TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Implementierung eines hardwaregestützten Schedulers für den Bytecode-Prozessor SHAP

Vortrag zum Beleg

Marco Kaufmann
marco.kaufmann@inf.tu-dresden.de

Dresden, 26.08.2008

Inhalt

Schedulingrelevante Aufgaben

- Threadverwaltung, Threadscheduling.
- Verwaltung von Monitoren.
- Blockierende Ein - / Ausgabe.
- Interrupts.

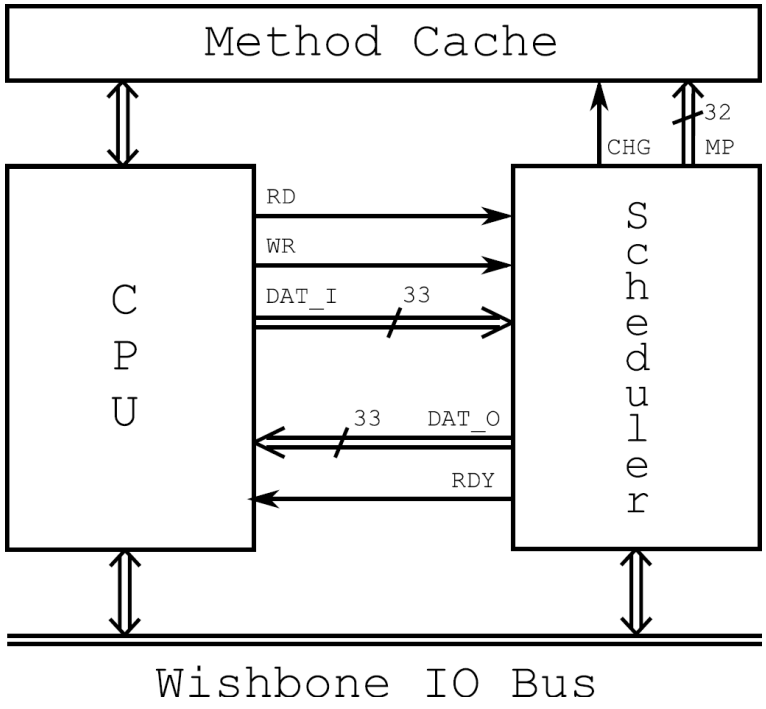
Derzeit in SHAP durch Mikrocode realisiert.

➔ Belastung der CPU, Anwendungsprogrammen steht somit weniger Rechenzeit zur Verfügung.
➔ Ferner derzeit kein prioritätenbasiertes Scheduling unterstützt.

Aufgabenstellung

- Auslagerung eines Großteils der schedulingrelevanten Aufgaben in eine dedizierte Hardwarekomponente.
- Literaturstudium zu gebräuchlichen Schedulingansätzen für JVM – Implementierungen im Embedded- Bereich.
- Berücksichtigung von Monitorverwaltung und blockierender Ein - / Ausgabe.
- Berücksichtigung einer möglichen Multi-Core-Erweiterung.
- Auswahl, Entwurf und Implementierung eines Ansatzes.
- Zusammenfassung und Dokumentation des Entwurfs und der erzielten Ergebnisse.

Schnittstellen des Schedulermoduls



- Schnittstelle zum Wishbone IO Bus.
- Direkte Schnittstelle zur CPU, Mikrocodes **`stsu`**, **`ldsu`**.
- Schnittstelle zum Method Cache.
- Keine Schnittstelle zum Memory Manager!

Threadscheduling in eingebetteten Systemen

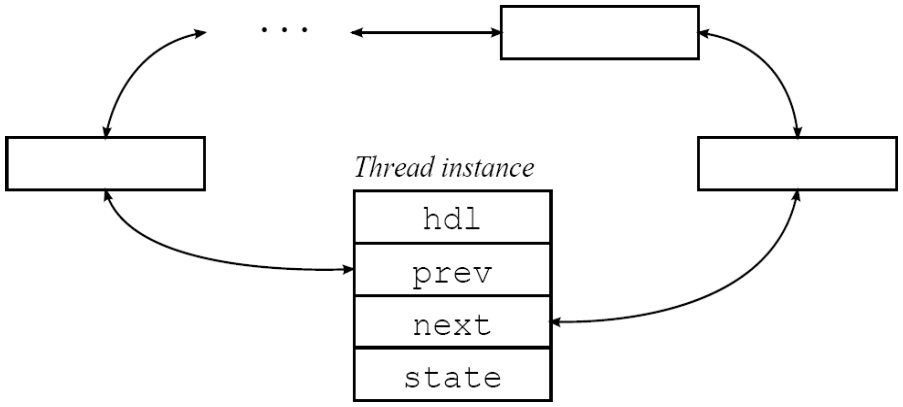
➔ Echtzeitfähigkeit !!!

- Vorhersagbarkeit und Einhaltung von Deadlines wichtiger als Performance, hohe Auslastung oder Durchsatz.
- verbreitet: EDF, RMS.
- falls Parameter a priori bekannt: RMS, Round Robin.
- jeder Schedulingalgorithmus geeignet, der einem Job eine maximale Laufzeit garantieren kann (z.B. GP).

➔ Gilt auch für Implementierungen der JVM im eingebetteten Bereich.

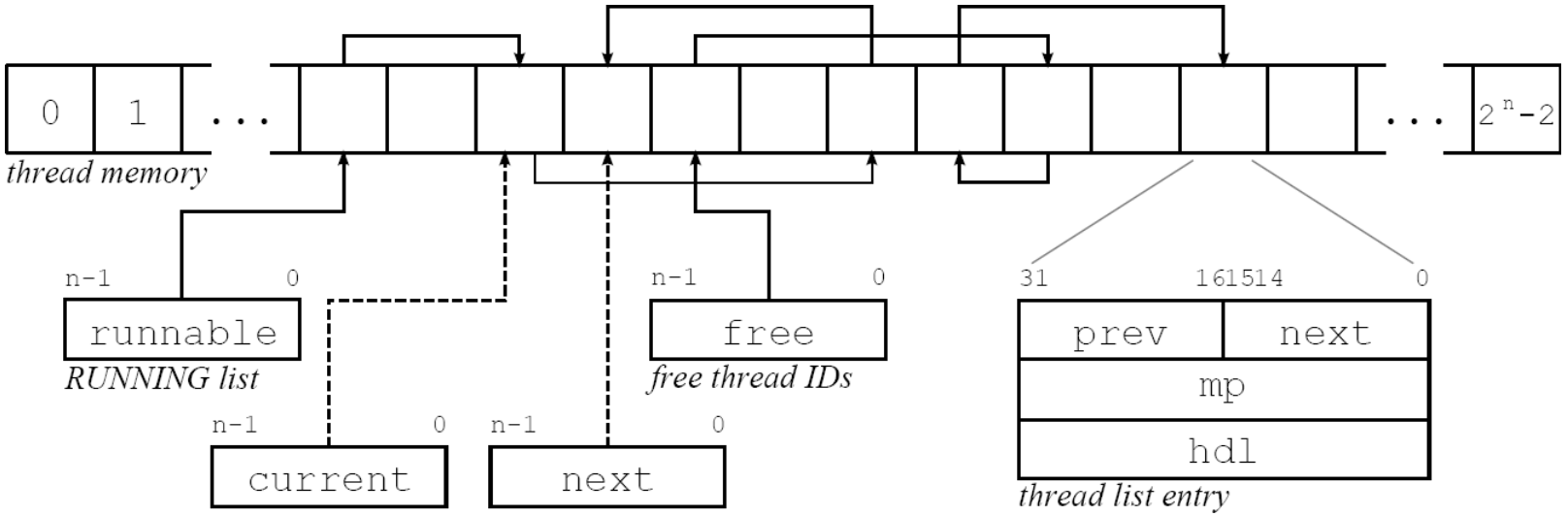
- Z.B. Komodo: FPP, EDF, LLF, GP.

Derzeitige Implementierung in SHAP



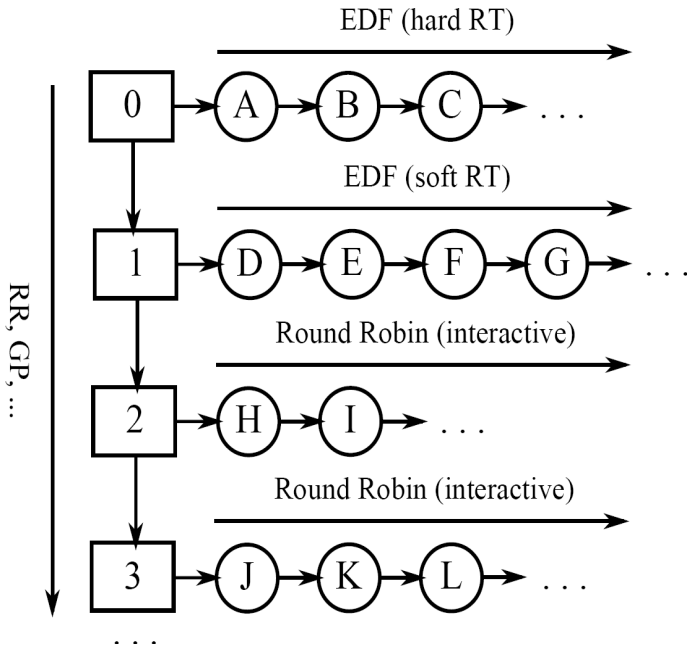
- schedulingrelevante Daten im Object Heap.
- *Runnable* als ringförmige Verkettung von Thread Objekten.
- keine Threadprioritäten.
- Round Robin Scheduling.

Implementierung im Schedulermodul (I)



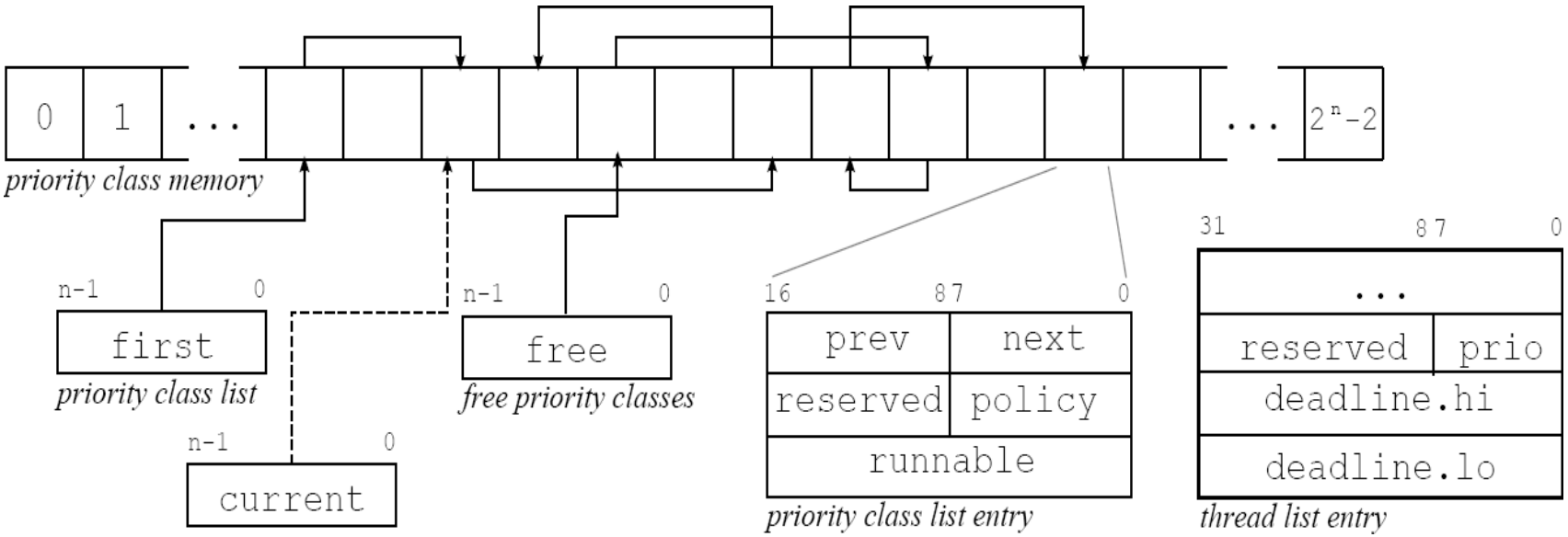
- schedulingrelevante Daten als Datensätze im Scheduler.
- Thread IDs anstatt Thread Objekte.

Threadscheduling im Schedulermodul

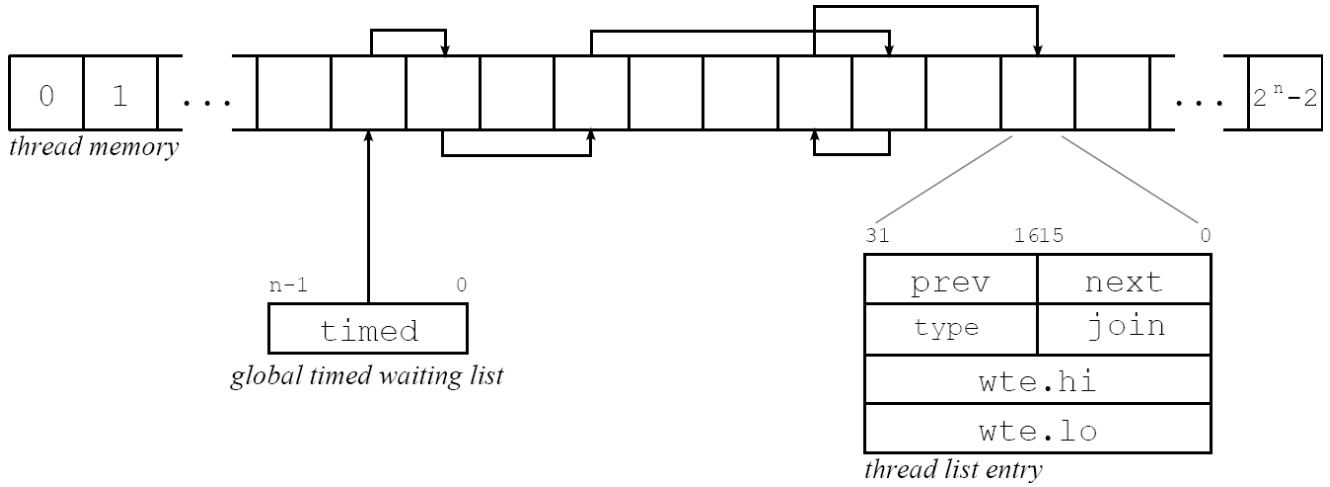


- *Runnable* unterteilt in Prioritätenklassen.
- Zweidimensionales Scheduling: Anwendung unterschiedlicher Schedulingstrategien auf Thread – und Prioritätenklassenebene.
- Threadebene: RR, EDF.
- Klassenebene: RR, GP, FPP.
- Genügt sowohl Echtzeitanforderungen als auch Anforderungen interaktiver Threads.

Implementierung im Schedulermodul (II)



Implementierung im Schedulermodul (III)



- globale, zeitlich sortierte Warteschlange für Timed Waiting.
  - ➔ *Thread.sleep*, *Thread.join* und *Object.wait.*
- eigene Join – Warteschlange für jeden Thread.
  - ➔ zusätzlicher Eintrag in globaler Warteschlange für Timed Waiting.
  - ➔ je zwei *prev* – und *next* – Felder notwendig.

Derzeitige Implementierung in SHAP

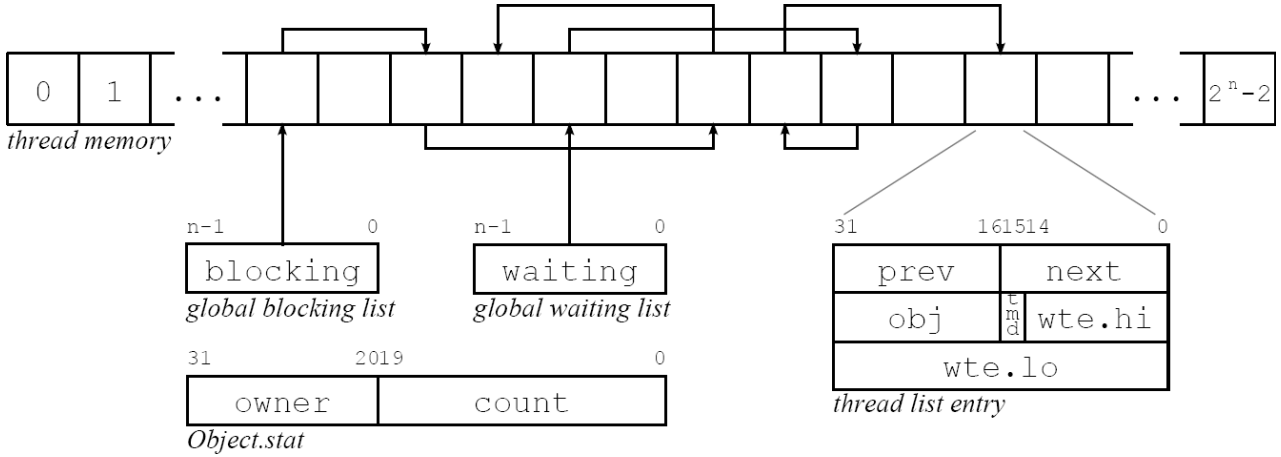
- thread – aktueller Monitorbesitzer
- count – 8 Bit Lockcounter
- ver – Benachrichtigung durch Notify

- Monitore: monitorenter, monitorexit.
- Wait: wait_enter, wait_timed, wait_wait, wait_leave.
- Notify: notify.

Probleme der derzeitigen Implementierung

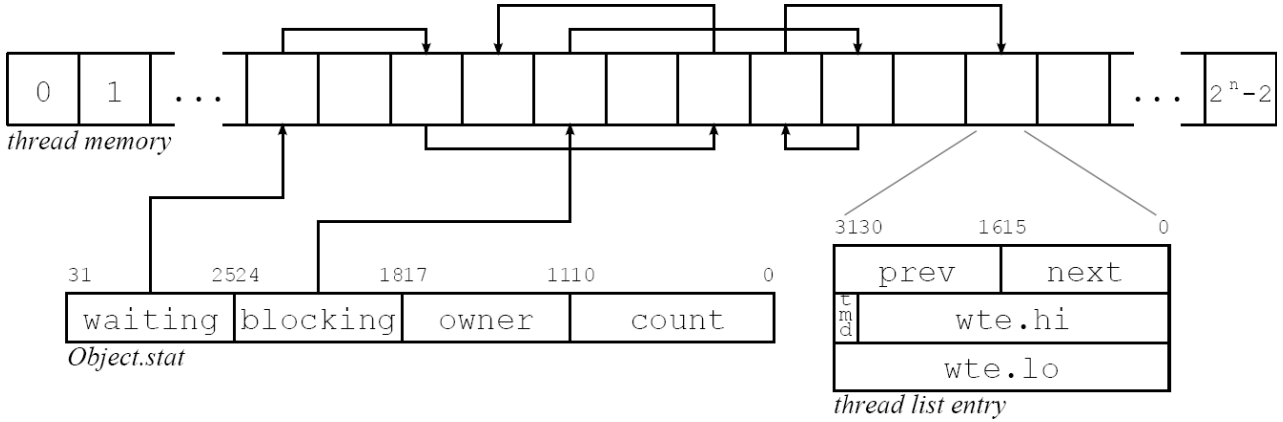
- Polling !!!
- Lockcounterüberlauf möglich (ab 1 kB Stack).
- Vollständiger Umlauf von *Object.ver* möglich.
- Keine Unterscheidung zwischen *Object.notify* und *Object.notifyAll.*
- Sicherheitslücke in *Object.wait*, *Object.notify* und *Object.notifyAll*: Überprüfung, ob aktueller Thread den Monitor hält, fehlt.

Ansatz I



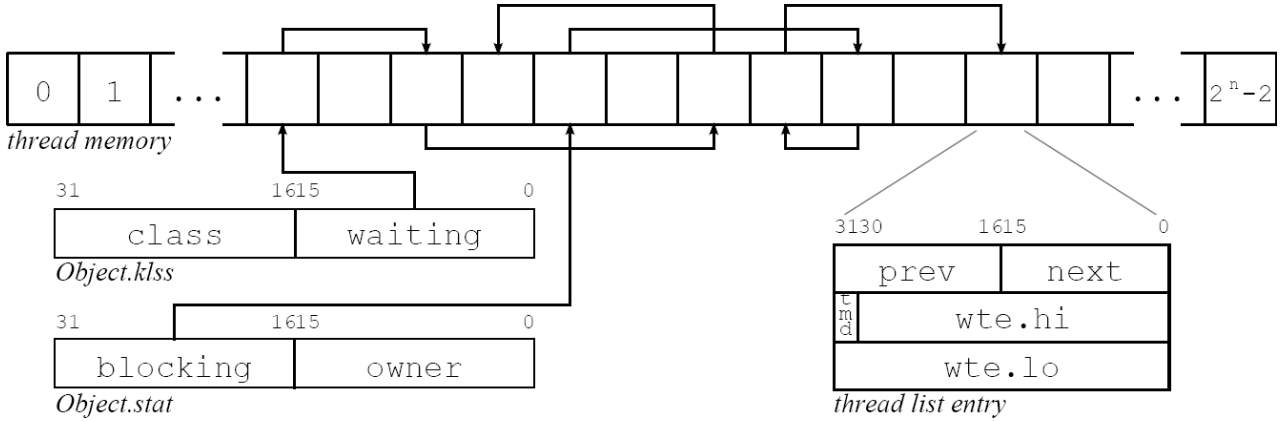
- separate Warteschlangen für blockierende und wartende Threads im Scheduler.
- *owner*: Thread ID anstatt Thread Referenz.
- *ver* - Feld entfällt.
- 20 Bit Lockcounter, Überlauf erst ab 4MB Stack möglich.

Ansatz II (zur Implementierung ausgewählt)



- eigene Warteschlangen für jeden Monitor.
  - ➔ kein Durchsuchen bei Notify oder Freiwerden eines Monitors notwendig.
  - ➔ Für Timed Waiting: zusätzlicher Eintrag in globaler Warteliste (Folie 13).
- 128 Threads, Lockcounterüberlauf ab 8kB Stack möglich.
- Bei Bedarf: niederwertige 16 Bit von *Object.klss* können zur Verbreiterung der Felder verwendet werden.

Ansatz III (später)



- Verzicht auf Lockcounter.
  - ➔ kein Überlauf möglich!
  - ➔ sichern des aktuellen 1 Bit Monitorzustandes auf dem Stack.
- 65535 Threads (aus Sicht der Monitorverwaltung) möglich.
- Modifikation der Semantik von *monitor_enter.*
  - ➔ Patchen des Bytecodes durch den Linker notwendig.
  - ➔ schwer in aktuelle Toolchain integrierbar.

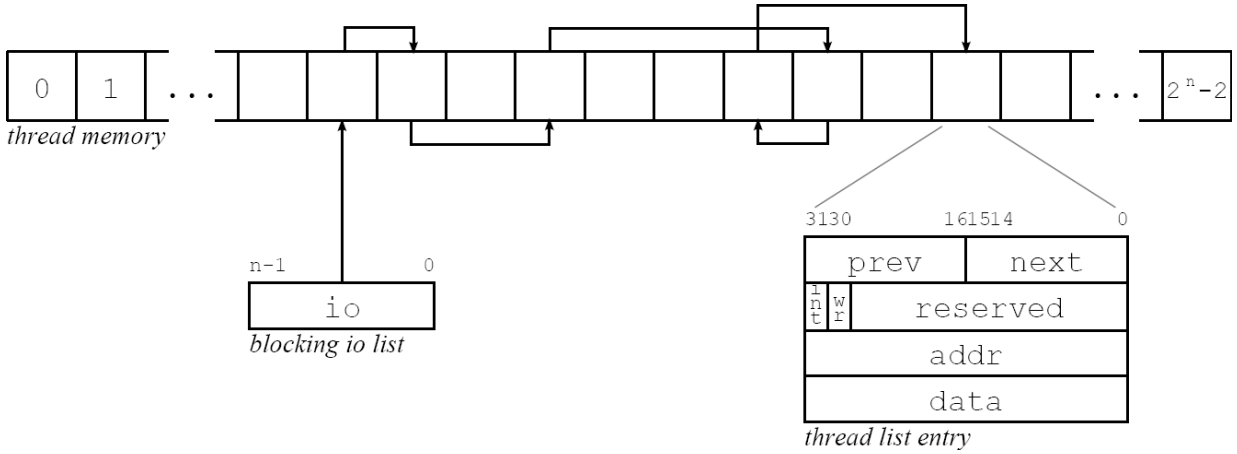
Derzeitige Implementierung in SHAP

- Vier Mikrocodes für blockierende – und nicht blockierende Ein – und Ausgabe: *iord*, *iowr*, *ldios*, *ldiod.*
- Nicht blockierende Ein - / Ausgabe: Rückkehr unabhängig von *iostate.*
- Blockierende Ein - / Ausgabe durch Polling realisiert:

```
if (iostate != READY)
{
  jpc--;
  goto thread_yield;
} else return iodata;
```

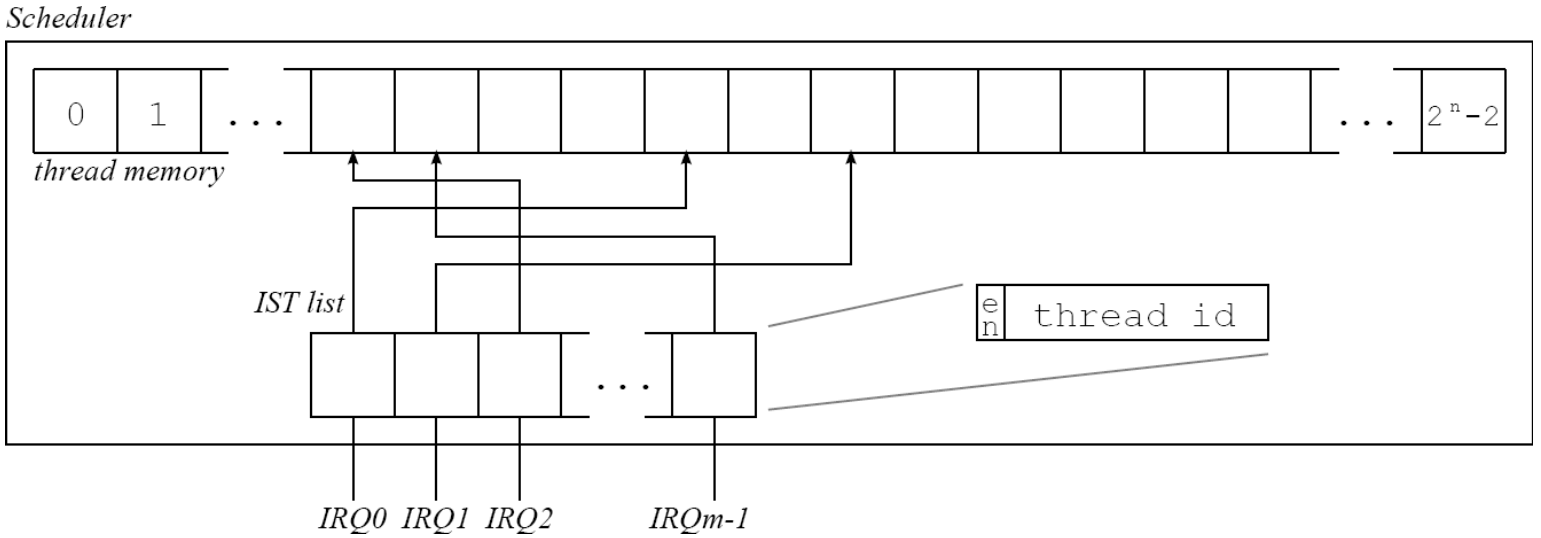
Problem: Polling !!!
Außerdem derzeit keine unterbrechbaren IO Kanäle.

Implementierung im Schedulermodul



- Polling wird anstelle der CPU durch den Scheduler übernommen.
- Blockierende Threads werden erst aufgeweckt, wenn IO Operation erfolgreich.
- Programmierbare Pollingzykluslänge.

Implementierung im Schedulermodul



- Interrupts in SHAP derzeit nur durch pollende Threads simulierbar ➔ hohe Latenz.
- Schedulermodul implementiert „echtes“ Interruptsystem.
- ISTs anstelle von ISRs.

1. Einleitung
2. Threadverwaltung und Scheduling
3. Monitore, Wait und Notify
4. Blockierende Ein – und Ausgabe
5. Interrupts
6. Meilensteine

Meilensteine

- ✓ Scheduler ←→ Wishbone Schnittstelle, Milli – und Nanosekundentimer, programmierbare Zeitscheibenlänge.
- ✓ Scheduler ←→ CPU Schnittstelle, Mikrobefehle **`stsu`**, **`ldsu`**, Spezialbytecode *get_nanos.*
- ➢ Threadverwaltung und Threadscheduling (FPP, RR, GP, EDF).
- • Monitorverwaltung.
- • Blockierende Ein - / Ausgabe.
- • ggf. Interrupts.
- • ggf. weitere Schedulingstrategien.

Literaturverzeichnis (I)

- Preußer, T. B.; Zabel, M.; Reichel, P.:
  „*The SHAP Microarchitecture and Java Virtual Machine*".
  Forschungsbericht, Technische Universität Dresden, Fakultät Informatik, Institut für Technische Informatik, June 11, 2008
- Zabel, M.; Preußer, T. B.; Reichel, P.: „*SHAP Reference Manual*".
  Forschungsbericht, Technische Universität Dresden, Fakultät Informatik, Institut für Technische Informatik, June 11, 2008
- Liu, Jane W. S.: „*Real Time Systems*".
  Prentice Hall, 2000. ISBN 0-13-099651-3
- Tanenbaum, Andrew S.: „Moderne Betriebssysteme".
  Pearson Studium, 2003, 2. Auflage. ISBN 3-8273-7019
- Kay, J.; Lauder, P.: „A Fair Share Scheduler".
  In: Communications of the ACM (1988), p.44-55

Literaturverzeichnis (II)

- Bacon, David F.; Konuru, R.; Murthy, C.; Serrano, M.: *„Thin Locks: Featherweight Synchronization for Java*“. IBM T.J. Watson Research Center
- Kreuzinger, J.; Brinkschulte, U.; Pfeffer, M.; Uhrig, S.; Ungerer, Th.: *„Real-time Event-handling and Scheduling on a Multithreaded Java Microcontroller*“. Microprocessors and Microsystems, Band 27, Nummer 1, Seiten 19-31, 2003

Vielen Dank für Ihre Aufmerksamkeit!