Probeklausur

Systemprogrammierung

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

Aufgabe 1.

(a) Definieren Sie, was unter einem Prozess und unter einem Thread zu verstehen ist. Erläutern Sie den Begriff des Kontextwechsels.

(b) Beschreiben Sie zwei typische Scheduling-Strategien: First-Come-First-Serve und Round-Robin. Welche Vor- und Nachteile ergeben sich jeweils?

(c) Welche Synchronisationsprimitive gibt es in der Programmierung von Betriebssystemen, und wozu dienen sie? Erläutern Sie kurz Mutex, Semaphore und Condition Variable.

(d) Diskutieren Sie, warum preemptives Scheduling Vorteile bei der Verhinderung von Ressourcenverknappung haben kann, und welche Kosten dadurch entstehen.

Aufgabe 2.

(a) Speicherhierarchien und Caching: Erläutern Sie die Ebenen Cache, Hauptspeicher und Sekundärspeicher. Definieren Sie die Begriffe Cache-Hit und Cache-Miss.

(b) Geräteunabhängige Ein-/Ausgabe und Treiber: Beschreiben Sie den Unterschied zwischen generischer I/O-Schicht und treiber-spezifischer Implementierung; erklären Sie, wie Abstraktion die Portabilität erleichtert.

(c) Interrupt Handling: Beschreiben Sie den Unterschied zwischen Polling und interrupt-gesteuerter Verarbeitung. Welche Vor- und Nachteile ergeben sich in Bezug auf Reaktionszeit und CPU-Auslastung?

(d) Speicherverwaltung in modernen Systemen: Erklären Sie den Unterschied zwischen virtuellem Speicher und physischem Speicher; skizzieren Sie kurz das Prinzip der Seitentabelle.

Aufgabe 3.

(a) Beschreiben Sie drei Ansätze zur Synchronisation zwischen Threads im Kernel- oder Benutzermodus: Sperren, Barrieren, Events. Erläutern Sie kurz deren Einsatzfelder.

(b) Semaphoren: Unterscheiden Sie binäre und zählende Semaphoren; erläutern Sie, wie der Zähler verwendet wird, um konkurrierende Zugriffe zu regeln.

(c) Eine einfache Synchronisationsaufgabe: Skizzieren Sie, wie zwei Threads via Mutex einen geteilten Zähler sicher inkrementieren können. Beschreiben Sie den Ablauf, ohne Code zu liefern.

(d) Interprozesskommunikation: Nennen Sie zwei Mechanismen der Kommunikation zwischen Prozessen und erläutern Sie kurz deren Stärken und Schwächen.

Aufgabe 4.

(a) Sicherheit in Betriebssystemen: Welche grundlegenden Schutzmechanismen schützen vor Daten- und Identitätsdiebstahl? Geben Sie drei Beispiele.

(b) Ressourcenschonung: Diskutieren Sie Ansätze, wie effiziente Betriebssysteme den Ressourcenverbrauch senken und Energie sparen können.

(c) Treiber-Integrität und Systemvertrauen: Warum ist Treiber-Sicherheit entscheidend für Systemsicherheit?

(d) Nachhaltigkeit im Betriebssystem-Design: Welche Prinzipien fördern eine nachhaltige Nutzung von Rechenressourcen?

Lösungen

Lösung zu Aufgabe 1. (a)

Prozesse vs. Threads

- **Prozess:** Eine eigenständige Ausführungseinheit eines Programms mit eigenem Adressraum (Schutz- und Abgrenzung durch die MMU), eigener Ressourcenbelegung (Dateideskriptoren, Handles, Speicher, gegebene Systemressourcen) und eigener Ausführungskontext. Ein Prozess kann mehrere Threads enthalten.
- **Thread:** Die kleinste Ausführungseinheit innerhalb eines Prozesses. Threads teilen sich den Adressraum und Ressourcen des übergeordneten Prozesses, besitzen aber eigenen Ausführungskontext (z. B. Registerzustand, Programmzähler, Stackpointer).
- **Kontextwechsel:** Das Speichern des aktuellen Ausführungskontexts (Register, Program Counter, Stackpointer, ggf. Speicherschutzzustände, TLB- und Cache-Status) eines Threads/Prozesses und das Wiederherstellen eines anderen. Kontextwechsel können zwischen Threads desselben Prozesses (leicht) oder zwischen Prozessen (schwer) erfolgen und beinhalten oft Flushing/Invaliderung von Cache und TLB.

Lösung zu Aufgabe 1. (b)

Scheduling-Strategien

- **First-Come-First-Serve (FCFS):** Non-preemptives Scheduling-Verfahren, bei dem der zuerst eingereihte Prozess als Nächstes ausgeführt wird. Vorteile: einfach, deterministisch. Nachteile: long-Batch-Prozesse können andere blockieren; schlechte Interaktionsreaktionszeiten; durchschnittliche Wartezeit kann hoch sein.
- **Round-Robin (RR):** Preemptives Scheduling mit fester Zeitquantum (Zeitscheibe). Vorteile: Fairness, gute Reaktionszeit für interaktive Prozesse, einfache Implementierung. Nachteile: Wahl der Quantumsgröße beeinflusst Leistung stark; zu kleines Quantum führt zu vielen Kontextwechseln, zu großes Quantum ähnelt FCFS; Kontextwechsel-Overhead kann kostenintensiv sein.

Lösung zu Aufgabe 1. (c)

Synchronisationsprimitive

- **Mutex (Mutual Exclusion):** Sperre zum Schutz kritischer Abschnitte. Nur ein Thread kann zurzeit den geschützten Bereich betreten; verhindert Datenkorruption durch gleichzeitigen Zugriff.
- **Semaphore:** Zählerbasierte Synchronisation mit freigegebenen Ressourcen. Binäre Semaphoren (0/1) ähneln Mutexen; zählende Semaphoren regeln zulässige Zugriffe auf endliche Ressourcen (z. B. Instanzen einer Ressource).
- **Condition Variable:** Warte- und Signalisierungsmechanismus, um Threads zu benachrichtigen, wenn sich eine Bedingung ändert. Typisch in Kombination mit einem Mutex verwendet, um Wartezustände zu handhaben.

Lösung zu Aufgabe 1. (d)

Preemptives Scheduling und Kosten

- **Vorteile:** Verhindert einseitiges Blockieren von Ressourcen durch lange laufende Prozesse; erhöht Reaktionsfähigkeit, insbesondere für interaktive Anwendungen; erleichtert Fairness und Fortschritt unter Mehrprozessorsystemen.
- **Kosten:** Häufige Kontextwechsel verursachen Overhead (Speicher- und Cache-Verlust, TLB-Flush), erhöhten Energieverbrauch, mögliche Cache- und Speicher-Verdrängung (Cache Thrashing) und komplexere Scheduling-Politiken (z. B. Prioritäten, Verkleinerung von Pufferungen).

Lösung zu Aufgabe 2. (a)

Speicherhierarchien und Caching

- **Cache:** Schneller, begrenzter schneller Zwischenspeicher zwischen CPU und Hauptspeicher. Enthält aktuell verwendete Daten/Instruktionen. **Hit** bedeutet erfolgreicher Zugriff aus dem Cache; **Miss** bedeutet, dass die Daten nicht im Cache liegen und aus dem Hauptspeicher geladen werden müssen.
- **Hauptspeicher (RAM):** Größer, langsamer als Cache; dient als primärer Arbeitsbereich für Programme.
- **Sekundärspeicher:** Langsam, aber voluminös (Festplatte, SSD). Wird für persistente Daten genutzt und kann durch Paging/Swapping mit dem Hauptspeicher interagieren.

Lösung zu Aufgabe 2. (b)

Geräteunabhängige I/O und Treiber

- **Generische I/O-Schicht:** Bietet abstrakte, plattformunabhängige Interfaces (z. B. Dateisystem-API, generische Treiber-Schnittstelle, Virtuelle Dateisysteme). Ermöglicht Portabilität über verschiedene Hardware-Plattformen hinweg.
- **Treiber-spezifische Implementierung:** Nutzt hardwarenahe Details, adressiert Register, Interrupts und besondere Eigenschaften des jeweiligen Geräts. Optimiert Leistung, aber reduziert Portabilität.
- **Abstraktion** erleichtert Portabilität, Wiederverwendung und Wartbarkeit, während Adapter-Treiber je nach Zielhardware implementiert werden.

Lösung zu Aufgabe 2. (c)

Interrupt Handling

- **Polling:** CPU fragt periodisch den Status eines Geräts ab. Vorteile: einfache Umsetzung, geringe Interrupt-Overheads bei wenigen Geräten. Nachteile: potenziell hohe CPU-Auslastung, schlechtere Reaktionszeit bei seltenen Ereignissen.
- **Interrupt-gesteuerte Verarbeitung:** Gerät signalisert dem Kernel, wenn ein Ereignis vorliegt. Vorteile: effiziente CPU-Nutzung, bessere Reaktionszeiten bei asynchronen Ereignissen. Nachteile: Interrupt-Handling-Overhead, Complexity bei Interrupt-Storms und Synchronisation.

Lösung zu Aufgabe 2. (d)

Speicherverwaltung in modernen Systemen

- **Virtueller Speicher vs. physischer Speicher:** Virtueller Speicher bietet jedem Prozess eine isolierte Sicht auf Speicher, was Kopplungen zwischen Prozessen reduziert und speicherbasierte Schutzmechanismen ermöglicht. Physischer Speicher ist das tatsächliche RAM.

- **Seitentabelle (Page Table):** Mapping von virtuellen Seiten auf physische Seiten. Der Kernel verwaltet diese Tabellen, um Adressübersetzungen zur Laufzeit durchzuführen. Typischer Aufbau umfasst PTEs (Page Table Entries) mit Bits für Präsenz, Berechtigungen, Caching-Flags, etc.
- **TLB (Translation Lookaside Buffer):** Schneller Cache für häufige Adressübersetzungen; TLB-Hits vermeiden den Registerzugriff auf die Seitentabelle, TLB-Misses führen zur Nachlatisierung der Seitentabelle.

Lösung zu Aufgabe 3. (a)

Ansätze zur Synchronisation zwischen Threads

- **Sperren (Locks):** Schützen kritische Abschnitte durch gegenseitige Exklusion. Verhindern gleichzeitige Modifikationen gemeinsamer Daten.
- **Barrieren (Barriers):** Synchronisieren mehrere Threads an einer bestimmten Ausführungspunkt; alle Threads müssen den Barrier-Punkt erreichen, bevor fortgefahren wird.
- **Events (Signalisierung/Benachrichtigung):** Threads warten auf bestimmte Ereignisse oder Zustandsänderungen und werden durch Signale geweckt oder fortgesetzt.

Lösung zu Aufgabe 3. (b)

Semaphoren: Binäre vs. Zählende

- **Binäre Semaphoren:** Zähler ist 0 oder 1; dienen ähnlich wie Mutex zum Schutz eines kritischen Abschnitts.
- **Zählende Semaphoren:** Zähler gibt die Anzahl der verfügbaren Ressourcen oder Permit-Zugriffe an; Threads blockieren, bis der Zähler positiv ist, danach wird der Zähler dekrementiert.

Lösung zu Aufgabe 3. (c)

Ablauf: Zwei Threads inkrementieren sicher einen geteilten Zähler via Mutex

- Beide Threads versuchen, den Mutex zu erwerben, bevor sie den geteilten Zähler lesen.
- Nach dem Erwerb inkrementieren sie den Zähler kritisch, und schreiben das neue Resultat zurück.
- Vor dem Verlassen des kritischen Abschnitts wird der Mutex freigegeben, sodass andere Threads den Abschnitt betreten können.
- Durch diese Sequenz bleibt der Zugriff synchronisiert, was Datenkorruption durch gleichzeitige Inkrementation verhindert.

Lösung zu Aufgabe 3. (d)

Interprozesskommunikation (IPC)

- **Pipes:** Unidirektionale oder bidirektionale Byte-Streams zwischen Prozessen. Vorteile: einfache, gute Performance für Streaming-Daten; Nachteil: begrenzte Puffergröße, ungerichtete Kommunikation (je nach Pipe).
- **Gemeinsamer Speicher (Shared Memory) inklusive Synchronisation:** Schnelle, direkte Datenaushäufung ohne Copy-Kosten; Nachteil: Synchronisation erfordert zusätzliche Mechanismen (z. B. Semaphoren/Mutex), um Konsistenz sicherzustellen; Abhängigkeit von sorgfältiger Garbe.

Lösung zu Aufgabe 4. (a)

Sicherheit in Betriebssystemen

- **Schutzmechanismen:** Isolation (Prozess-Adressraum, Privilege Levels), Zugriffskontrollen, und Ressourcenbegrenzung.
- **Beispiele:**
  1. **Speicherschutz:** MMU sorgt dafür, dass Prozesse nur auf ihren eigenen Adressraum zugreifen.
  2. **User-/Kernel-Modus-Trennung:** Unterscheidung zwischen privilegierten und unprivilegierten Operationen.
  3. **Zugriffsrechte/ACLs oder Capabilities:** Festlegung, welche Prozesse welche Ressourcen verwenden dürfen.

Lösung zu Aufgabe 4. (b)

Ressourcenschonung und Energie sparen

- **Energiesparende Techniken:** DVFS (Dynamic Voltage/Frequency Scaling), Sleep/Standby- und Hibernate-Modi, adaptive Leistungs-/Kühlungsmanagement.
- **Sonstige Maßnahmen:** Optimierte Scheduling-Politiken, Speicher-Dunkel-Modi, verbesserte Interrupt-Verarbeitung, effiziente Treiber- und Kernel-Designs.

Lösung zu Aufgabe 4. (c)

Treiber-Integrität und Systemvertrauen

- Treiber laufen oft mit hohen Privilegien. Unsigned oder kompromittierte Treiber können Kernel-Sicherheitslücken ausnutzen.
- **Maßnahmen:** Signierung von Treibern, geprüfte Patch- und Update-Mechanismen, regelmäßige Sicherheitsprüfungen, Beschränkung von Privilegien, Verifikation der Integrität von Treibern.
- Ergebnis: Hohe Systemstabilität, geringeres Risiko von Kernel-Compromises.

Lösung zu Aufgabe 4. (d)

Nachhaltigkeit im Betriebssystem-Design

- **Prinzipien:** Energieeffizienz durch ressourcenbewusste Planung, Anpassung an Lastprofile, langsame/rasche Skalierung von Ressourcen (z. B. Container-Orchestrierung, Virtualisierung).
- **Weitere Prinzipien:** Modularität, Offenheit, Wartbarkeit, Transparente Ressourcenabrechnung, Unterstützung moderner Sleep-States und Thermal-Management.