Übungsblatt 4

Übung zur Betriebssystemarchitektur
SS 2005

Dipl.-Inf. Bernhard Rabe
Betriebssysteme & Middleware

Aufgabe 4.2

- Implementierung von Scheduling-Algorithmen
- a) kooperativer Round Robin Scheduler
- b) präemptiver Round Robin Scheduler

Kooperatives Scheduling

- Ein Thread muss freiwillig die CPU an einen anderen Thread abgeben

Präemtives Scheduling

- Thread wird in seiner Ausführung unterbrochen und zu einem späteren Zeitpunkt an der gleichen Position fortgesetzt

Thread-Kontrolle

- DWORD ResumeThread(HANDLE *hThread*)
  - dekrementiert den Suspend-Zähler des Threads
  - setzt die Ausführung fort, wenn der Zähler 0 ist
- DWORD SuspendThread(HANDLE *hThread*)
  - setzt die Ausführung des Threads aus und erhöht den Suspend-Zähler

Usermode-Scheduler

- Kontrolle und Steuerung von Win32 Threads
- Kontrolle durch den Windows Schedulers besteht weiterhin -> Fibers
- API
  - UMS_ThreadNew
  - UMS_ThreadExit
  - UMS_ThreadYield
  - UMS_ThreadSchedule
  - UMS_ThreadScheduleP

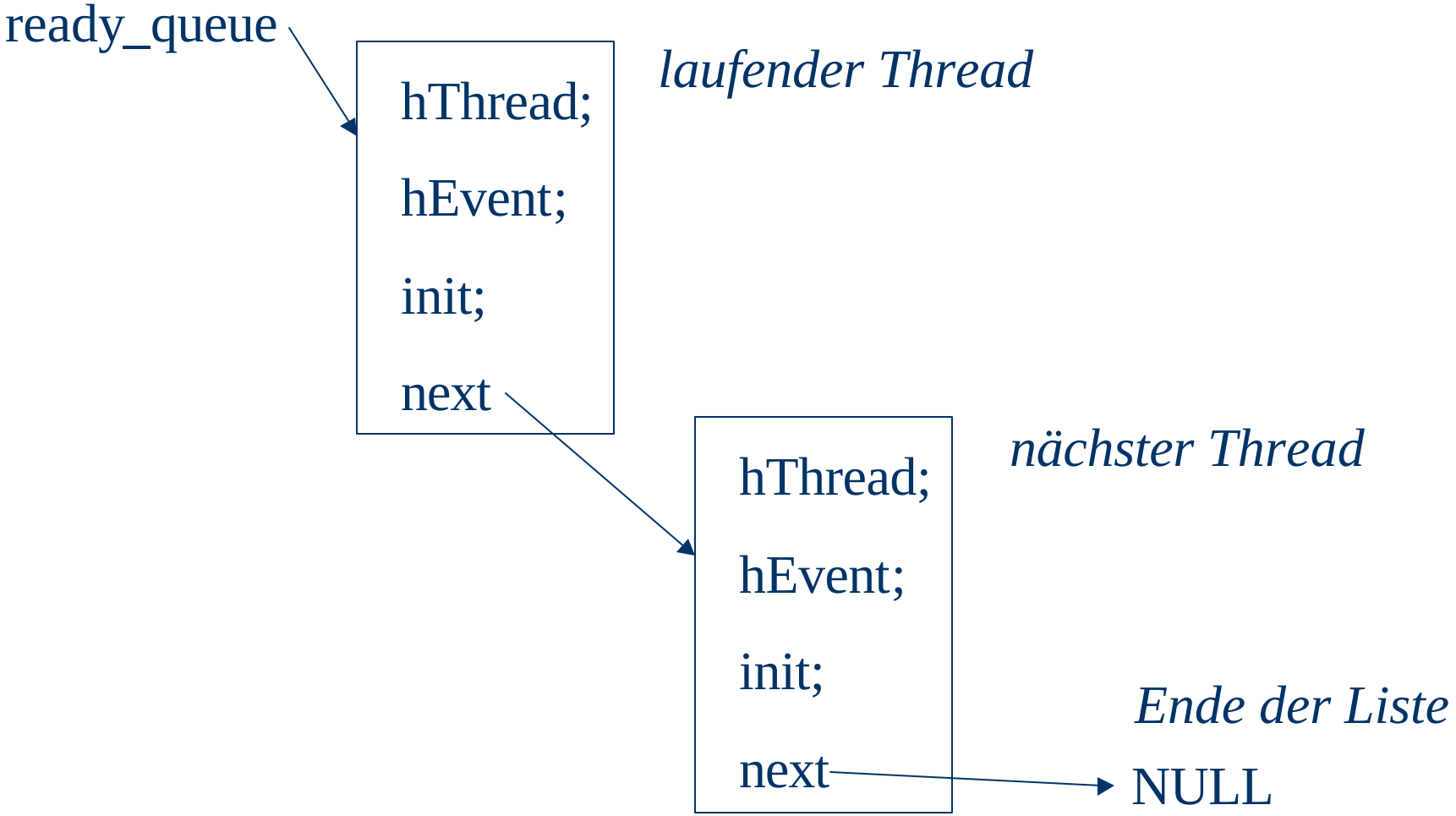
UMS_ThreadNew

- Erzeugung eines Win32 Threads
- Threads werden Suspendiert erzeugt
  - dwCreationFlags=CREATE_SUSPENDED
  - keine Ausführung nach dem Start
  - Start der Ausführung mit ResumeThread
- Verwaltung der Threads in einer verketteten Liste

ready_queue

- verkette Liste von thread_entry Einträgen
-
```
typedef struct _thread_entry{
    HANDLE hThread;
    HANDLE hEvent;
    DWORD dID;
    unsigned char init;
    struct _thread_entry *next;
} thread_entry;
thread_entry *ready_queue;
```

Ready-Queue


ready_queue
hThread;
hEvent;
init;
next
laufender Thread
hThread;
hEvent;
init;
next
nächster Thread
Ende der Liste
NULL

ThreadSchedule()

- startet den Thread am Anfang der ready_queue mit ResumeThread

```
ready_queue->init=0;
ResumeThread(ready_queue->hThread);
```

- rufender Thread wartet auf die Signalisierung seines Event-Objekts (hMain)

```
WaitForSingleObject(hMain,INFINITE);
```

ThreadExit()

- entfernt den rufenden Thread aus der Ready-Queue und beendet ihn
- weckt nächsten Thread (oder Main-Thread) mit *ResumeThread* bzw. *SetEvent* auf
- Handles freigeben
- Speicher freigeben

Aktivierung eines Threads in ThreadYield()

- 1. laufender Thread (durch ThreadSchedule gestartet) ruft ThreadYield auf
- aktualisiert die Einträge der ready_queue
- weckt den nächsten Thread auf
  - ResumeThread, wenn init=1
  - SetEvent, wenn init = 0
- wartet auf die Signalisierung seines Event-Objekts hEvent

ResumeThread/SuspendThread

- ThreadYield Lösung ?

```
ResumeThread(ready_queue->hThread);
SuspendThread(GetCurrentThread());
```

sample1.c

- First In First Out (FIFO)
  - Batch-Processing
- keine Unterbrechung
- Wechsel bei der Beendigung eines Thread mit UMS_ThreadExit()

sample1.c

sample2.c

- kooperatives Round-Robin Scheduling
- laufender Thread kann mit ThreadYield() die CPU aufgeben (Windows-Scheduler nicht betrachtet)
- ThreadExit beendet einen laufenden Thread

sample2.c

sample3.c

- präemptiver Round-Robin-Scheduler
- Thread hat keine Kontrolle über die Vergabe der CPU (Windows-Scheduler nicht betrachet)
- Scheduler: UMS_ThreadScheduleP(quantum)
- mit Resume/SuspendThread lösbar

sample3.c