Probeklausur

Systemprogrammierung

Universität:	Technische Universität Berlin
Kurs/Modul:	Systemprogrammierung
Bearbeitungszeit:	90 Minuten
Erstellungsdatum:	September 6, 2025

Bearbeitungszeit: 90 Minuten.

Aufgabe 1.

(a) Beschreiben Sie den Unterschied zwischen Prozess, Thread und Kernel-Thread; geben Sie zu jeder Kategorie die Hauptcharakteristika an.

(b) Skizzieren Sie zwei Scheduling-Verfahren, deren Merkmale und typische Vor- bzw. Nachteile: First-Come, First-Served und Round-Robin mit Zeitscheibe.

(c) Erläutern Sie das Phänomen Race Condition und nennen Sie zwei propagierte Synchronisationsmechanismen, die solche Probleme verhindern können.

(d) Nennen Sie zwei Formen der Prozesskommunikation und erläutern Sie deren wesentliche Eigenschaften hinsichtlich Datenfluss und Synchronisation.

Aufgabe 2.

(a) Beschreiben Sie den Unterschied zwischen virtuellem Speicher und physischem Speicher. Erklären Sie kurz Paging, Seitentabellen und die Rolle der Translation Lookaside Buffer (TLB).

(b) Skizzieren Sie grob den Ablauf einer Interrupt-Verarbeitung von der Unterbrechung bis zum Kontextwechsel.

(c) Erläutern Sie die Grundidee einer Seitenersetzung und nennen Sie ein realistisches Beispiel einer Ersetzungsstrategie, beispielsweise LRU.

(d) Beschreiben Sie die Rolle von Gerätetreibern in einem Betriebssystem und wie sie typischerweise mit dem Kernel kommunizieren.

Aufgabe 3.

(a) Benennen Sie die vier klassischen Bedingungen, die einen Deadlock verursachen können.

(b) Diskutieren Sie zwei Strategien zur Vermeidung oder Vermeidung von Deadlocks in Mehrprozessorsystemen.

(c) Beschreiben Sie den Unterschied zwischen Semaphoren und Mutexen und geben Sie an, welche Synchronisationsprobleme sie jeweils adressieren.

(d) Erklären Sie die Bedeutung von Atomic-Operationen und Memory-Barriers in Mehrprozessorsystemen.

Aufgabe 4.

(a) Beschreiben Sie den Ablauf eines Interrupt-Handling-Zyklus aus Sicht eines Prozessors, einschließlich Priorisierung und Kontextwechsel.

(b) Vergleichen Sie die Auswirkungen eines Monolithischen Betriebssystems mit einem Microkernel-Ansatz auf Treiberstrukturen und Sicherheit.

(c) Erläutern Sie, wie das Betriebssystem Ressourcen wie CPU-Zeit, Speicher und I/O busbar hierarchisch verwaltet und welche Rolle Scheduling-Policies dabei spielen.

(d) Nennen Sie zentrale Sicherheitsaspekte für sichere Betriebssysteme, insbesondere Speicherschutz, Prozess-Isolation und robuste Interrupt-Verarbeitung, und diskutieren Sie deren Bedeutung für kritische Infrastrukturen.

Lösungen

Lösungen zu den Aufgaben des Skripts

Aufgabe 1.

(a) Unterschied zwischen Prozess, Thread und Kernel-Thread; Hauptcharakteristika

Lösung:

- *Prozess*:
  - eigenständiger Ausführungskontext mit eigenes Adressraum (virtueller Speicher), eigener Ressourcen (Dateideskriptoren, Dateien, IPC-Objekte),
  - besitzt einen Prozesszustand (PCB/Task-Control-Block) und wird vom Scheduler verwaltet,
  - Kommunikation mit anderen Prozessen erfolgt typischerweise über Interprozesskommunikation (IPC),
  - Kontextwechsel zwischen Prozessen ist relativ teuer (Privatsphäre des Adressraums, SSP/TLB-Cache-Verluste).
- *Thread* (user-space Thread, „Light-Weight-Thread“ innerhalb eines Prozesses):
  - teilt denselben Adressraum (heap, globale Variablen) mit anderen Threads des gleichen Prozesses,
  - besitzt eigenen Stack und Programmzähler (PC) – unabhängig vom anderen Threads,
  - Context-Switch ist leichter als bei Prozessen; schnellerer Zugriff auf gemeinsam genutzte Daten,
  - Scheduling erfolgt innerhalb des Prozesses durch den Prozess-Scheduler bzw. das Betriebssystem-Scheduler-Subsystem.
- *Kernel-Thread*:
  - Threads, die im Kernel-Modus laufen, vom Kernel-Scheduler verwaltet,
  - arbeiten im Kernel-Adressraum (nicht sichtbar als normale Benutzerprozesse),
  - dienen der Ausführung von kernel-spezifischen Aufgaben (z. B. Treiber- oder Scheduler-Komponenten) unabhängig von Benutzerprozessen,
  - kommunizieren oft mittels Kernel-API mit anderen Kernel-Diensten (z. B. Interrupt-Handling, Deferred Work).

(b) Zwei Scheduling-Verfahren: Merkmale, Vor- und Nachteile

Lösung:

- **First-Come, First-Served (FCFS)**:
  - Merkmale: einfach implementierbar; nicht-präemptiv; nach Ankunft sortiert.
  - Vorteile: geringe Implementierungskomplexität; vorhersehbar in einfachen Systemen.

  - Nachteile: Convoy-Effekt (lange Prozesse blockieren nachfolgende), lange Wartezeiten im Durchschnitt, schlechte Ausnutzung bei heterogenen Arbeitslasten.

- **Round-Robin (RR) mit Zeitscheibe**:
  - Merkmale: präemptiv; fester Quantumszeitschlitz Q; Kontextwechsel pro Sub-Task.
  - Vorteile: faire Verteilung der CPU-Zeit; reaktionsschnell bei interaktiven Anwendungen.
  - Nachteile: Wahl der Zeitscheibe ist kritisch (zu klein -> Kontextwechsel-Overhead); zu groß -> Annäherung an FCFS; geringe Effizienz bei CPU-bound Tasks.

(c) Race Condition und zwei Synchronisationsmechanismen zur Vermeidung

Lösung:

- *Race Condition*: Tritt auf, wenn mehrere Threads/Prozesse auf gemeinsam genutzte Daten zugreifen und das Endergebnis vom zeitlichen Ablauf der Ausführung abhängt. Ohne geeignete Synchronisation können inkonsistente Zustände entstehen.
- *Synchronisationsmechanismen zur Vermeidung*:
  - **Mutex** (Mutual Exclusion): Binärer Sperrmechanismus; besitzt Eigentümerschaft; nur der inhabernde Thread darf den kritischen Abschnitt betreten; verhindert gleichzeitigen Zugriff.
  - **Semaphor** (P/V-Operationen; Zähler): Zählende oder binäre Semaphoren; ermöglichen Synchronisation (Signalisierung, Warten) bzw. Ressourcenmanagement; erlaubt kontrollierte Mehrfachzugriffe bzw. Koordination zwischen Threads.

(d) Zwei Formen der Prozesskommunikation; Eigenschaften bzgl. Datenfluss und Synchronisation

Lösung:

- **Gemeinsamer Adressraum (Shared Memory)**:
  - Daten werden direkt im gemeinsam genutzten Adressraum gelesen/geschrieben,
  - sehr hohe Durchsatzrate, geringe Latenz,
  - erfordert explizite Synchronisation (Locking, Barriers), um Race Conditions zu vermeiden.
- **Nachrichtenbasiert (Message Passing)**:
  - Prozesse kommunizieren über Nachrichtenkanäle (Pipes, Sockets, Message Queues),
  - klare Schnittstelle, geringere Fehlerrisikoeinstreuungen durch Datenkapselung,
  - Synchronisation wird durch das Kommunikationsprotokoll bereitgestellt; kann blocking/non-blocking sein, oft asynchron durch Messaging-Queues.

Aufgabe 2.

(a) Virtualer Speicher vs. physischer Speicher; Paging, Seitentabellen, Rolle des TLB

Lösung:

- *Virtueller Speicher*:
  - jeder Prozess hat seinen eigenen, schützbaren Adressraum (virtuelle Adressen),
  - Abbildung durch Seitentabellen-Mechanismus; Schutzgrenzen zwischen Prozessen; größerer logischer Adressraum als physischer RAM.
- *Physischer Speicher*:
  - echter RAM, in Frames unterteilt; Adressraum ist endlich und limitiert durch Hardware.
  - Paging ermöglicht flexible Nutzung, Paging-Out/Swapping bei Bedarf.
- *Paging, Seitentabellen, TLB*:
  - Paging zerlegt Adressraum in feste Seiten (virtuell) und Seitenrahmen (physisch),
  - Seitentabellenordner speichern Zuordnung (Virtuelle Seite -> Physischer Rahmen),
  - TLB (Translation Lookaside Buffer) als schneller Cache der Seitentabellen-Einträge; reduziert Zugriffskosten auf Speicher-Translationspfad.

(b) Grobe Ablauf einer Interrupt-Verarbeitung von Unterbrechung bis Kontextwechsel

Lösung:

- Interrupt tritt durch Peripherie/HW auf;
- CPU speichert Kontext des aktuellen Threads/Prozesses (Registersatz, Programmzähler) auf Stack; ggf. Interrupt-Priorität wird berücksichtigt;
- Sprung zum Interrupt-Handler (ISR) über Vektor-Tabelle; Interrupt wird maskiert bzw. priorisiert verarbeitet;
- ISR führt so kurze Arbeit wie möglich aus; ggf. Deferred Work/Bottom-Half (DPC) wird geplant;
- ISR beendet, ggf. Kernel-Scheduler aktiviert, um zu entscheiden, ob ein Kontextwechsel notwendig ist;
- Rückkehr aus dem Interrupt (IRET/RET) mit ggf. neu geladenem Kontext (neues Task- oder User-Kontext);
- Falls nötig, aktueller Task durch Scheduler ersetzt (Kontextwechsel).

(c) Grundidee einer Seitenersetzung und realistisches Beispiel einer Ersetzungsstrategie (z. B. LRU)

Lösung:

- *Seitenersetzung* (Page Replacement): Bei einem Page-Fault muss eine Seite aus dem physischen Speicher entfernt werden, um Platz für die geforderte virtuelle Seite zu schaffen.
- *Beispiel einer Ersetzungsstrategie*:
  - **LRU (Least Recently Used)**: Die Seite, die am längsten nicht mehr verwendet wurde, wird ersetzt, da sie am unwahrscheinlichsten erneut gebraucht wird.
  - Praktische Implementierung: Näherung durch Clock-Algorithmus oder Second-Chance-Algorithmus, um Kosten zu reduzieren.

(d) Rolle von Gerätetreibern und wie sie typischerweise mit dem Kernel kommunizieren

Lösung:

- *Rolle der Treiber*: Übersetzen von Betriebssystem-API-Aufrufen in Hardware-Operationen; Bereitstellung eines abstrakten Interfaces für Peripherie; Verwaltung von Pufferungen, DMA, Interrupts und Fehlerbehandlung.
- *Kommunikation mit dem Kernel*:
  - Treiber implementieren Kernel-Interfaces (Dateisystem- bzw. Gerätedatei-Interfaces), verwenden Kernel-APIs (Wait Queues, Sleep/Wleep, Mutexen, Spinlocks),
  - Zugriff auf Hardware typischerweise via Speicherkarten (Memory-Mapped I/O) oder IO-Ports; DMA-Transfers koordiniert der Treiber,
  - Geräte erzeugen/legen Interrupts fest; Kernel liefert Mechanismen zur Registrierung von Interrupt-Service-Routinen (ISRs) und zur späteren Verarbeitung mittels Bottom-Half/Deferred Work.

Aufgabe 3.

(a) Vier klassische Bedingungen, die Deadlocks verursachen können

Lösung:

- *Mutual Exclusion* (getrennte Ressourcen) – Ressourcen können von nur einem Prozess gleichzeitig genutzt werden.
- *Hold and Wait* – Prozesse halten bereits zugewiesene Ressourcen und warten auf weitere.
- *No Preemption* – Ressourcen können nicht gewaltsam entzogen werden.
- *Circular Wait* – eine geschlossene Warteschlange von Prozessen, die auf Ressourcen warten.

(b) Zwei Strategien zur Vermeidung bzw. Vermeidung von Deadlocks in Mehrprozessorsystemen

Lösung:

- **Ressourcen-Ordering (globale Ressourcennummerierung)**:
  - Alle Ressourcen werden in einer globalen Reihenfolge angefordert (z. B. nach Ressourcennummer),
  - ein Prozess muss Ressourcen in aufsteigender Reihenfolge anfordern; verhindert kreisende Wartebedingungen.
- **Banker's Algorithm** (Sicherheitsüberprüfung bei Reservierung von Ressourcen):
  - Der Scheduler gewährt nur Anfragen, wenn danach noch ein sicherer Zustand möglich ist,
  - verhindert Deadlocks durch Vorhalten von Ressourcen nur, wenn das System nach der Zuweisung in einen sicheren Zustand kommt.

(c) Unterschied zwischen Semaphoren und Mutexen; adressierte Synchronisationsprobleme

Lösung:

- *Mutex*:
  - besitztonymes Sperrenkonzept; nur der Besitzer einer Mutex darf sie entsperren;
  - primär für exklusiven Zugriff auf Critical Sections vorgesehen (Beitrag zur Mutual Exclusion).
- *Semaphore*:
  - Zähler-basiert; kann binär (0/1) oder zählend sein;
  - dient sowohl der Synchronisation als auch dem Ressourcen-Counting; ermöglicht Signalisierung (Producer/Consumer) und Koordination.

(d) Bedeutung von Atomic-Operationen und Memory-Barriers in Mehrprozessorsystemen

Lösung:

- *Atomic-Operationen* – Operationen, die ununterbrochen und unteilbar ausgeführt werden; definieren kritische Abschnitte ohne Locking, unterstützen lock-free Datenstrukturen.
- *Memory-Barriers (Fences)* – Anweisungen, die Speicheroperationen in einer bestimmten Reihenfolge erzwingen; wichtig zur Verhinderung von Reordering durch Compiler/CPU; erzielt korrekte Sichtbarkeit von Speicherupdates zwischen Threads/Prozessen.
- Typische Semantik: Acquire (erhält Sperre) / Release (freigibt Sperre); Memory-Ordering-Modelle (z. B. sequential consistency, acquire/release) definieren, wie Operationen sichtbar werden.

Aufgabe 4.

(a) Ablauf eines Interrupt-Handling-Zyklus aus Sicht eines Prozessors (Priorisierung, Kontextwechsel)

Lösung:

- Interrupt-Trigger: Peripherie oder Hardware löst Unterbrechung aus.
- Kontext-Sicherung: CPU speichert Kontext des aktuell laufenden Designs (Programmzustand, Register, Stack-Frame).
- Interrupt-Vektor: CPU springt zum passenden ISR über eine Interrupt-Vector-Tabelle.
- ISR-Verarbeitung: kurzer, zeitkritischer Code; ggf. Schedule-Entscheidung oder Einrichtung von Deferred Work (Bottom-Half, DPC).
- ggf. Kontextwechsel: Scheduler prüft, ob eine neuere Aufgabe Vorrang hat; bei Bedarf Wechsel des Tasks.
- Rückkehr: Enden des Interrupts, Restore des Kontexts, Rücksprung zur auszuführenden Anwendung.
- Priorisierung: Höhere Priorität Interrupts können - sofern unterstützt - verschachtelt behandelt werden; ISR selbst sollte so kurz wie möglich bleiben.

(b) Monolithisches Betriebssystem vs. Microkernel-Ansatz – Auswirkungen auf Treiberstrukturen und Sicherheit

Lösung:

- *Monolithischer Kernel*:
  - Treiber im Kernel-Modus; direkter Zugriff auf Kernel-Datenstrukturen;
  - Vorteil: hohe Leistung, geringer IPC-Overhead, direkte Hardware-Ansteuerung;
  - Nachteil: größere Fehlermöglichkeiten, Fehler in Treibern können Kernel destabilisieren.
- *Microkernel-Ansatz*:
  - Kernel minimal (Scheduler, IPC, Grundgerüste), Treiber und weitere Dienste laufen in Benutzermodus als separate Prozesse;
  - Vorteil: bessere Stabilität/Sicherheit (Fehler isoliert), einfache Fehlersicherung;
  - Nachteil: erhöhter IPC-Overhead, potenziell längere Kommunikationswege.

(c) Wie das Betriebssystem Ressourcen wie CPU-Zeit, Speicher und I/O-Bus hierarchisch verwaltet; Rolle von Scheduling-Policies

Lösung:

- *CPU-Zeit*:
  - Zuweisung von Zeitfenstern (Quantums) oder dynamische Priorisierung; Implementierung durch Schedulern (z. B. CFS, RR, PRIO-basierte Schedulers);
  - Ziel: faire, deterministische oder QoS-orientierte Verteilung der Rechenzeit.
- *Speicher*:
  - virtuelle Adressierung, Seitentranslationsmechanismen, Paging/Swapping; Memory-Management-Einheiten regeln Zugriff, Schutz und Freigabe;
  - Seiten- und Cache-Hierarchie beeinflusst Performance; TLB-Schutzmechanismen sichern Isolation.
- *I/O-Bus*:
  - Scheduling-Mechanismen auf E/A-Ebene (I/O-Queues, Elevator- bzw. LOOK-Algorithmen); Priorisierung/Fairness, DMA-Nutzung, Interrupt-Handling.
- *Rolle der Scheduling-Policies*:
  - Definieren die Regeln, nach denen Ressourcen zugeteilt werden (z. B. deterministische Lateness, durchschnittliche Reaktionszeit, Durchsatz);
  - Policies berücksichtigen QoS-Anforderungen, Prioritäten, Umgebungsfaktoren (Interaktivität vs. Throughput) und Behnad der Konkurrenz.

(d) Zentrale Sicherheitsaspekte für sichere Betriebssysteme; Speicherschutz, Prozess-Isolation und robuste Interrupt-Verarbeitung; Bedeutung für kritische Infrastrukturen

Lösung:

- *Speicherschutz*:
  - Schutz des Adressraums jeder Anwendung durch MMU/TTBR- bzw. Page-Table-Mechanismen; Verhinderung von Speicherzugriffen außerhalb legitimer Bereiche.
- *Prozess-Isolation*:
  - Separierte Adressräume, kontrollierte Interprozesskommunikation, Grenzen zwischen Benutzer- und Kernelmodus;
  - Fehler- und Angriffsisolation, Reduktion von Privilege-Escalation-Risiken.
- *Robuste Interrupt-Verarbeitung*:
  - Sichere Behandlung von Interrupts, minimale Belegung der CPU-Zeit im ISR, klare Trennung von zeitkritischer ISR-Arbeit und nachgelagerter Verarbeitung;
  - Robustheit gegen DoS-/Interrupt-basierten Angriffen; Schutzmechanismen gegen verlorene Interrupts oder Prioritätenumkehr.
- *Bedeutung für kritische Infrastrukturen*:
  - Hohe Verfügbarkeit, Determinismus, Fehlertoleranz, Nachweisbarkeit (Audit, Logging) und Schutz vor Spoofing/Manipulation von Kern-Komponenten.