Humboldt University

Computer Science Department
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Operating Systems Principles

Lab 1 – Memory Management Unit

Fünftes Praktikum

- 2 Wochen Zeit zum Lösen der Aufgaben
  - Aufgabenstellung auf der SAR Website
  - Abgabe über GOYA

- In dieser Woche
  - Erläutern der Aufgabenstellung
- Folgende Woche
  - Zeit zur Bearbeitung
- Nächste Veranstaltung
  - Siehe Goya

Aufgabe

- Implementierung einer MMU im Userspace
- Mehr virtueller als physikalischer Speicher
- “Physikalischer Speicher” mittels malloc allokieren
- Zugriff auf virtuellen Speicher mittels 2 Funktionen
- Übersetzung von virt. in phys. Adressen
- Auslagern von nicht benötigten Seiten in ein Swap File

Systems Architecture Group
http://sar.informatik.hu-berlin.de

Aufgabe

- Implementierung einer MMU im Userspace

- Hinweis:
  - Speicher soll Byte-weise adressiert werden können
  - System ist little-endian

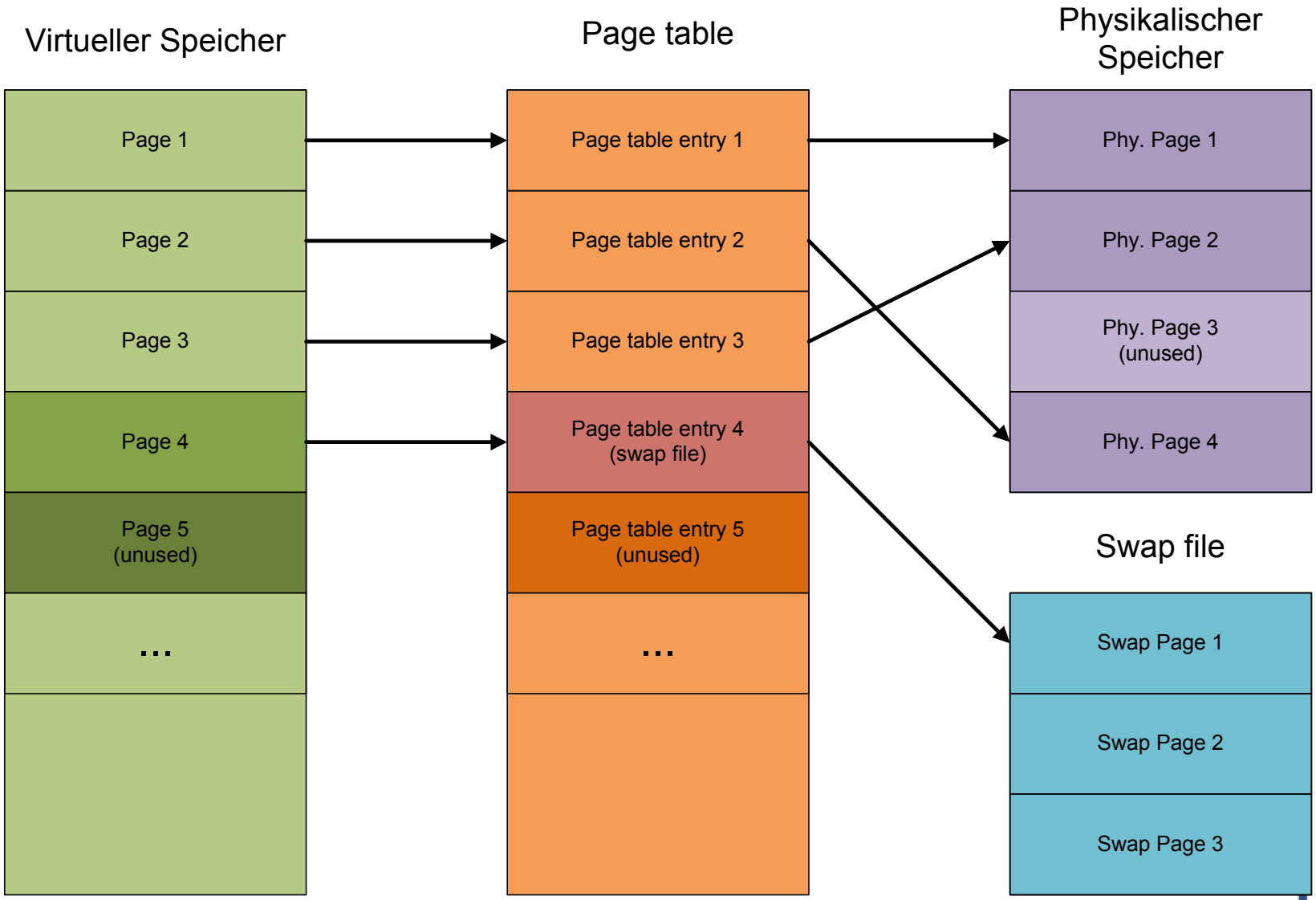
- Was nicht dazu gehört!!
  - Speicherschutz (Zugriffsrechte)
  - Keine hochkomplizierten Ersetzungsstrategien (Paging)

Lab 5 – Memory Management Unit

- MMU
  - Übersetzung von virtuellen in physikalische Adressen
  - Für jedes Laden & Speichern (load/store)
  - Software steuert Überstezung
    - MMU ist “programmierbar”
    - Benötigt privilegierte Anweisungen
  - CPU-Sicht
    - Virtuelle Adressen
  - Jeder Prozeß hat seinen eigenen virtuellen Adressraum ([0, high]
  - Memory chip or I/O device view
    - ^Physikalische Adressen

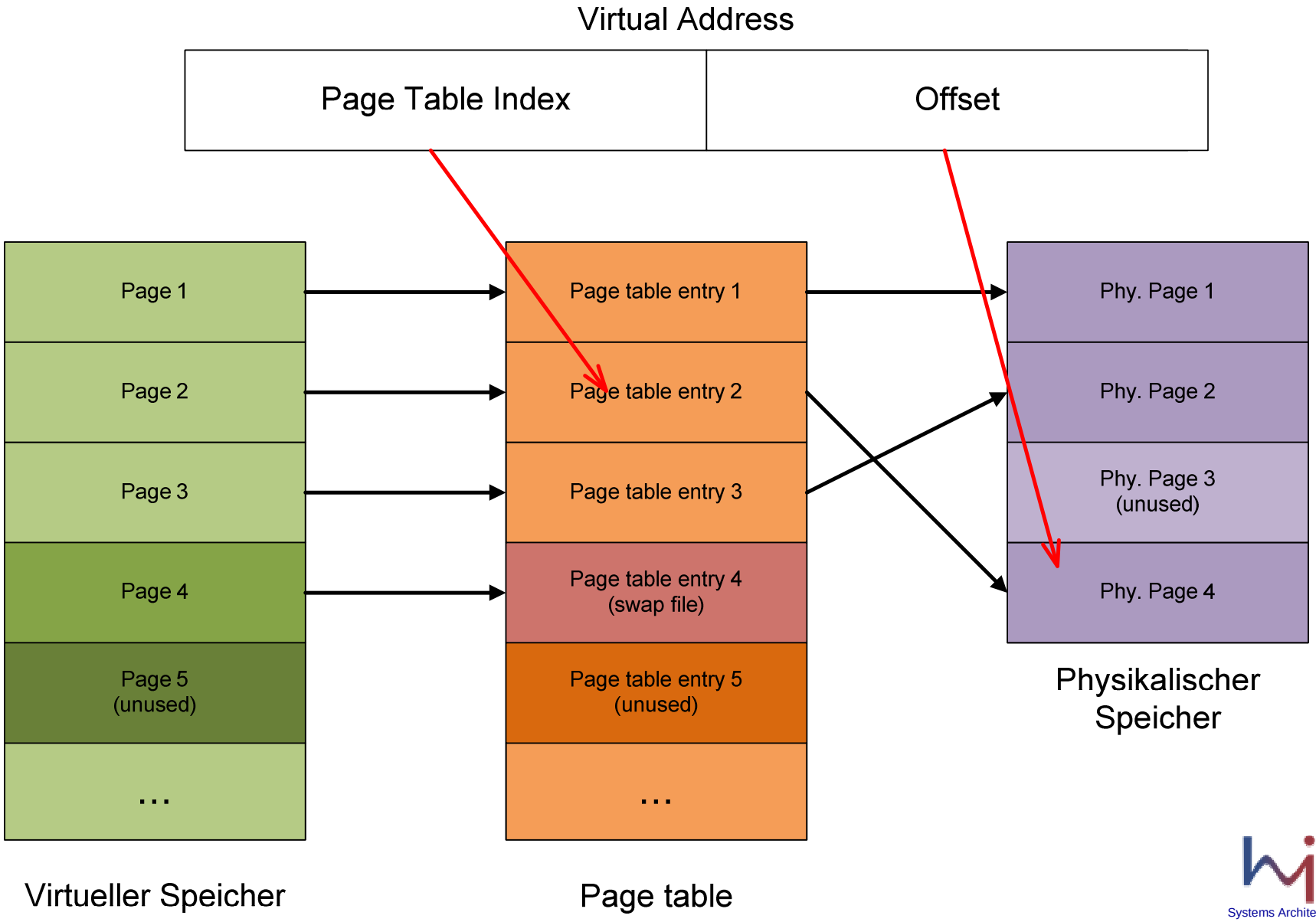
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Lab 5 - MMU



Systems Architecture Group
http://sar.informatik.hu-berlin.de

Lab 5 - MMU



Systems Architecture Group
http://sar.informatik.hu-berlin.de

Lab 5 – Memory Management Unit

- Files
  - page.(h|c): Funktionen und Strukturen für eine Speicherseite
  - pagetable.(h|c): Funktionen für die Pagetable
  - swap.(h|c): Zugriff auf Swap-File
  - mmu.(h|c): MMU
  - mmu_test.c

- Funktionen in mmu.h
  - mmu *init_mmu(int phy_mem_size, int virt_mem_size, int page_size, char* swapfile)
  - int32_t get_int32_t(mmu *mmunit, uint32_t address)
  - void store_int32_t(mmu *mmunit, uint32_t address, int32_t value)

Lab 5 – Memory Management Unit

- Struktur *mmu*
  - Enthält Informationen für MMU (Grösse phy/virt. Memory,…)
  - Parameter der Funktionen zum Schrieben & Lesen des Speichers

```
typedef struct mmu_info {
  int phy_mem_size;
  …
} mmu;
```

- Mögliche/Nötige Erweiterungen:
  - Pagetable
  - Name des Swapfiles
  - …

Systems Architecture Group
http://sar.informatik.hu-berlin.de

Lab 5 – Memory Management Unit

- mmu *init_mmu(int phy_mem_size, int virt_mem_size, int page_size, char* swapfile)
  - *Initialisierung der MMU*
  - „Allocation des physikalischen Speichers“
  - Anlegen der Pagetable
  - Erzeugen der Swap-Datei

- int32_t get_int32_t(mmu *mmunit, uint32_t address)
  - *Lesen eines int32_t aus dem Speicher*
  - *Übersetzen der virt. Adresse in eine phy. Adresse (Pagetable)*

- void store_int32_t(mmu *mmunit, uint32_t address, int32_t value)
  - *Speichern eines int32_t*

Lab 5 – Memory Management Unit

```
…
#include "mmu.h"

#define PAGE_SIZE       512
#define PHY_MEM_SIZE    1024 /* 1 M/KByte */
#define VIRT_MEM_SIZE 4*1024 /* 4 M/KByte */

int main( int argc, char **argv)
{
…
  _mmu = init_mmu(PHY_MEM_SIZE, VIRT_MEM_SIZE, PAGE_SIZE, "./swapfile.swp" );

  for ( i = 0; i < VIRT_MEM_SIZE; i = i + 4 ) {
    store_int32_t(_mmu, i, i);
  }
…
  for ( i = 0; i < VIRT_MEM_SIZE; i = i + 4 ) {
    if ( get_int32_t(_mmu, i) != i ) errors++;
  }

  printf("Errors: %d\n",errors);
…
}
```


Systems Architecture Group
http://sar.informatik.hu-berlin.de

Dateioperation

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ... );
```

- Öffnet eine Datei
- Argumente
  - path: Name(Pfad) der Datei
  - oflags: Flags zum Öffnen der Datei (Anhängen,…)
- Rückgabe
  - Filedeskriptor

Dateioperation

```
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbyte);
```

- Liest aus einer Datei in einen Buffer
- Argumente
  - filedes: Filedeskriptor der Datei
  - buf: Zeiger auf den Buffer in den die gelesenen Daten geschrieben werden
  - nbyte: Anzahl der zu lesenden Bytes
- Rückgabe
  - Anzahl der gelesenen Bytes

Dateioperation

```
#include <unistd.h>

ssize_t write(int fildes, void *buf, size_t nbyte);
```

- Schreibt aus einem Buffer in eine Datei
- Argumente
  - filedes: Filedeskriptor der Datei
  - buf: Zeiger auf den Buffer in den die Daten gelesenen werden
  - nbyte: Anzahl der zu schreibenen Bytes
- Rückgabe
  - Anzahl der geschriebener Bytes

Dateioperation

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

- Setzt den File Offset einer geöffneten Datei
- Nächste Dateioperation wird dort lesen oder schreiben
- Argumente
  - filedes: Filedeskriptor der Datei
  - offset: zu setzender Offset
  - whence: Option zum setzen des Offsets
    - SEEK_SET: File offset wird genauer an „offset“ gesetzt
    - SEEK_CUR: File offset ist aktueller Offset (File location) plus „offset“.
    - SEEK_END: neuer Offset ist „offset“ Bytes vom Ender der Datei
- Rückgabe
  - Offset

Dateioperation

```
#include <unistd.h>

int close(int fildes);
```

- Schließt eine Datei
- Argumente
  - filedes: Filedeskriptor der Datei
- Rückgabe
  - Status

Systems Architecture Group
http://sar.informatik.hu-berlin.de