Betriebssysteme Übung
2. Tutorium „System Calls & Multiprogramming“

Wiederholung SysCalls

Wozu?

- Sicherheit
- Stabilität
- Erfordert verschiedene *modes of execution*:

  user mode – privileged mode
- User-Programme dürfen daher I/O u.dgl. nicht selbst ausführen
- → Anforderung an das Betriebssystem

SysCalls – Ablauf

- System call ist **software interrupt**
- Interrupt-Service-Routine im Kernel
- Stellt Typ des system calls fest...
- ...und führt request durch
- wie bei jedem Interrupt: Kontrolle zurück zu Instruktion die dem SC folgt

SysCalls in Nachos

- ausgelöst durch MIPS **syscall-Instruktion**
- **trap** in Nachos-Kernel
- MIPS simulator implementiert traps durch Aufruf der Routine **RaiseException()** (`machine.cc`)
- **ExceptionHandler()** erhält Argument, das die Art der Exception spezifiziert

SysCalls in Nachos

- Instruktionen zur Ausführung des Systemcall

  in `test/`**`start.s`**

  (hier und in `syscall.h` muss noch SC_RENICE hinzugefügt

  werden)
- Code, der den Systemcall angibt, wird in Register 2 geladen
- Zusätzliche Argumente in Registers 4-7
- Rückgabewerte werden in Register 2 erwartet

Beispiel SC_HALT

halt.c

```
#include "syscall.h"

int main()
{
    Halt();
    /* not reached */
}
```

syscall.h

```
#define SC_Halt 0
...
.....
void Halt();
...
```

start.s

```
#include "syscall.h"

    .globl Halt
    .ent   Halt
Halt:
    addiu $2,$0,SC_Halt
    syscall
    j $31
    .end Halt
```

exception.cc

```
Void ExceptionHandler
(ExceptionType which)
{
}
```

Exceptions

SysCall ist nur ein möglicher Exception-Typ, weitere:

- `PageFaultException` keine gültige Speicherseite
- `ReadOnlyException` Write auf “read-only“-Speicher
- `BusErrorException` ungültige phys. Adresse
- `AddressErrorException`
  Zugriff ausserhalb Adressraumes
- `OverflowException` Overflow bei add oder sub
- `IllegalInstrException`
  Ungültige Instruktion

Speicherverwaltung

Wunschliste:

- Abbildung virtueller Adressen auf physische Adressen zur Laufzeit (address translation)
- Jedes Userprogramm beginnt bei Speicherstelle 0
- Memory Management Unit (Hardware!)

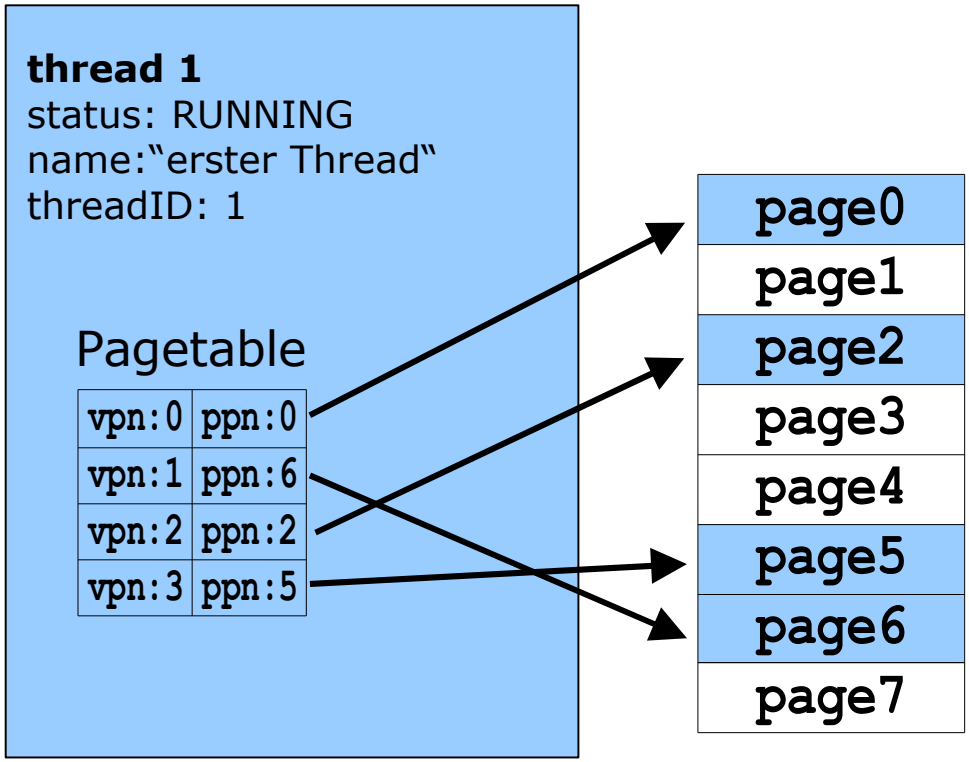
Nachos Hauptspeicher

- Speicher der MIPS-Simulation ist ein einfaches Array
- Speicher ist in **`NumPhysPages`** Seiten der Größe **`PageSize`** (`machine.h`) aufgeteilt
- Speichergröße: **`NumPhysPages*PageSize`**
- Zugriff auf den Hauptspeicher immer via **`Translate(int virtAddr, int* physAddr, ...)`** (`translate.cc`)
- Das alles ist Teil der **Hardware**, darf also nicht verändert werden! (ausgenommen **`NumPhysPages`**)

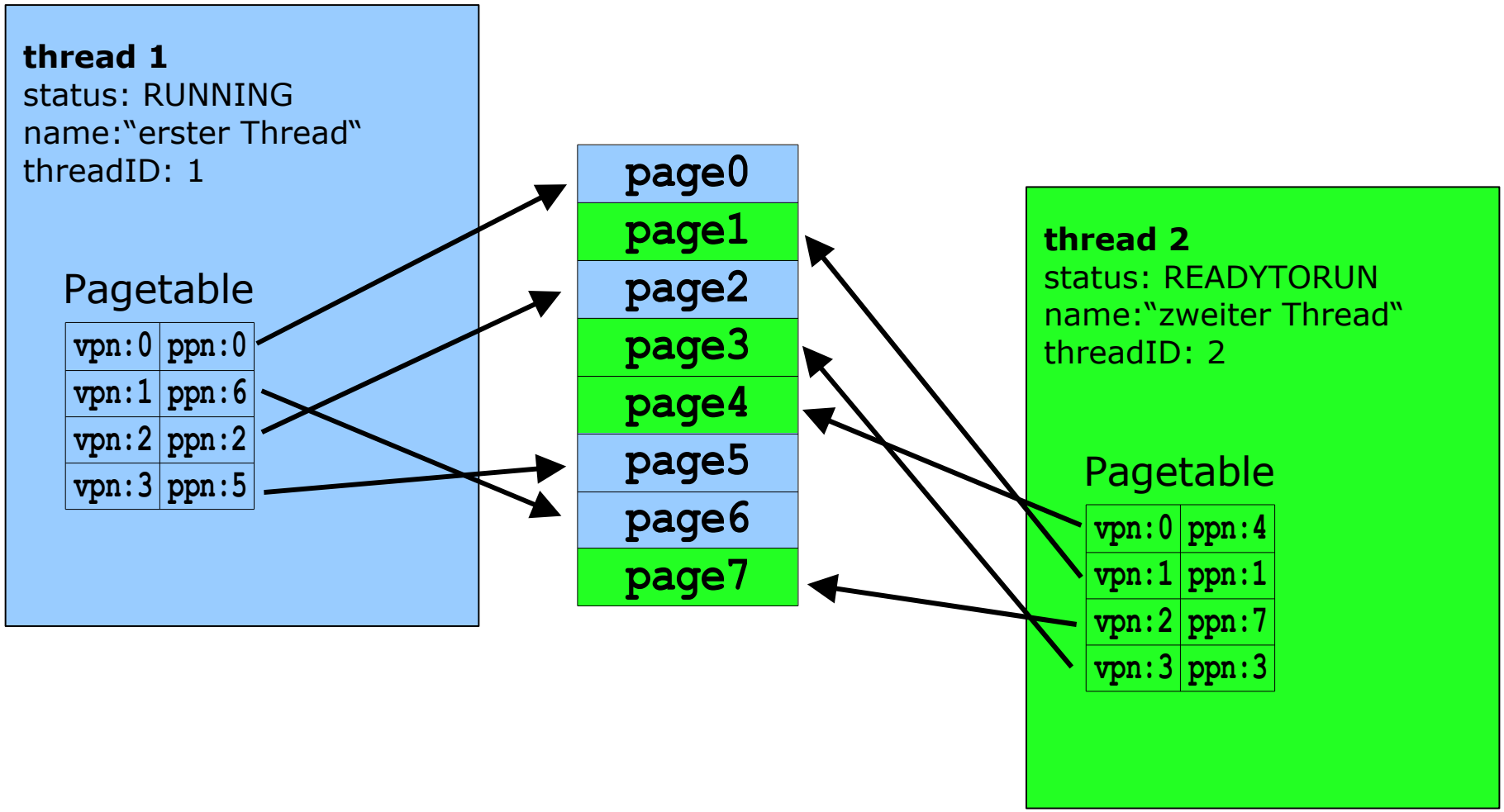
Nachos AddrSpace

- Jeder Thread hat eigenen Adressraum
- derzeit: direkte Abbildung virtueller Adressen in physische Adressen
- multiprogramming: Laden des Programmes ist abzuändern, Speicherseiten müssen anders ausgewählt werden
- Kenntnis der freien Speicherseiten nötig
- →Ändern des AddrSpace-Konstruktors nötig
- relevante Variablen: **`pageTable`**, **`numPages`**

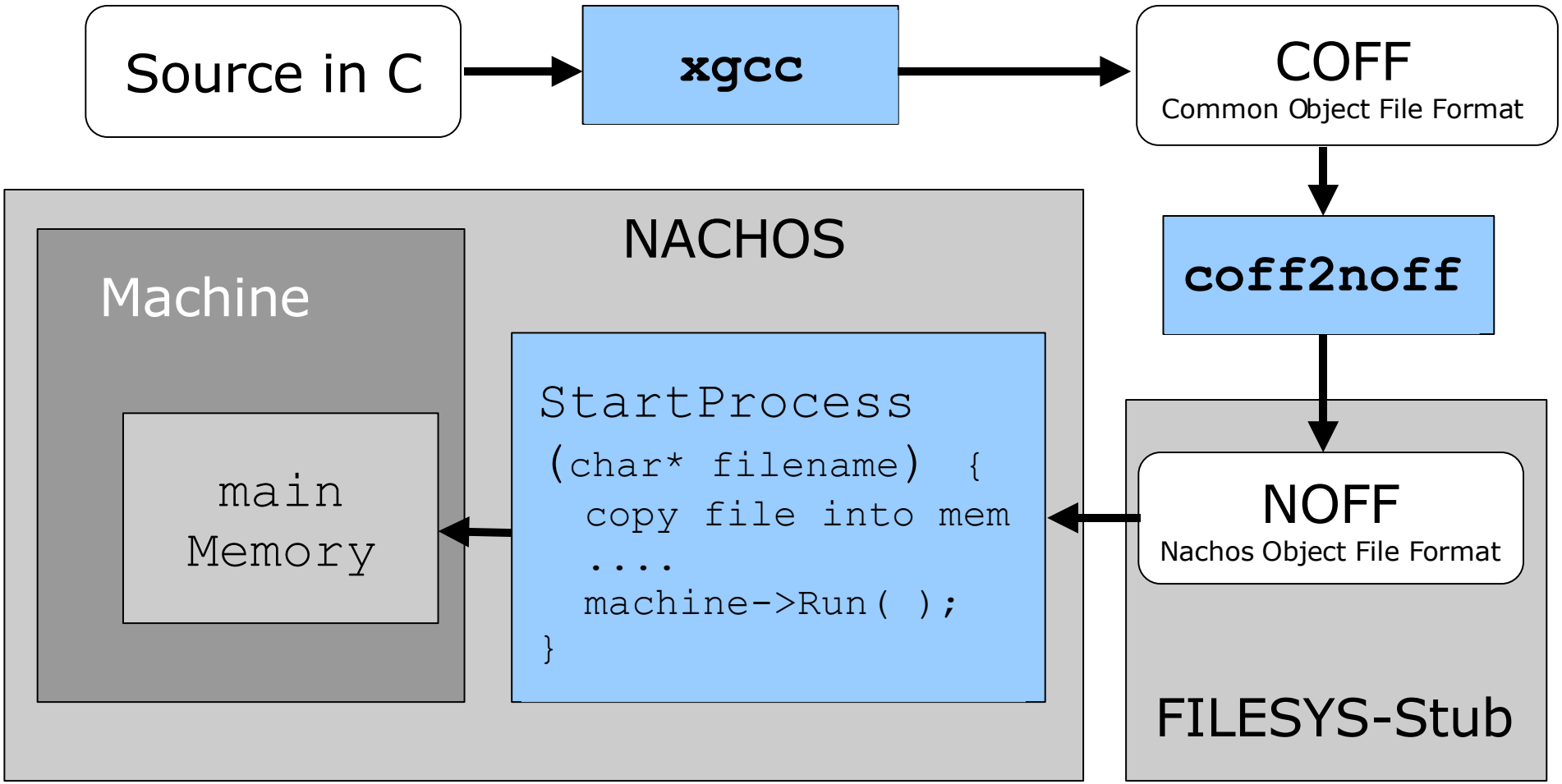
Speicherverwaltung, 1 Thread



Speicherverwaltung, mehrere Threads



Userprogramme



Userprogramme

Verwaltungsaufgaben:

- Wer ist der Eltern-Prozess?
- Welche Ressourcen verwendet der Prozess?
- Was tun mit den Kind-Prozessen wenn ein Prozess beendet wird?
- Was tun, wenn ein Prozess fehlerhaft läuft?
- Was machen wir mit offenen Files?

Userprogramme

Starten eines Userprogrammes:

- prüfen, ob genug Speicher verfügbar
- Neuen Adressraum anlegen, Binary laden
- neuen Thread für den Prozess erzeugen, Adressraum zuweisen und **`Fork()`**en
- Register/Userstate initialisieren
- Pointer auf Programm-Parameter in Register 4 schreiben
- **`machine->Run()`**

Timeslicing mit Prioritäten

Verschiedene Varianten möglich:

- Verlängern der Abarbeitungszeit (Ansatzpunkte: n-Ticks bis Timerausloesung, n-mal gleichen Prozess auswählen, ...)

  Vorteil: einfach zu implementieren

  Nachteil: System „ruckelt“ wenn Prozesse mit sehr hoher Priorität laufen
- Prozess öfters laufen lassen (statistische Auswahl aus lauffähigen Prozessen)

  Vorteil: bessere Parallelität und Antwortverhalten

  Nachteil: aufwändig zu implementieren

Implizite Aufgabenstellungen

Wir wollen:

- Stabiles, fehlertolerantes Betriebssystem
- Prozesse sollen voreinander geschützt werden
- fehlerhafter Prozess darf das Betriebssystem und andere Prozesse nicht gefährden
- Welche Fehlerfälle können auftreten, wie sollen diese behandelt werden?