# Linux Virtual Memory System

## Swapping: Abschließende Betrachtung

Prof. Dr.-Ing. Andreas Heil



v1.0.1

# X86 Linux

Bisher: Verschiedene Aspekte der Virtualisierung kennen gelernt

Jetzt: Konkrete Betrachtung von Linux (für Intel x86)

# Linux Adressraum

- Besteht aus einem Adressbereich für den Kernel und für User-Anwendungen
- In 32-Bit Systemen ist die Trennung bei Adresse 0xC0000000, Adressen von 0x00000000 bis 0xBFFFFFFF sind User virtuelle Adressen, von 0xC0000000 bis 0xFFFFFFFF sind Kerne virtuelle Adressen, 64-Bit Systeme haben eine ähnliche Teilung (allerdings andere Adressen)

# Kernel Adressraum (1)

- Kernel Adressraum ist unter Linux aufgeteilt
- Logische Kernel Adressen (engl.kernel logical addresses)
  - Entspricht dem „normalen“ Adressraum des Kernels
  - PageTables, Per-Process Kernel Stacks etc. liegen hier
  - Kann nicht auf Platte ausgelagert werden
  - Besonderheit: Dieser Adressraum ist auf den ersten Teil des physikalischen Speichers gemappt
  - Speicher, der im logischen Adressraum fortlaufend ist, ist dies auch im physikalischen Speicher
  - Wird fürDirectMemory Access genutzt (I/O)

# Kernel Adressraum (2)

- Virtuelle Kerneladressen (engl.kernel virtual addresses)
  - Fortlaufende Pages im virtuellen Adressraum müssen nicht fortlaufend im physikalischen Speicher sein
  - Ist leichter zu allokieren, wird z.B. für Puffer genutzt

# Page Tables

- X86: hardware-basierte, multi-level Page Table
- Eine Page Table pro Prozess
- Betriebssystem kümmert sich um
  - Erzeugen, Löschen von Prozessen und derenContext-Switches
  - Sicherstellen, dass die MMU die korrekte Page Table verwendet
- S. 11, Kapitel 23 OSTEP, 64-Bit Page Table

# Large Page support

- Linux unterstützt nicht nur Standard 4KB Seiten
- 2MB und sogar 1GB Seiten werden unterstütz (HugePages)
- Größerer (=weniger) Pages liefert besseres TLB Verhalten
- Bei 4KB Seiten kommen Page Misses häufig vor
- Manche Anwendungen benötigen bis zu 10% ihrer Cycles aufgrund von Page Misses
- HugePages reduzieren diesen Aufwand
- Nachteil: Große Pages ziehen interne Fragmentierung nach sich

# Page Cache

- Drei Quellen für Pages
  - Memory-MappedFiles (Daten aus Dateien und Geräten)
  - Heap Pages und
  - Stack Pages für jeden Prozess
- Page Cache überwacht, ob Pages
  - Nur gelesen wurde („clean“) oder
  - Geändert wurden (d.h. „dirty“ sind)
  - DirtyPages werden regelmäßig zurückgeschrieben

# 2Q Replacement Algorithm

- Nutzt zwei Listen
  - InactiveList: Wenn auf eine Seite das erste Mal zugegriffen wurde
  - ActiveList: Wenn auf eine Seite aus derInactiveList zugegriffen wird, wandert sie in dieActiveList
- Muss eine Seit ersetzt werden, wird eine aus derInactiveList gelöscht
- Seiten von derActiveList werden regelmäßig zurück in dieInactiveList geschoben
- ActiveList entspricht ~2/3 des gesamten Caches
- Die Listen selbst werden mit einem Clock Algorithmus verwaltet

# Security Probleme

- Pufferüberlauf (engl.bufferoverflow)
  - Wird ausgenutzt, um beliebige Daten in ein System einzuschleusen
  - Hier könnten Daten größer 100 Byte an die Funktion übergeben werden
  - Puffer läuft über und überschreibt ggf. andere Daten im System

# Priviledged Escalation

- Eigentlich nicht schlimm, ggf. falsche Ausgaben oder das Betriebssystem stürzt ab
- ABER: Mit ausreichend Geduld lasen sich Angriffe entwickeln, die einen solchen Pufferüberlauf nutzen, um das System in einen privilegierten Modus zu versetzen (engl.priviledge escalation)
- Grundsätzlich ist das keine so gute Idee… (vgl. ersten Vorlesungseinheiten)
- Gegenmaßnahmen
- NX-Bit (No-eXecute): Besitzt eine Seite dieses Bit, kann daraus kein Code ausgeführt werden

# Return-Oriented Programming

- Return-OrientedProgramming(ROP)
- Code verlinkt an vielen Stellen auf die C-Bibliothek
- Nun wird Code derart gezielt eingeschleust und der Stack derart überschrieben, dass Sprungadressen nicht mehr in die C-Bibliothek weisen, sondern in Schadcode (Instruktion) und eine entsprechend Return-Instruktion.
- Diese sog. Gadgets können nun so hintereinander ausgeführt werden, dass quasi beliebiger Code ausgeführt werden kann
- Gegenmaßname
- Betriebssystem führt sog.AddressSpace LayoutRandomization(ASLR) ein, Anordnung von Stack, Heap und Code erfolgt immer wiederzufalligund die Adressen sind somit nicht mehr vorhersehbar

# Meltdown & Specter

- ASLT wird auch für den Kernel eingesetzt, dieser habt damit aber ein paar Probleme
  - Moderne CPUs führen eine sog.Speculative Execution aus, „raten“ also welche Instruktion als nächste ausgeführt werden könnte, und machen das auch schon mal
  - Falls richtig geraten, wird das Programm schneller, wenn nicht, wird es nicht langsamer
  - Speculativ Execution hinterlässt allerdings Spuren (Prozessor Caches, Verzweigungen, Sprungadressen etc.)
  - Hierdurch kann sogar auch Speicherbereiche zugegriffen werden, die eigentlich durch die MMU geschützt sind
  - Mehr dazu unter specterattack.com bzw. meltdownattack.com

# Referenzen

[1]Maziero, C., dos Santos, D. &Santin, A. Evaluationofdesktopoperatingsystemsunderthrashingconditions. JBrazComputSoc19, 29–42 (2013), https://doi.org/10.1007/s13173-012-0080-8