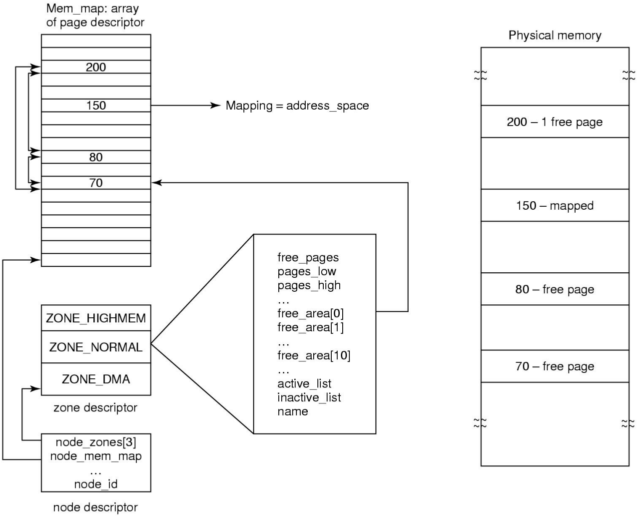

Figure: Memory Map in Linux

Memory Management

The Kernel Memory Map

- A **page descriptor** is maintained **for every page** frame containing a **link to the virtual address space** in which it is used, **link pointers**, and other fields
- A **zone descriptor** is maintained for every zone, containing information on the **memory utilisation**, page **replacement bounds**, an array (ordered by size) of linked lists of page descriptors for unused pages

Linux

Overview: Structure of a Linux System

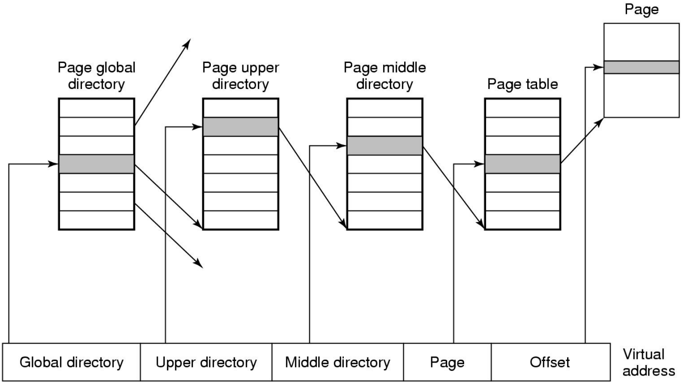

Figure: Linux's Four Level Page Table

Memory Management

The Page Allocator

- Physical memory is managed by the **page allocator** (one per zone) that allocates and frees physical pages using the **buddy algorithm**
- A request for memory is **rounded up** to the nearest power of 2, e.g. a request for 7 pages is rounded up to 8 ($\Rightarrow$ internal fragmentation)

Memory Management
The Page Allocator (Cont'ed)

- The buddy algorithm **divides memory recursively** until a **contiguous chunk** of the correct size is found (free lists in the zone descriptor are used to keep track of blocks)
- Free neighbouring blocks are **merged**

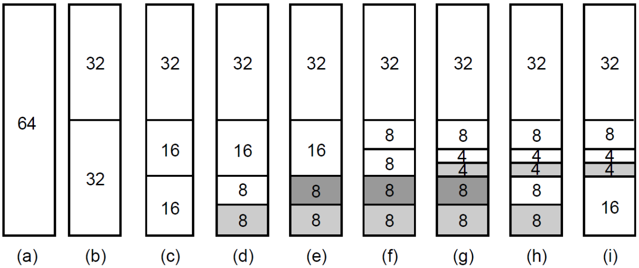

Figure: The Buddy Algorithm (Tanenbaum)

Memory Management
The Page Allocator

- Several **memory management subsystems** rely on the page allocator to manage their own memory:
  - `kmalloc()` for allocating **arbitrary sized memory chunks** in the kernel by splitting up individual pages
    - Separate data structure in the kernel are required to manage this
    - Memory is pinned (i.e., cannot be paged out) and must be freed explicitly
  - The **slab allocator** that carves smaller units out of one or multiple contiguous pages to store, e.g., to cache/store kernel data structures
  - The virtual memory manager

Memory Management

Swapping and Paging in Linux

- Linux uses a **pure demand paging** approach (i.e. no pre-paging or working sets)
- Pages reclaimed by the **paging daemon** can be:
  - Mapped to files on the disk, e.g. for **text segments** and **memory mapped files**
  - Mapped onto a **swap file** or **swap partition** (which is faster – it avoids overhead, is contiguous, and uses a next fit algorithm)
- "**Swapping**" is implemented on a **page granularity** (i.e. paging and virtual memory are used)

Memory Management
Page Replacement in Linux

- Linux's **page replacement algorithm** is a modified version of the **second chance clock algorithm**:
  - Each pass of the clock reduces the **age value**
  - Each reference to a page increases the age value

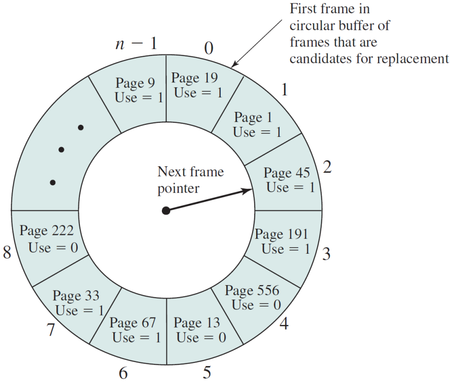

Figure: Memory Image (Tanenbaum)

Memory Management

Page Replacement in Linux

- Linux distinguishes between **unreclaimable**, **swappable** which must be re-written to the swap area, **syncable** which must be written to the disk, and **discardable pages** which can be reclaimed immediately
- Priority is given to **reclaiming "easy" pages**, e.g. discardable or not referenced pages
- The page daemon aims at keeping a good **set of free pages** available at all times

Summary
Take-Home Message

- Processes, task structures, process creation, process scheduling in Linux
- Memory management, buddy algorithm, slab allocator, page replacement in Linux