Hausaufgabe 1

Gruppe 1

Kevin Bock
Philipp Kückes
Dora Szücs
Sarah Köhler

Aufgabe 2.1

a)

- Prozesse sind unabhängig von einander
- Threads teilen den selben Adressraum während jeder Prozess einen eigenen hat
- geringerer Overhead bei Threads / weniger Ressourcen benötigt
- Threads sind schneller zu erzeugen und zu beenden als Prozesse
- Zwischen Threads kann schneller gewechselt werden als zwischen Prozessen (da der Kontext nicht gewechselt werden muss)
- Kommunikation zwischen Threads ist einfacher als zwischen Prozessen (bei Prozessen muss immer der Kernel die Kommunikation unterstützen, dagegen können Threads direkt, z.B. über Variablen kommunizieren)
- Prozesse: mehr Sicherheit bei verschiedenen Nutzern durch strikte Trennung, geschützer Zugriff auf Ressourcen
- Prozess kann aus mehreren Threads bestehen
- Threads werden meist für geringere Aufgaben verwendet
- ein einzelner Threads kann den gesamten Prozess lahmlegen, wenn er blockiert
- Daten können von allen Threads eines Prozesses zugleich genutzt und verändert werden, da sie den selben Adressraum teilen -> Programmierer muss für Integrität der Daten sorgen

Anwendungsfälle von Threads:
- wenn häufige Kommunikation nötig ist
- wenn gemeinsamer Zugriff auf Daten nötig ist, z.B. bei der parallelen Berechnung von Matrizen
- bei vielen ähnlichen / verwandten Aufgaben
- Aufteilung von Hintergrund-Aufgaben und Verarbeitung von Nutzereingaben in einem Programm (Bsp. Textverarbeitung: Ein Thread verarbeitet die Eingaben während ein zweiter Thread im Hintergrund die Eingaben auf Rechtschreibung prüft)
- bei vielen I/O-Operationen, da dann ein Thread die Daten lesen kann während ein anderer die bereits gelesenen Daten verarbeitet

Anwendungsfälle von Prozessen:
- sicherheitskritische Anwendungen, wo verhindert werden muss, dass andere Nutzer auf die Daten des Prozesses zugreifen
- verschiedene Anwendungsprogramme

b)

Nutzen von Threads bei der Implementierung eines Videoservers:

Vorteile:

- erzeugen grundsätzlich weniger Overhead

- weniger Ressourcen benötigt bei der Bereitstellung der selben Videodateien für verschiedene Empfänger, da Threads auf die selben Daten im Cache zugreifen können
- mehrere Threads (des selben Prozesses) können auf den selben Socket zugreifen, da sie im selben Adressraum arbeiten
- schneller Wechsel zwischen Threads möglich -> geringe gefühlte Verzögerung für die Nutzer des Video-Dienstes
- einfache Kommunikation zwischen Threads, z.B. über Variablen
- Threads können schnell erzeugt und beendet werden

Nachteile:

- ein Fehler bei der Videoübertragung kann den Prozess für mehrere Nutzer blockieren

Organisation:

Es gibt eine maximale Anzahl von Threads. Neue Anfragen werden in einer Queue gespeichert und dann von freien Threads übernommen. Jeder Thread ist für eine TCP-Verbindung zuständig und liefert alle Daten über diese Verbindung aus. Sobald diese geschlossen wurde, kann der Thread eine neue Anfrage aus der Queue bearbeiten. So kann sichergestellt werden, dass nie zu viele Threads gleichzeitig erzeugt werden. Um Ressourcen zu sparen, können bei geringer Last Threads beendet werden (was ebenfalls mit geringem Aufwand möglich ist) und erst bei Bedarf wieder neu erzeugt werden. Dazu überwacht ein zusätzlicher Thread die Queue sowie die Anzahl der aktiven Threads und erzeugt bei Bedarf einen neuen Thread.
Auch dies spricht für Threads, da die Kommunikation zwischen Threads einfacher zu organisieren ist als zwischen Prozessen.

Aufgabe 2.2

fork/join:

P0
fork rechts
P1
fork P3
P4
join
P3
P7
join rechts
P11
rechts:
    P2
    fork P5
    P6
    fork P9
    P8
    join P9
    join P5
    P10

parbegin/parend:

```
P0
parbegin
        begin
                P1
                parbegin
                        P3
                        P4
                parend
                P7
        end
        begin
                P2
                parbegin
                        P5
                        begin
                                P6
                                parbegin
                                        P8
                                        P9
                                parend
                        end
                parend
                P10
        end
parend
P11
```