Betriebssysteme Übung
Tutorium „System Calls & Multipgrogramming“

Assignment 2 – Userprogramme

- Userprogramme laufen innerhalb der MIPS-Simulation
- Mit `xgcc` kompilieren
- Beispiele im Verzeichnis `test/`
- Beispielcode zum Starten eines einzelnen Userprogrammes in `userprog/progtest.cc`, wird mit `nachos -x <file>` ausgeführt
- So kann auch beim Multiprogramming der erste Prozess (z.B. eine Shell) geladen werden

Task 1 – System Calls (1)

Was ist ein Syscall?

- Userprogramme wollen Betriebssystemfunktionen verwenden
- Dazu lösen sie eine Exception (Softwareinterrupt) aus
- Diese wird vom Exception-Handler im Kernel abgefangen (`userprog/exception.cc`, `userprog/syscall.h`)
- Parameter werden in definierten CPU-Registern übergeben
- Nach der Bearbeitung wird die Kontrolle wieder an das Userprogramm übergeben

Task 1 – System Calls (2)

zu implementieren:

- `SC_EXEC` neues Userprogramm (aus Image-Datei) starten
- `SC_JOIN` auf EXIT eines anderen Prozesses (hier nicht mehr Thread* sondern numerische PID) warten und dessen Exitstatus zurückliefern
- `SC_EXIT` Prozess beenden und Exitstatus merken
- `SC_YIELD` Thread-Wechsel
- `SC_RENICE` Prozesspriorität ändern

Task 1 – System Calls (3)

weiters: Dateioperationen

- `SC_CREATE`, `SC_OPEN`, `SC_WRITE`, `SC_READ`, `SC_CLOSE`
- Wrapper auf das Linux-Filesystem
- Filedeskriptoren 0 und 1 für Konsolenein- und ausgabe reserviert und automatisch geöffnet (`machine/console.cc`)
- Beim Beenden/Killen eines Programmes sollen noch geöffnete Dateien wieder geschlossen werden
- Daher Verwaltung der geöffneten Filedeskriptoren und Zuordnung zu den Prozessen nötig (`userprog/bitmap.cc`).

Task 2 – Multiprogramming (1)

Timeslicing

- Timer, der regelmässig einen Interrupt auslöst
- Jeder Thread (Userprogramm) läuft eine bestimmte Anzahl von Ticks, dann wird (zwangsweise!) ein Context-Switch (Yield) ausgeführt
- eine Möglichkeit Prozessprioritäten zu implementieren (eine andere: im Scheduler mehrmals den selben Thread auswählen)
- Timer in `machine/timer.cc` bereits implementiert
- Interrupthandler wird bereits angelegt (`thread/system.cc`, `machine/interrupt.cc`)

Task 2 - Multiprogramming (2)

Speicherverwaltung:

- freie/verwendete Speicherseiten müssen bekannt sein (`userprog/bitmap.cc`)
- Jeder Prozess verwendet seinen eigenen Adressraum, d.h. er adressiert intern Speicher ab der virtuellen Adresse 0!
- Übersetzung von virtueller auf physikalische Adresse notwendig
- Jeder Prozess hat eine PageTable (`userprog/addrspace.cc`)

Task 2 - Multiprogramming (3)

Speicherverwaltung:

- der physikalische Speicher (`/machine/machine.h`) ist groß genug um alle Userprogramme aufzunehmen (VM erst in Assignment 3), es genügt beim Laden beliebige (freie) physikalische Seiten zu belegen
- Bei jedem Context-Switch wird die PageTable des neuen Prozesses geladen
- Bei Nachos belegt der Kernel keinen Speicher

Task 3 – Exec mit Parameterübergabe

- Userprogramme sollen beim Start Parameter (einen String) übergeben bekommen
- Eine Möglichkeit: Parameter auf einer zusätzlichen Speicherseite im Adressraum des Programmes ablegen
- `int main(char* argv)`

  Der char* steht per Definition in CPU-Register 4.

  Er muss natürlich auch auf eine virtuelle Adresse umgerechnet werden!