Übungsblatt 4

Übung zur Betriebssystemarchitektur

WS 2004/05

Dipl.-Inf. Bernhard Rabe

Betriebssystem & Middleware

Interprozesskommunikation

- Win32
  - Anonymous Pipe
  - Named Pipe
  - Mailslots
  - MemoryMapped Files (Shared Memory)
- Linux
  - Anonymous Pipe’s
  - Fifos (Named Pipes)
  - Shared Memory
  - mmap

Interprozesskommunikation

- IPC
  - Informationsaustausch zwischen Betriebssystemprozessen
  - 1:1, 1:N Kommunikation
  - teilweise Rechnerübergreifend

Mailslots

- N:1 Kommunikation
- Netzwerkfähig (Windows Domänen)
- `\\.\mailslot\[Pfad]name`
  - Mailslot of dem lokalen Rechner
  - z.B. `\\.\mailslot\slot1`
- `\\Rechnername\mailslot\[Pfad]name`
  - Mailslot auf einem bestimmten Rechner
- `\\Domain\\mailslot\[Pfad]name`
  - alle Mailslots unter Pfad in der Domäne
- `\\*\mailslot\[Pfad]name`
  - alle Mailslots unter Pfad in der primäre Domäne

CreateMailSlot

- **HANDLE CreateMailslot(**
  LPCTSTR *lpName*,
  DWORD *nMaxMessageSize*,
  DWORD *lReadTimeout*,
  LPSECURITY_ATTRIBUTES *lpSecurityAttributes* **);**
- Erzeugung nur Lokal möglich
- *nMaxMessageSize*, 0=jede Größe
- *lReadTimeout*
  - 0=Leseoperation kehrt zuürck, wenn keine Nachricht vorhanden
    - MAILSLOT_WAIT_FOREVER Leseoperation wartet bis eine Nachricht eintrifft

MailSlot Information

- **BOOL GetMailslotInfo(**
  HANDLE *hMailslot*,
  LPDWORD *lpMaxMessageSize*,
  LPDWORD *lpNextSize*,
  LPDWORD *lpMessageCount*,
  LPDWORD *lpReadTimeout* **);**
- **BOOL SetMailslotInfo(**
  HANDLE *hMailslot*,
  DWORD *lReadTimeout* **);**

MailSlot Öffnen

- **HANDLE CreateFile(**
  LPCTSTR *lpFileName*,
  DWORD *dwDesiredAccess*,
  DWORD *dwShareMode*,
  LPSECURITY_ATTRIBUTES *lpSecurityAttributes*,
  DWORD *dwCreationDisposition*,
  DWORD *dwFlagsAndAttributes*,
  HANDLE *hTemplateFile* **);**

Mailslot öffnen

- *lpFileName* Mailslot-Adresse \\....
- *dwDesiredAccess* GENERIC_WRITE
- *dwShareMode* FILE_SHARE_READ
- *dwCreationDisposition* OPEN_EXISTING
- *dwFlagsAndAttributes*
  FILE_ATTRIBUTE_NORMAL
- Domänen-Wildcard *\... begrenzt Nachrichtengröße auf 424 Byte

Mailslot Schreiben (Klient)

- **BOOL WriteFile(**
  HANDLE *hFile*,
  LPCVOID *lpBuffer*,
  DWORD *nNumberOfBytesToWrite*,
  LPDWORD *lpNumberOfBytesWritten*,
  LPOVERLAPPED *lpOverlapped* **);**

```
char buf[100];
if(!WriteFile(hMailslot,buf,strlen(buf),&w
  ritten,NULL)) ErrorExit();
```

MailSlot Lesen (Server)

- **BOOL ReadFile(**
  HANDLE *hFile*,
  LPVOID *lpBuffer*,
  DWORD *nNumberOfBytesToRead*,
  LPDWORD *lpNumberOfBytesRead*,
  LPOVERLAPPED *lpOverlapped* **);**
- analog zu WriteFile

Anonymous Pipes

- **BOOL CreatePipe(**
  PHANDLE *hReadPipe*,
  PHANDLE *hWritePipe*, **LPSECURITY_ATTRIBUTES** *lpPipeAttributes*,
  DWORD *nSize* **);**
- Lese- bzw. Schreibeende
- StartupInfo bei CreateProcess()
- ReadFile/WriteFile

Named Pipes

- **HANDLE CreateNamedPipe(**
  LPCTSTR *lpName*,
  DWORD *dwOpenMode*,
  DWORD *dwPipeMode*,
  DWORD *nMaxInstances*,
  DWORD *nOutBufferSize*,
  DWORD *nInBufferSize*,
  DWORD *nDefaultTimeOut*, **LPSECURITY_ATTRIBUTES** *lpSecurityAttributes* **);**
- `\\.\pipe\Name` Erzeugung nur Lokal
- 1:1 Kommunikation
- mehrere Instanzen mit dem gleichen Namen möglich

- *dwOpenMode*
  - PIPE_ACCESS_DUPLEX Bi-direktional
  - PIPE_ACCESS_INBOUND Klient→Server
  - PIPE_ACCESS_OUTBOUND Server→Klient
- *dwPipeMode*
  - *PIPE_TYPE_BYTE*
  - *PIPE_TYPE_MESSAGE*
  - *PIPE_WAIT*

Dis/Connect Pipe (Server)

- **BOOL ConnectNamedPipe(**
  HANDLE *hNamedPipe*,
  LPOVERLAPPED *lpOverlapped* **);**
  - Server Prozess wartet auf eine Klientenverbindung zur Pipe
- **BOOL DisconnectNamedPipe(HANDLE** *hNamedPipe* **);**
  - Server Prozess beendet die Pipeinstanz

Dis/Connect Pipe (Klient)

- **`BOOL WaitNamedPipe( LPCTSTR` *`lpNamedPipeName`*`, DWORD` *`nTimeOut`* **`);`**
  - wartet bis eine Verbindung zur Pipe möglich ist
  - nTimeOut
    - `NMPWAIT_USE_DEFAULT_WAIT`
    - `NMPWAIT_WAIT_FOREVER`
    - oder Millisekunden
- **`CreateFile()`**
  - *`lpFileName`* **`\\[Rechnername]\pipe\name`**
  - *`dwDesiredAccess`* **`GENERIC_[READ|WRITE]`**
  - *`dwCreationDisposition`* **`OPEN_EXISTING`**
- **`CloseHandle()`** Schließt Pipe

Named Pipe Informationen

- **`BOOL GetNamedPipeHandleState( )`**
  - aktuelle Instanzen, Timeouts...
- **`BOOL SetNamedPipeHandleState( )`**
  - Modus _BYTE vs. _MESSAGE, _NOWAIT vs _WAIT
- **`BOOL GetNamedPipeInfo( )`**
  - Klient- oder Serverende, Buffergrößen

Asynchrones Lesen/Schreiben

- `Create<Operation>`
  - `FILE_FLAG_OVERLAPPED`
- *`lpOverlapped`* bei `Read/WriteFile`
  `ConnectNamedPipe`
  - OVERLAPPED Struktur

OVERLAPPED

```
typedef struct _OVERLAPPED {
  ULONG_PTR Internal;
  ULONG_PTR InternalHigh;
  DWORD Offset;
  DWORD OffsetHigh;
  HANDLE hEvent; } OVERLAPPED;
```

- Event wird bei fertiggestellter Operation signalisiert
- beginne Operation an Dateiposition `Offset`

GetOvelappedResult

- **BOOL GetOverlappedResult(**
  HANDLE *hFile*,
  LPOVERLAPPED *lpOverlapped*,
  LPDWORD *lpNumberOfBytesTransferred*,
  BOOL *bWait* **);**
- Handle dass bei der Overlapped Operation angegeben wurde
- lpNumber.. Anzahl der transferierten Bytes
- bWait auf die Beendigung der Overlapped Operation warten

```
char buf[100];
int len=100;
DWORD rd;
OVERLAPPED ovl={0,0,0,0};
ovl.hEvent=CreateEvent(0,FALSE,FALSE,0);
ReadFile(hfile,buf,len,NULL,&ovl);
```

- Warte auf Fertigstellung

```
WaitForSingleObject(ovl.hEvent,INFINITE);
GetOverlappedResult(ovl.hEvent,&ovl,&rd,TRUE);
printf("%d read: %s\n",rd,buf);
```

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

(Anonyme) Unix Pipes

- #include <unistd.h>
- `int pipe(int filedes[2])`
- Kommunikation
  - `filedes[1]` ⟶ `filedes[0]`
- `int dup(int oldfd)`
  - erzeugt eine Kopie von oldfd und nutzt den kleinsten verfügbaren (geschlossenen) Filedeskriptor
  - Ziel den muss kleinsten Filedeskriptor haben!

- **`int dup2(int oldfd, int newfd)`**
  - erzeugt eine Kopie von `oldfd` in `newfd`
  - `newfd` wird bei Bedarf geschlossen

```
int hpipe[2]; pipe(&hpipe[0]);
if(0==fork())
{
   dup2(hpipe[1],1); /*close(1);dup(hpipe[1]);*/
   close(hpipe[0]); close(hpipe[1]);
   execl("/bin/ls", "ls",0); ....
}
dup2(hpipe[0],0); /*close(0); dup(hpipe[1]);*/
close(hpipe[0]); close(hpipe[1));
execl("/bin/more", "more",0); ...
```

Named Pipes

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname,mode_t mode);
```

- erzeugt eine FIFO Spezialdatei im Filesystem
- Systemprogramm: mkfifo
- kann mit libc Dateifunktionen wie eine Datei behandelt werden

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
  - 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

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

Aufgabe 4.1

- Windows ACL
  - vererbbare Rechte
  - Rechte für mehrere Gruppen
- und Unix Gruppen
  - rwx-Bits für Besitzter,Gruppe und Andere
  - Benutzer können in mehreren Gruppen sein
  - Rechte sind additiv

Aufgabe 4.2

- Mailslots
- lesen.c
  - Liest aus einem angegebenen Mailslot auf dem lokalen Rechner
- schreiben.c
  - erzeugt

Aufgabe 4.3

- Fraktal mit mehreren Prozessen erzeugen
- Ausgabe der Ergebnisse in ein gemeinsam genutzen Speicher (Shared Memory)
- Erzeugung eines DIB (BMP) Bildes
  - Header Informationen auf der Übungsseite
  - Farbwerte pro Pixel: BGR
  - Bild wird von unten nach oben gespeichert

bmp_mandelbrot.