Übungsblatt 5

Übung zur Betriebssystemarchitektur
SS 2005

Dipl.-Inf. Bernhard Rabe
Betriebssysteme & Middleware

Inhalt

- Kernel-Debugger
- Virtueller Speicher
- Speicher API
- Memory-Mapped Files
- Shared Memory
- IPC

- EPROCESS dt nt!_EPROCESS
- ETHREAD dt nt!_ETHREAD
- !process
- !vm
- !memusage
  - pagefile
- !drivers

Virtueller Speicher

- Vergrößerung des für ein Programm verfügbaren Speichers
- CPU übersetzt logische Speicheradressen in physikalische Speicheradressen
- Trennung der Adressräume von Prozessen
  - keine Kenntnis von physikalischen Adressen
- flacher fortlaufender Adressraum

Logische & Physikalische Adressen

- Abbildung durch Betriebssystem & CPU
  - Segmentierung
    - unterschiedlich große Segmente
    - Segmenttabelle; Basis- und Limitadresse des Segments
    - logische Adresse (Segment,Offset)
    - Auslagerung von Segmenten

- Paging
  - Aufteilung des physikalischen Speichers in Frames gleicher Größe
  - Aufteilung des logischen Speichers in Pages (gleiche Größe wie Frames)
  - Seitentabelle (Page-Table): (Index in der Pagetable, Offset im Frame)
  - Auslagerung von (kleinen) Seiten

Virtueller Speicher API

- VirtualX
- malloc libc
- HeapX
- File-Mapping

Speicherbereiche

- Stack
  - Funktionslokaler automatischer Speicher
- Heap
  - dynamisch wachsender Speicher
- virtueller Adressraum
  - fortlaufende Adressen
- Working Set
  - Seiten die ohne Page-Faults verfügbar sind

VirtualAlloc

- **LPVOID VirtualAlloc(LPVOID** *lpAddress*, **SIZE_T** *dwSize*, **DWORD** *flAllocationType*, **DWORD** *flProtect* **);**
  - Reservered vs. Commited Speicherseiten aus dem virtuellen Adressraum des rufenden Prozesses
  - 2-Phasen Ansatz zu Speicherreservierung
  - *flAllocationType*
    - MEM_RESERVE reserviert nur Speicheradressen, kein Speicher im Pagefile → Zugriff liefert Speicherzugriffsfehler
    - MEM_COMMIT Speicher im Pagefile steht zur Verfügung

VirtualAlloc (II)

- Reserved Speicher
  - Windows legt Virtual Address Descriptors (VAD) im Page-Table an
  - kein Backing Store (Platz im Pagefile)
- Commited Speicher
  - gültiger Page-Table Eintrag
  - Backing Store gesichert

VirtualFree

- **BOOL VirtualFree(LPVOID** *lpAddress*, **SIZE_T** *dwSize*, **DWORD** *dwFreeType* **);**
- lpAddress Startadresse der freizugebenden Seiten (Bereich) (von VirtualAlloc)
- *dwFreeType*
  - MEM_DECOMMIT wieder im den Reserved Modus
  - MEM_RELEASE Seiten wieder freigeben
    - keine Größenangabe

Heaps

- dynamisch wachsender Speicher
- 1 Heap pro Prozess
  - Windows Prozesse können weitere Heaps anlegen

Libc-API

- Benutzen den Default-Heap des Prozesses
- void *malloc(size_t size)
  - allokiert Speicher auf dem Prozess-Heap
- void *realloc(void *memblock, size_t size)
  - vergrößert den Speicherblock memblock auf die neue Größe size (Speicher kann verschoben werden)
- free(void *memblock)
  - gibt Speicher wieder frei

vm.exe

Heap-API

- **HANDLE GetProcessHeap(void);**
  - Handle auf den Default-Heap des Prozesses
- **HANDLE HeapCreate(DWORD** *flOptions*, **SIZE_T** *dwInitialSize*, **SIZE_T** *dwMaximumSize* **);**
  - erzeugt neuen Heap

Heap-API

- **LPVOID HeapAlloc(HANDLE** *hHeap*, **DWORD** *dwFlags*, **SIZE_T** *dwBytes* **);**
- allokiert Speicher auf dem angegebenen Heap
- malloc-Verhalten bei Benutzung des Default-Heaps
  - HEAP_NO_SERIALIZE
  - HEAP_GENERATE_EXCEPTIONS

Heap-API

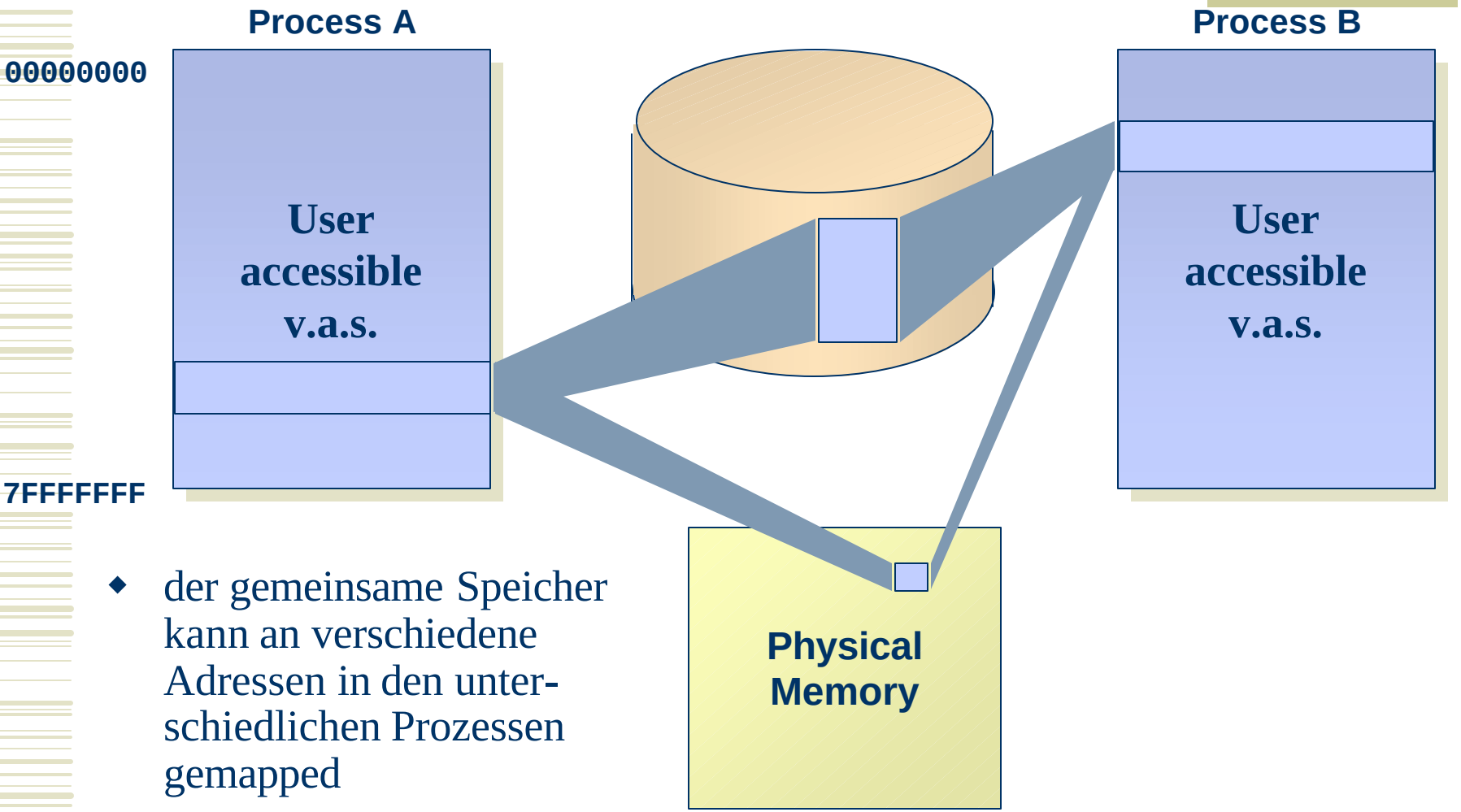
- **LPVOID HeapReAlloc(HANDLE** *hHeap*, **DWORD** *dwFlags*, **LPVOID** *lpMem*, **SIZE_T** *dwBytes* **);**

- **BOOL HeapFree(HANDLE** *hHeap*, **DWORD** *dwFlags*, **LPVOID** *lpMem* **);**
- **BOOL HeapDestroy(HANDLE** *hHeap* **);**
  - kein CloseHandle!

Memory-Mapped-Files

- Speicher der in mehreren Prozessen sichtbar ist
- gemeinsam genutzte System-DLLs
- Shared Memory über Pagefile

Shared Memory = File Mapped by Multiple Processes



- der gemeinsame Speicher kann an verschiedene Adressen in den unterschiedlichen Prozessen gemapped

Memory Mapped Files

- **HANDLE CreateFileMapping(**
  HANDLE *hFile*,
  LPSECURITY_ATTRIBUTES *lpAttributes*,
  DWORD *flProtect*,
  DWORD *dwMaximumSizeHigh*,
  DWORD *dwMaximumSizeLow*,
  LPCTSTR *lpName* **);**
- benötigt geöffnete Datei
- `hFile=0xFFFFFFFF` PageFile

- *flProtect*
  - *PAGE_READWRITE*
  - *PAGE_READONLY*
  - *PAGE_WRITECOPY*
- *dwMaximumSizeHigh,dwMaximumSizeLow*
  - Größe des verfügbaren Speicherbereiches
  - wenn 0, dann aktuelle Größe von hFile
- *lpName* Name des Mapping Objektes

OpenFileMapping

- **HANDLE OpenFileMapping(**
  DWORD *dwDesiredAccess*,
  BOOL *bInheritHandle*,
  LPCTSTR *lpName* **);**
- öffnet ein benanntes FileMapping Objekt

MapViewOfFile

- ```
  LPVOID MapViewOfFile(
      HANDLE hFileMappingObject,
      DWORD dwDesiredAccess,
      DWORD dwFileOffsetHigh,
      DWORD dwFileOffsetLow,
      SIZE_T dwNumberOfBytesToMap );
  ```
- blendet einen Bereich eines FileMapping-Objektes in den aktuellen Prozess ein
- Startposition und Länge
  ```

UnMapViewOfFile

- **`BOOL UnmapViewOfFile( LPCVOID`** *`lpBaseAddress`* **`);`**
- blendet den Speicherbereich mit der Adresse lpBaseAddress aus
- die gemappte Datei *`hFile`* bleibt solange geöffnet bis alle Views geschlossen sind

Verteilte Fraktalberechnung

- 500x500 Pixel großes Fraktal Bild als BMP-Datei erzeugen
- Server-Prozess erzeugt Mapping-File (BMP-Datei) und schreibt den BMP-Header
- Server startet Worker-Prozesse, die ein Teil des Fraktals berechnen und die Ergebnisse an die entsprechende Position in das Mapping-File schreiben

bmp_master.c

Aufgabenblatt 4

POSIX-Shared Memory

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int
  shmflg);
```

- `key` Bezeichner, Name
  - `key_t ftok(char *pathname, char proj);` erzeugt key_t aus Datei und proj
- `size` Größe des Speicherbereiches
- shmflg
  - IPC_CREAT erzeugt neues Speichersegment
- Rückgabe Identifier für das Speichersegment

Shared Memory

- `char *shmat(int shmid,char* shmaddr,int shmflg)`
  - blendet Speicherbereich mit der ID *shmid* ein
  - gewünschte Speicheadresse, 0 automatisch
- `int shmdt(char *shmaddr)`
  - blendet den Speicherbereich an shmaddr aus dem Prozessspeicherraum aus
- `int shmctl(int shmid,int cmd, struct shmid_ds *buf)`
  - setzt oder liefert Informationen über das Speichersegment shmid

shm.c

POSIX Memory Mappend Files

- `#include <sys/mmap.h>`
- `void* mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);`
  - blendet `length` Bytes beginnend mit `offset` aus `fd` in dem Prozessspeicher ein
  - `start` gewünschte Speicheradresse, sollte 0 sein
- `int unmap(void * start, size_t length);`
  - blendet Speicher an start wieder aus

Posix

Aufgabe 5.4

- Textdatei Invertierer mit VirtualAlloc
- Eingabedatei (1.Parameter) vollständig einlesen und virtuellen Speicher in Stücken von 16 Kb commiten
- Alle Zeilen vom Ende zum Anfang in die Ausgabedatei schreiben