UNIVERSITY OF AARHUS

Faculty of Science

Department of Engineering

Indlejret Software Udvikling
Eksamens Dispositioner

Bjørn Nørgaard
IKT
201370248
bjornnorgaard@post.au.dk

Joachim Andersen
IKT
20137032
joachimdam@post.au.dk

Sidste ændring: December 14, 2015
LaTeX-koden kan findes her[1]

---

[1] https://github.com/BjornNorgaard/I3ISU/tree/master/Eksamen

Todo list

Indholdsfortegnelse

List of Figures

1 Programs in relation to the OS and the kernel

1.1 Sub topics

- Processes and threads.

- Threading model.

- Process anatomy.

- Virtual memory.

- Threads being executed on CPU, the associated scheduler and cache.

1.2 Curriculum

- Slides "Intro to OS's".

- Slides "Parallel programs, processes and threads".

- OLA: "Anatomy of a program in memory", Gustavo Duarte.

- OLA: "The free lunch is over".

- OLA: "Virtual memory", pages 131-141.

- OLA: " Introduction to operating systems".

- OLA: "Multithreading".

- Kerrisk: Ch. 3-3.4 - System programming concepts.

- Kerrisk: Ch. 29 - Threads: Introduction.

1.3 Exercises

- Posix Threads.

1.4 Processes and threads

- En **process** er en instans af et program, som eksekveres.

- En **thread** er en del af eksekveringen, alle processer har mindst én thread.

Processes

- Har hver sit memory space.

- Process A kan ikke skrive i Process B's hukommelse.

- Kan kun kommunikere gennem IPC[1]

- Kan skabe andre processer som kan eksekvere det samme eller andre programmer.

Threads

- Alle tråde i en process dele hukommelse på heap'en.

- Alle tråde har hver sin stack og program counter.

- Kan fucke med hinanden

  - Skal passe på at man ikke sletter de øvrige trådes data.

hvad vil program counter sige?

1.5   Threading model

Der findes tre forskellige modeller:

- User level threading.

- Kernel level threading.

- Hybrid level threading.

User level threading

- Simpel implementering, ingen kernel support for threads.

- Ekstremt hurtig thread kontekst skift (ikke brug for kernel handling).   **woot?**

- Ikke muligt at håndtere flere kerner.   **hvorfor?**

Figure 1: User level threading illusteret.

---

[1] Inter-Process Communication, mekanismer kontrolleret af OS.

Kernel level threading

- Brug for thead bevisthed i kernel.

- Mapper direkte til threads som *scheduleren* kan kontrollere.

- Effektiv brug af flere kerner.

Figure 2: Kernel level threading illusteret.

Hybrid level threading

- Komplex implementering.   why?

- Kræver god koordination mellem userspace og kernelspace scheduleren - ellers ikke optimal brug af resources.

Figure 3: Hybrid level threading illusteret.

1.6   Process anatomy

- Når et program startes, starter en ny process.

- En process kører i sin egen memory sandbox, som et *virtual address space* (4GB på 32-bit platform).

- Hver process har sin egen **page table/virtual address space**.

1.7   Virtual memory

1.8   Threads being executed on CPU, associated scheduler and cache

2 Synchronization and protection

2.1 Sub topics

- Data integrity - Concurrency challenge.

- Mutex and Semaphore.

- Mutex and Conditionals.

- Producer / Consumer problem.

- Dining philosophers.

- Dead locks.

2.2 Curriculum

- Slides: "Thread Synchronization I and II".

- Kerrisk: Chapter 30: Thread Synchronization.

- Kerrisk: Chapter 31: Thread Safety and Per-Thread Storage (Speed read)".

- Kerrisk: Chapter 32: Thread Safety and Per-Thread Storage (Speed read)".

- Kerrisk: Chapter 53: Posix Semaphores (Named not in focus for this exercise)".

- OLA: "pthread-Tutorial" - chapters 4-6.

- OLA: "Producer/Consumer problem".

- OLA: "Dining Philosophers problem".

2.3 Exercises

- Posix Threads

- Thread Synchronization I & II

3 Thread communication

3.1 Sub topic

- The challenges performing intra-process communication.

- Message queue.

    - The premises for designing it.
    - Various design solutions - Which one chosen and why.
    - Its design and implementation.

- Impact on design/implementation between before and after the Message Queue.

- Event Driven Programming.

    - Basic idea.
    - Reactiveness.
    - Design - e.g. from sequence diagrams to code (or vice versa).

3.2 Curriculum

- Slides: "Inter-Thread Communication".

- OLA: "Event Driven Programming: Introduction, Tutorial, History - Pages 1-19 & 30-51".

- OLA: "Programming with Threads - chapters 4 & 6".

3.3 Exercises

- Thread Communication

4   OS API

4.1   Sub topics

- The design philosophy - Why OO and OS Api?

- Elaborate on the challenge of building it and its currenct design:

    - The PIMPL / Cheshire Cat idiom - The how and why.
    - CPU / OS Architecture.

- Effect on design/implementation:

    - MQs (Message queues) used with pthreads contra MQ used in OO OS Api.
    - RAII in use.
    - Using Threads before and now.

- UML Diagrams to implementation (class and sequence) - How?

4.2   Curriculum

- Slides: OS Api”.

- OLA: OSAL SERNA SAC10”.

- OLA: Speciffcation of an OS Api”.

- Kerrisk: Chapter 35: Process Priorities and Schedul-ing”.

4.3   Exercises

- OS API.

5 Message Distribution System (MDS)

5.1 Sub topics

- Messaging distribution system - Why & how?

- The PostOffice design - Why and how?

- Decoupling achieved.

- Design considerations & implementation.

- Patterns per design and in relation to the MDS and PostOffice design:

  - GoF Singleton Pattern
  - GoF Observer Pattern
  - GoF Mediator Pattern

5.2 Curriculum

- Slides: "A message system".

- OLA: "GoF Singleton pattern".

- OLA: "GoF Observer pattern".

- OLA: "GoF Mediator pattern".

5.3 Exercises

- The Message Distribution System

6 Resource handling

6.1 Sub topics

- RAII - What and why?

- Copy construction and the assignment operator.

- What is the concept begind a Counted SmartPointer?

- What is $boost :: shared\_ptr <>$ and how do you use it?

6.2 Curriculum

- Slides: ”Resource Handling”.

- OLA: ”RAII - Resource Acquisition Is Initialiation”.

- OLA: ”SmartPointer”.

- OLA: ”Counted Body”.

- OLA: ”$boost :: shared\_ptr$”.

- OLA: ”Rule of 3”.

6.3 Exercises

- Resource Handling.