Fragenkatalog – Informatik IV – Rechner- und Betriebssysteme (2005-07)

1. **Welche Arten von Instruktionen sind idR auf der konventionellen Maschinenebene verfügbar?**
   - arithmetisch – logische Befehle
   - Instruktionen zum Rechnen (add, sub,..)
   - Datenverschiebeoperationen (mov)
   - Vergleiche (compare)
   - (bedingte) Sprünge, Aufrufe (call), return's

2. **Was sind Register?**
   - extrem schnelle, flüchtige, sehr teure Speicher mit geringer Speicherdichte
   - werden mit Prozessortakt betrieben

3. **Welche speziellen Register sind bei den meisten CPU-Typen zu finden?**
   - IP = instruction-pionter → zeigt auf nächste Instruktion
   - PC = program-counter
   - SP = stack pointer → Zeigt auf oberste Zelle des Stacks
   - BP = base pointer
   - PSW = program status word
   - SR = status register

4. **Warum gibt es privilegierte Instruktionen? Nenne mind. 2 Bsp.!**
   um hardwareseitigen Schutz des BS-Kerns vor (böswilligen) Anwedungsprogrammen zu ermöglichen → BS-spezifische Aufrufe, die nicht im Usermodus möglich sind
   - Interrupts sperren / freigeben
   - Hardware-seitige Ein- / Ausgabe
   - Zugriff auf bestimmte Speicherbereiche

5. **Wie viele Instruktionen pro Sekunde kann ein typischer RISC-Prozessor bei einem Takt von 1 GHz theoretisch mindestens ausführen? Ist die theoretische Annahme realistisch?**
   - 1 Milliarde Instruktionen möglich, aber nicht realistisch, da I/O wesentlich langsamer ist. Instr. können also gar nicht schnell genug „herbeigeschafft" werden ; zusätzlich: Unterbrechungen durch Interrupts während der Laufzeit

6. **Wozu dient die Speicherhierarchie in Rechnersystemen?**
   - Abgrenzung der verschiedenen Speichertypen → der ideale Speicher ist extrem schnell, extrem groß und billig → praktisch allerdings unrealistisch
   - schnelle Speicher sind idR in der Kapazität klein und teuer (z.B. Register)
   - große Speicher sind idR langsam und billig (z.B. Festplatten)

7. **Warum unterstützen die meisten CPU-Typen einen Benutzer- und einen Systemmodus?**
   Weil Anwendungsprogramme (Benutzermodus) den Computer nur benutzen, nicht aber steuern und kontrollieren dürfen / sollen. → Schutz des BS

8. **Erkläre die Funktionsweise der CALL- & der RET-Instruktionen bei Prozess. der 80x86-Fam.!**
   - CALL: Aufruf eines Unterprogramms → Sprung an die bestimmte Adresse, eigener Stack muss eingerichtet werden (Veränderung des SP), Rücksprungandresse merken
   - RET: Rückkehr an die gemerkte Rücksprungadresse des Aufrufers

9. **Wozu dienen test-and-set, compare-and-exchange bzw. vergleichbare Instruktionen?**
   - zur atomaren (unteilbaren) Ausführung von „testen und setzen" bzw. „vergleichen und ersetzen"
     → Verwendung bei der Auswertung von Wartebedingungen (SpinLocks)

10. **Was ist in der sog. Prozedurkonvention geregelt?**
    - es wird geregelt, was beim Aufruf einer Prozedur gemacht wird:
    → welche Register müssen vom Aufrufer gerettet werden (flüchtig)
    → welche Register müssen vom Aufgerufenen gerettet werden („nicht-flüchtig")
    → wie erfolgt die Parameterübergabe (über Stack, über Register, beides)
    → Lage der Parameter bzgl. Base-Pointer

11. **Was versteht man unter flüchtigen Registern?**
    - Register, bei denen nicht sichergestellt ist, dass der Inhalt erhalten bleibt, wenn eine andere Prozedur aufgerufen wird
    - Sicherung durch den Compiler (→ Aufrufer oder aufgerufener?)

12. **Was versteht man unter nicht flüchtigen Registern?**
    - Register, bei denen sichergestellt ist, dass ihr Inhalt erhalten bleibt, wenn eine andere Prozedur aufgerufen wird

13. **Wozu dient die lokale Basis?**
    - ein Pointer innerhalb des Stack (BP = BasePointer), der als relative Adressierungsbasis dient
    - z.B.: Parameter negativ zur lokalen Basis ; Variablen positiv zur lokalen Basis

14. **Wozu dienen Unterbrechungen?**
    - zum Anzeigen von: Ausnahmesituationen
    - Abwicklung dringlicher Angelegenheiten (in Bezug auf Geräte, z.B. I/O)
    - Clock → Taktgeber zum Prozesswechsel

15. **Erkläre den wesentlichen Unterschied zwischen Traps und Interrupts!**
    - Traps sind synchron, sie treten zur Erkennung und Behandlung von Programmfehlern oder BS-Diensten auf
    - Interrupts sind asynchron und werden von Geräten ausgelöst (I/O, Clock) , können jederzeit ausgelöst werden (haben höhere Priorität als Traps)

16. **Wann löst die CPU einen Trap aus?**
    - bei arithmetischen Fehlern (z.B. div. by zero)
    - bei Page Faults (Zugriff auf gesperrten Speicher)
    - Ausführung privilegierter, illegaler Instruktionen
    - Breakpoint von Debuggern
    - spez. Trap-Instruktionen für bestimmte Dienste vom OS

17. **Erkläre den Ablauf einer Unterbrechungsbehandlung bei einer typischen CISC-CPU!**
    - Gerät löst Unterbrechung über Signalleitung aus
    - CPU wechselt in den Systemmodus (Wechsel in Systemadressraum)
    - Retten von PC und SR auf KernelStack
    - Bestätigten des Interrupts
    - ermitteln der Handleradresse anhand der Unterbrechungsnummer mit Hilfe von Vektortabelle
    - Aufruf des Interruptshandlers

18. **Welche Aktivitäten muss ein Interrupt-Handler wenigstens ausführen, damit ein unterbrochenes Programm wieder fehlerlos fortgeführt werden kann?**
    - Retten aller flüchtigen Register
    - sowie aller Register, die die initiale Kopplungsroutine verwendet
    - Bestätigen des Interrupts an der CPU

19. **Warum ist es sinnvoll, eine Hochsprachenanbindung für Interrupt-Handler zu schaffen?**
    - um neben der Geräteabhängigkeit nicht auch noch eine CPU-Abhängigkeit zu schaffen (wie es durch Assemblercode der Fall wäre)

20. **Wozu dienen Busse in Rechnersystemen?**
    - zur Kommunikation verschiedener Geräte bzw. Speicher untereinander
    - Datenaustausch zw. Geräten

21. **Warum existiert in den meisten Rechnersystemen eine Hierarchie von Bussystemen?**
    - unterschiedliche Bedürfnisse, unterschiedliche Geschwindigkeiten → sehr langsame und extrem schnelle Bussysteme
    - physikalische Beschränkungen: Steuerleitungen schneller Bussysteme können idR nicht sonderlich lang sein, außerdem können auch nicht beliebig viele Geräte angeschlossen sein
    - manche Busse müssen meist nur Signale übertragen, andere große Datenmengen

22. **Was ist in einem Busprotokoll geregelt?**
    - die genauen Parameter des Ablaufs der Kommunikation, z.B. Gerät schickt Anfrage an den Arbiter zur Reservierung des Bus', Gerät legt / liest Daten auf den / vom Bus etc.

23. **Was ist der Unterschied zw. Adress- und Datenbus?**
    - Datenbus: zur Datenübertragung (z.B. 8x256 Bit)
    - Adressbus: zur Bekanntgabe von Lese- bzw. Schreibadressen (z.B. 8 Bit, 16 Bit, 40 Bit)

24. **Was versteht man unter dem „Multiplexing" von Adress – und Datenleitungen?**
    - Adressleitungen sind in den Datenleitungen enthalten + eine Steuerleitung, die festlegt, ob gerade Daten oder Adressen auf dem Bus sind

25. **Was versteht man unter DMA-Geräten?**
    - DMA = direct memory acess
    - Gerät kann direkt auf den Hauptspeicher zugreifen, Umweg über den Prozessor entfällt somit
    - benötigt einen DMA-Controller

26. **Erkläre den Unterschied bzw. den Zusammenhang zwischen einem Prozess und einem Programm!**
    - Programm: feste Abfolge von Instruktionen, Festlegung zur Compilezeit
    - Prozess: Instruktionen werden dynamisch zur Laufzeit festgelegt

27. **Was ist der Unterschied zw. schwer- und leichtgewichtigen Prozessen?**
    - schwergewichtige Prozesse: haben eigener Adressraum → beim Prozesswechsel muss ein Adressraumwechsel stattfinden
    - leichtgewichtige Prozesse teilen sich 1 Adressraum

28. **Skizziere den logischen Aufbau des Adressraumes eines schwergewichtigen Prozesses!**
    - Userstack: von low nach high: Programmcode, (initiale) Daten, Heap (wächst nach high) Stack (wächst von high (oberste Adresse) nach low)
    - Kernelstack: Stack (wächst von high nach low) für Prozesskontrollblock

29. **Erkläre den Unterschied zw. koorperativen und präemptiven Prozessen!**
    - koorprativ: synchron; jeder Prozess muss seinen Nachfolger kennen, Umschaltung erfolgt freiwillig (durch yield() / resume())
    - präemptiv: asynchron; Prozess ist frei unterbrechbar, Umschaltung erfolgt unfreiwillig von „außen" (z.B. durch Clock → Interrut; Zeitscheibenvergabe) → Scheduler muss vorhanden sein, der alle Prozesse kennt und verwaltet

30. **Wozu dienen leichtgewichtige Prozesse (Threads)?**
    - laufen innerhalb eines schwergewichtigen Prozesses ab (mehrere leichtgew. pro 1 schwergew.)
    - haben keinen eigenen Adressraum, sondern nur einen Stack
    - Zugriff auf die globalen Variablen des gemeinsamen Prozesses
    - aufwendige Adressraumwechsel innerhalb einer Applikation werden so vermieden

31. **Warum hat jeder Thread seinen eigenen Stack?**
    - für lokale Variablen und Daten des Threads
    - sein Gedächtnis → Statusinformationen des Threads

32. **Du sollst einen effekt. Webserver entwerfen. Können Dir Threads dabei helfen? Wenn ja,wie?**
    - **ja!** Benutzeranfragen werden als Threads verarbeitet → alle Threads haben Zugriff auf gemeinsamen Speicher → parallele Abarbeitung der Anfragen möglich → Benutzeranfragen in einer Liste speichern, die von einem extra „VerwaltungsThread" verwaltet werden. Dieser wartet auf Anfragen, wenn welche da sind, trägt er sie in eine Liste ein
      „Arbeiterthread" (mehrere Threads → Threadpool): Schaut nach, ob Anfragen in der Liste sind. Wenn ja, dann bearbeitet er diesen (Auftrag annehmen, Daten holen, Daten verschicken)
      Nur begrenzte Anzahl an „Arbeiterthreads" um Datendurchsatz nicht zu blockieren

33. Du sollst einen Window-Manager entwerfen. Können dir Threads dabei helfen? Wenn ja, wie?
- Nein!

34. **Welche Vor- bzw. Nachteile haben sogen. Kernel-Level-Threads?**
+ BS-Kern kennt alle Threads
+ Erkennung von Stacküberläufen (da Regelung direkt von OS)
+ keine Blockaden
- Notwendigkeit von Systemaufrufen, langsame Erzeugung und Thread-Wechsel

35. **Welche Vor- bzw. Nachteile haben sog. User-Level-Threads?**
+ schnelle Erzeugung und Thread-Wechsel,da kein BS-Aufruf nötig (→ kein Wechsel in BS-Modus) (BS-Kern kennt Threads nicht)
- blockierende Systemaufrufe (→ gesamte schwergew. Prozess kann blockieren → Nebenläufigkeit kann somit verloren gehen)
- Gefahr von Stacküberläufen, da keine Verwaltung von BS

36. **Was passiert, wenn ein User-Level-Thread einen blockierenden Systemaufruf ausführt?**
- damit werden alle anderen Threads des schwergewichtigen Prozesses auch blockiert
- Vorteil der Nebenläufigkeit geht somit verloren

37. **Was passiert, wenn ein Kernel-Level-Thread einen blockierenden Systemaufruf ausführt?**
- nur der aufrufende Thread wird blockiert

38. **Wozu dient der Scheduler?**
- kennt und verwaltet alle lauffähigen Prozesse in einer Readyliste
- Zuteilung des Prozessors an die Prozesse
- Einleitung des Prozesswechsels
- Stellt Methoden zum Hinzufügen und Entfernen von Prozessen auf / aus der Readyliste
- sollte stets fair sein (z.B. durch Verteilung von Zeitscheiben)

39. **Warum werden in interaktiven Systemen ein-/ausgabeintensive Prozesse bevorzugt?**
- beim Warten auf Benutzereingaben hat das System nichts zu tun
- ständige Ein- und Ausgaben belasten die CPU wenig
- CPU wird beim Warten auf I/O schnell abgegeben → System wirkt schnell

40. **Was unterscheidet einen I/O-intensiven Prozess von einem CPU-intensiven?**
- I/O-intensiv: sind in ihrer Verarbeitungsgeschwindigkeit von Geräten / Benutzer abhängig; Belasten die CPU sehr wenig
- CPU-intensiv: sind in ihrer Verarbeitungsgeschwindigkeit hauptsächlich von CPU / RAM abhäng.

41. **Was bedeutet es, wenn ein Prozess als „CPU bound" bezeichnet wird?**
- sehr CPU-lastig → beansprucht die CPU stark und zumeist andauernd mit geringen Unterbrech.

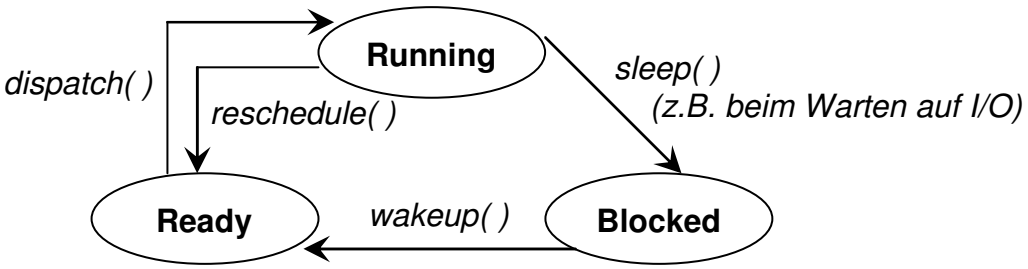
42. **Was bedeutet es, wenn ein Prozess als „I/O-bound" bezeichnet wird?**
- sehr Ein- / Ausgabelastiger Prozess, wartet idR viel auf Beendigung von Geräteoperationen / Benutzereingaben → sehr geringe CPU-Last

43. **Wozu dient die Multiprogrammierung?**
- um die Zeiten der CPU-mässigen Inaktivität eines Programms (z.B. bei I/O) werden mit der Ausführung eines anderen Programms ausgenutzt → optimale CPU-Auslastung, Multitasking

44. **Was ist bei allen Scheduling-Strategien zu beachten?**
- sollten fair sein (z.B. durch Vergabe von Zeitscheiben ; FIFO ; ..)
- jeder Prozess muss mal drankommen (keiner darf verhungern)

45. **Mit welcher Problematik müssen wir rechnen, wenn wir bestimmte Prozesse bevorzugen?**
- dass andere Prozesse verhungern, also nie die CPU bekommen → Starvation
→ unfair

46. **Welche logischen Zustände durchläuft ein lebender Prozess (vernachlässige Start- und Endzustände)? Zeichne den Zustandsgraphen und erläutere die Zustandsübergänge!**



47. **Erkläre den Unterschied zw. Prozeduren und Coroutinen!**
Prozeduren: müssen immer vollständig abgearbeitet werden. Kontrollwechsel nur hierarchisch möglich ; Coroutinen: können an beliebiger Stelle die Kontrolle abgeben

48. **Was versteht man unter dem Prozessorzustand eines Prozesses?**
- Adressraum, Stackpointer, Prozessorregister

49. **Welcher Teil des Prozessorzustandes muss bei einem Coroutinenwechsel nur gerettet bzw. wieder aufgesetzt werden? Warum?**
- Stackpointer (→ im Stack stehen dann die geretteten Register, die man wieder herstellen muss)
- Rücksprungadresse

50. **Wodurch ist der Zustand eines Prozesses vollständig beschrieben?**
- Prozessor (+ dessen Register) (bei Multiprozessorsystemen die CPU, auf dem der Prozess läuft)
- Adressraum
- Stack

51. **Erkläre den Zusammenhang zw. Prozessen und Coroutinen!**
- Coroutine = Aktivitätsträger mit eigenem Kontrollfluss, daher Vorstufe eines Prozesses
- Prozess = werden durch Prozessverwaltung gemanagt, können implizit aktiviert werden (Coroutinen nur explizit), schwergewichtige und leichtgewichtige Prozesse sind möglich

52. **Wozu dient ein Prozesskontrollblock?**
- enthält alle Verwaltungsinformationen für den Prozess:
Prozess-ID, Prozessorregister, Zustand (z.B.: Running, Ready, Blocked), Adressraum, Betriebsmittelbelegung

53. **Was versteht man unter einem kritischen Abschnitt?**
- ein Abschnitt im Quelltext, auf den überlappend zugegriffen werden kann, der eine gemeinsame Datenstruktur verändert und sofern er nicht geschützt wird zu kaum reproduzierbaren Fehlern führen kann → Nebenläufigkeit: verändernder Zugriff auf die gemeinsame Datenstruktur darf nicht unterbrochen werden

54. **Darf ein Prozess innerhalb eines kritischen Abschnitts (z.B. aufgrund einer Wartebedingung) blockieren?**
Nein! Von wem soll er denn aufgeweckt werden, wenn niemand anderes in den krit. Abschnitt eintreten kann, der „meine“ Wartebedingung erfüllt!!
(Ja, wenn die Wartebedingung erfüllt werden kann, ohne dass ein anderer Prozess in den krit. Abschnitt eintreten muss)

55. **Wie kann man kritische Abschnitte schützen? Nenne mindestens 3 Möglichkeiten!**
Int-Locks, Spin-Locks, Semaphoren, Monitore

56. **Kann man im Benutzermodus der CPU kritische Abschnitte durch das Sperren von Interrupts schützen?**
Nein! Das Ausschalten der Interrupts ist eine privilegierte Operation, das Anschalten ebenfalls → darf nur im Betriebssystemmodus ausgeführt werden → sonst könnte jedes Anwenderprogramm die Interrupts sperren und somit das System aufhängen

57. **Warum sollte man das Sperren von Interrupts weitgehend vermeiden?**
- weil speziell durch längeres & häufiges Ausschalten das System sehr schnell reaktionsträge wird
- es können sogar Interrupts verloren gehen

58. **Wo liegen die Vor- und Nachteile von Spin-Locks?**
+ bei Multiprozessorsystemen einfach und unkompliziert möglich → leicht zu implementieren
- aktives Warten („busy-Waiting“) → verbraucht viel CPU-Zeit

59. **Ist es in Einprozessorsystemen sinnvoll, an gesperrten Spinlocks sofort die CPU abzugeben?**
Ja! Sonst würde man bis zum Prozesswechsel aktiv warten, solange kann niemand „meine“ Wartebedingung ändern

60. **Ist es in Multiprozessorsystemen sinnvoll, an gesperrten Spinlocks sofort die CPU abzugeben?**
Nein! Da ein anderer Prozess auf einer anderen CPU laufen kann und somit meine Bedingung evtl. in wenigen Takten erfüllt werden könnte, während ich auf meiner CPU aktiv warte.
- Möglichkeit: nur kurz warten, dann doch die CPU abgeben

61. **Was ist bei Spinlocks für Mehrprozessorsystemen zu beachten?**
- Fairness ist nicht unbedingt zugesichert → Starvation möglich
- Verpassen der Erfüllung meiner Wartebedingung ist möglich
- Bus kann blockieren (aktives Warten, ständiges Abfragen von Zuständen)

62. **Warum ist das Blockieren aufgrund von Wartebedingungen nicht mit dem Blockieren an kritischen Abschnitten vergleichbar?**
- das Erfüllen von Wartebedingungen kann sehr lange dauern
- kritische Abschnitte hingegen sind sehr kurz

63. **Erkläre die Funktionsweise einer Semaphore!**
- dient zur Verwaltung von Betriebsmitteln
- Initialisierung mit einer Zählvariablen (= Anzahl der Betriebsmittel, z.B. 24 bei 24 Speicherzellen) Initialisierung mit 1 entspricht einer binären Semaphore
- Verwaltung von wartenden Prozessen (warten auf Betriebsmittel) in einer Liste
- 2 Methoden: signal(), wait();
- signal(): Falls keine Prozesse schlafen, inkrementiere den Zähler, sonst den nächsten schlafenden Prozess aufwecken
- wait(): Falls Zähler 0 ist, diesen Prozess in die Liste der schlafenden Prozesse einfügen, sonst Zähler dekrementieren

64. **Was sind die beiden Haupteinsatzgebiete von Semaphoren?**
- Betriebsmittelverwaltung (Zähler gibt Anzahl der BSmittel an)
- Schutz kritischer Abschnitte (binäre Semaphore)

65. **Was ist ein Monitor?**
- für alle Operationen des Monitors gilt: wechselseitiger Ausschluss ist garantiert
- zum Eintritt in den Monitor werden die Prozesse in eine Warteschlange eingereiht
- es kann sich immer nur 1 Prozess im Monitor befinden
- klare Eintritte ein bzw. Austritte aus dem Monitor → Systemschnittstelle

66. **Warum werden BS-Kerne häufig als Monitor implementiert?**
- aus praktischen Gründen: wechselseitiger Ausschluss für BS-Funktionen ist idR empfehlenswert
- Monitor stellt dies sicher und bietet klare Schnittstellen + Verwaltung in Warteliste

67. **Wie viele Prozesse können sich gleichzeitig im Kernel-Monitor befinden?**
genau 1

68. **Wo ist der Unterschied zw. Inter- und Intraprozesssynchronisation?**
Int**er**prozesssynchronisation: Koordination von Prozessen
Int**ra**prozesssynchronisation: Koordination innerhalb eines Prozesses (zur Vermeidung von Deadlocks muss man hier nichtblockierende Synchronisation verwenden)

69. **Warum braucht man nichtblockierende Synchronisationen?**
- sonst kann es zu Deadlocks kommen → Systemstillstand

70. **Wie werden in BS mit Adressraumtrennung Funktionen des BS-Kernes aufgerufen?**
- durch die Benutzung von Traps

71. **Warum haben Prozesse in BS mit Adressraumtrennung einen eigenen Kernel-Stack?**
- zur Sicherung ihres Zustandes bei Auftreten eines Interrupts
→ Interruptbehandlung

72. **Welche besondere Problematik ist mit der Kernelsynchronisation verbunden?**
- da hier nur nichtblockierende Synchronisation in Frage kommt, ergibt sich ein Betriebsmittel-Problem: Speicherfähigkeit ist begrenzt, man kann sich nicht beliebig viele Epiloge merken

73. **Wozu dient die Schleusensynchronisation?**
- regelt den Eintritt in den synchronen Betriebsystemkern
- falls Schleuse gesperrt → einfügen des Epilogs in Queue → Verwaltung v. Epilog-Warteschlange

74. **Welche Rolle spielen Interrupt-Prologe und Epiloge?**
- Interrupt tritt auf → Prozess wird unterbrochen → Kernel im Ausnahmemodus → asynchroner Prolog wird abgearbeitet (kann jeder Zeit auftreten, wird mit gesperrten Interrupts abgearbeitet, ist idR sehr kurz!) → später, im normalen Kernelmodus wird dann auf Anforderung des Prologs der Epilog abgearbeitet
→ Prolog: asynchron, Bestätigung des Interrupts, evtl. Daten zwischenspeichern, Entscheidung ob Epilog nötig
→ Epilog: synchron, Nachbehandlung, kann länger sein, (Ausführung im Kernel-Monitor?)

75. **Welche Aktivitäten sind einem Prolog, welche einem Epilog zuzuordnen?** → siehe 74

76. **Welche Aktivitäten darf ein Prolog auf keinen Fall ausführen?**
Operationen des Kernel-Monitors

Warum nicht?
→ es könnte ein Epilog im Kernel-Monitor sein. Wird durch einen Interrupt ein neuer Prolog aufgerufen, der in den (besetzten) Kernel-Monitor will, so steht das System, da der Prolog mit deaktivierten Interrupts durchläuft.

77. **Darf ein Epilog blockierende Operationen aufrufen?**
- wenn er das täte, könnte er den Kernel-Monitor nicht mehr verlassen

78. **Darf ein Prolog blockierende Operationen aufrufen?**
Nein! Ein Prolog wird asynchron abgearbeitet, mit deaktivierten Interrupts. Wenn er blockiert, gibt es einen Deadlock.

79. **können Prologe durch andere Prolog unterbrochen werden?**
- es können nur Interrupts höherer Priorität auftreten, sofern also ein solcher auftritt, kann dieser durchaus einen Prolog aufrufen

80. **Können Epiloge durch andere Epiloge unterbrochen werden?**
Nein! Epiloge werden in einer Warteschlange verwaltet. Somit kann immer nur 1 Epilog abgearbeitet werden, alle weiteren werden in der Schlange eingereiht. Außerdem kann der Epilog im Kernel-Monitor ausgeführt werden

81. **Können Epiloge durch Prologe unterbrochen werden?**
Ja! Prologe treten asynchron auf, bei einem Interrupt, und dieser kann jederzeit eintreten

82. **Dürfen Interrupt-Handler blockieren?**
Nein!

83. **Darf ein Rescheduling stattfinden, wenn geschachtelte Interrupts noch nicht vollständig abgearbeitet wurden?**
Nein! Erst müssen alle Prologe abgearbeitet werden! Sonst könnten Interrupts verloren gehen oder erst viel später abgearbeitet werden

84. **Eine serielle Schnittstelle wird mit 19200Bit/s betrieben. Wie hoch ist die Zeichenrate pro Sekunde? Skizziere den Eingabeteil eines Treibers für dieses Gerät und begründe Deinen Entwurf. Gehe von einer Schleusensynchronisation mir Pro- und Epilogen aus!**
19200 / 8 = 2400 Zeichen pro Sekunde (bei 8 Bit pro Zeichen)

85. **Darf ein Epilog immer sofort ausgeführt werden, wenn der Kernel-Monitor frei ist?**
Nein! Erst muss „nachgeschaut" werden, ob noch Prologe abzuarbeiten sind. Wenn alle abgearbeitet wurden, muss der Kernel-Monitor frei sein. Erst jetzt darf der Epilog abgearb. werden.

86. **Welche Möglichkeiten haben Gerätetreiber, den Zugriff auf Daten zu schützen, auf die sowohl im synchronen als auch im asynchronen Teil eines Treibers zugegriffen werden muss? Gib 2 Möglichkeiten an und diskutiere die Vor- und Nachteile!**

87. **Wozu dienen Bedingungsvariablen?**
- dienen zum Warten auf den Eintritt einer Bedingung (→ SpinLocks)
→ wechselseitiger Ausschluss

88. **Welche Aktionen müssen durchgeführt werden, wenn innerhalb eines Monitors blockierend auf den Eintritt einer Bedingung gewartet wird?**
- es muss ein wait() auf der Bedingungsvariablen ausgeführt werden, damit sich der Prozess schlafen legt und verlässt den Monitor
- mit signal() sollte er dann wieder von außen aufgeweckt werden können

89. **Was ist ein Deadlock?**
- wenn in einer Menge von Prozessen ein jeder auf ein Ereignis wartet, das nur von einem anderen ausgelöst werden kann

90. **Wozu dienen Betriebsmittelgraphen?**
- Darstellung der Zuordnungen von Prozessen zu Betriebsmitteln

91. **Welche 4 Bedingungen müssen erfüllt sein, damit ein Deadlock überhaupt auftreten kann?**
- Coffman-Regeln:
  - Betriebsmittel sind nur exklusiv nutzbar
    → *„mutual exclusion condition"*
  - Betriebsmittel können nicht präemptiv entzogen werden
    → *„no preemption condition"*
  - Prozesse die diese Betriebsmittel belegen, können auf Zuteilung weiterer Mittel warten
    → *„hold and wait condition"*
  - In einer zyklischen Kette von Prozessen, hat ein jeder wenigstens ein Betriebsmittel, auf das der jeweils nächste wartet
    → *„circular wait condition"*

92. **Was versteht man unter Starvation und unter welchen Bedingungen kann es auftreten?**
    - Verhungerung von Prozessen, die quasi bei der Betriebsmittelzuteilung nie an die Reihe kommen
    - geschieht meist bei mangelnder Fairness, beispielsweise von Schedulern

93. **Wann kann es im Zusammenhang mit kritischen Abschnitten zu sog. Prioritätsinversion kommen?**
    - wenn ein Prozess hoher Priorität einen SpinLock will, in dem sich bereits ein Prozess niedrigerer Priorität befinde

94. **Du sollst einen Web-Server für periodisch stark besuchte Webseiten entwerfen und sollst für Dein System die maximal mögliche Nebenläufigkeit garantieren. Wie synchronisierst Du den lesenden / schreibenden Zugriff auf die Web-Seiten?**
    - die aufragenden Leser bekommen eine höhere Priorität als die Schreiber. Damit ist allerdings Aktualität der Webseiten eher schlecht.
    - siehe 32

95. **Du sollst einen Web-Server für stark besuchte Webseiten entwerfen und sollst für Dein System immer die größtmögliche Aktualität der Daten sicherstellen. Wie synchronisierst Du den lesenden / schreibenden Zugriff auf die Webseiten?**
    - die Schreiber bekommen eine höhere Priorität als die Leser, da sie die Seiten schnell aktualisieren sollen. Allerdings nimmt man dabei geringe Nebenläufigkeit in Kauf

96. **Du sollst einen Web-Server für stark besuchte Webseiten entwerfen und sollst für Dein System einen hohen Grad an Nebenläufigkeit bei gleichzeitiger Aktualität der Daten garantieren. Wie synchronisierst Du den lesenden / schreibenden Zugriff auf die Webseiten?**
    - gleiche Priorität für Leser und Schreiber → faire Verteilung

97. **Welches Kommunikationsmuster wird deiner Meinung nach in verteilten Systemen am häufigsten benutzt? Begründe deine Wahl!**
    - Rendevous: ist einfach zu synchronisieren. Server kann sicher sein, dass der Client die Nachricht empfangen hat!

98. **Auf welche Weise kann der Empfänger einer Nachricht adressiert werden?**
    - direkt: Prozess-ID
    - indirekt: Port, Mailbox

99. **Erkläre den Unterschied zwischen verbindungsloser & verbindungsorientierte Kommunikation!**
    - verbindungslos: sporadische Kommunikation mit wechselnden Kommunikationspartnern
    - verbindungsorientiert: immer zwischen 2 Partnern, die zuvor eine Verbindung aufgebaut haben, → hoher Aufwand zur Herstellung einer Verbindung

100. **Erkläre die verschiedenen Phasen bei der verbindungsorientierten Kommunikation!**
    - Aufbauphase: Austausch der Komminukationsparameter, Reservierung der Betriebsmittel
    - Kommunikationsphase: Austausch beliebig vieler Daten
    - Abbauphase: Freigabe von Betriebsmittel, Beendigung der Verbindung

101. **Erkläre die Semantik einer synchronisierten Send-Operation!**
    - Sender wartet blockierend bis der Empfänger empfangsbereit ist
    - Empfänger wartet blockierend, bis der Sender sendebereit ist

102. **Welche Problematik ist mit puffernden Nachrichtenoperationen verbunden?**
    - hoher Betriebsmittelverbrauch
    - Laufzeiteinbußen bei großen Nachrichten
    - Pufferung beim Sender oder Empfänger?

103. **Welche Problematik ist mit nicht puffernden Nachrichtenoperationen verbunden?**
- unübersichtlich, Synchronisationsfehler → Nachrichten können verloren gehen

104. **Was versteht man unter einem Rendevous?**
- ein synchronisiertes Send, wo zusätzlich ein Reply vom Empfänger gesendet wird, wenn „alles geklappt hat" → Sender weiß, dass die Daten richtig und vollständig angekommen sind
- typisch für Client- / Server-Applikationen

105. **Mit einem Rendevous ist eine spezifische Kontrollflusssteuerung verbunden. Nenne ein programmiersprachliches Konzept mit einem vergleichbaren Kontrollflussverlauf!**
- Prozeduraufruf

106. **Was versteht man unter einseitiger Kommunikation?**
- z.B. beim Zugriff auf entfernte Speicher, entfernte Schreib-Lese-Operationen werden synchron mit dem Aufrufer durchgeführt

107. **Wozu dient die Flusskontrolle?**
- um die Geschwindigkeit der Datenübertragung zu Kontrollieren
  (sonst: Puffer zu schnell voll → Pufferüberlauf → Pakete gehen verloren → noch mal senden)
- bei Datenströmen:
  → Blockieren des Senders, wenn Sende- / Empfangspuffer gefüllt sind
  → Blockieren des Empfänger, wenn keine Daten mehr vorhanden sind

108. **Was sind Signale (in der Interprozesskommunikation)?**
- Nachrichten der Länge 0

109. **Welche generellen Form der Gerätesteuerung gibt es?**
- aktives Warten („Polling") auf das Gerät
- Interruptbasierend
- Mischformen aus beiden

110. **Wie kommunizieren Gerätetreiber mit Geräte-Controllern?**
- Lese- / Schreiboperationen an bestimmten Ports oder bestimmter Speicher-Adresse

111. **Was versteht man unter „busy waiting" (in Verbindung mit Geräten / Treibern)?**
= Polling
- aktives Warten, Monopolisierung der CPU / Betriebsmittel während der Warteoperation
- z.B. ständiges Abfragen von Statusregistern als Wartebedingung
- z.B. vor / nach dem Lesen / Schreiben von Daten / Absetzen von Kommandos

112. **Was versteht man unter „memory mapped I/O"?**
- Datentransfer zum / vom Gerät findet über ausgezeichnete Speicherbereiche statt
  → Gerät verhält sich wie ein Speicher
  → Zugriff aufs Gerät wie Zugriff auf Speicher

113. **Was versteht man unter „programmed I/O"?**
- Datentransfer findet mit Hilfe der CPU statt

114. **Skizziere die typischen Aktivitäten eines Gerätetreibers im Falle einer Interrupt-getriebenen Gerätesteuerung**

115. **Skizziere die typischen Aktivitäten ein Gerätetreibers für eine DMA fähige Platte**
Siehe Seite 21

116. **Warum werden Schreib / Leseanforderungen an eine Platte zunächst sortiert?**
Weil der Schreib-Lesearm langsam ist (mechanisch) und man unnötige, zeitraubende Bewegungen des Armes vermeiden möchte

Welche Form der Sortierung ist am effizientesten?
- Shortest-Seek-First – Methode (SSF)
  → die Aufträge werden nach dem Spurabstand zur aktuellen Position sortiert

Welche Form der Sortierung wird in der Praxis benutzt und warum?
- Elevator-Algoritmus
  → Die Aufträge werden in Laufrichtung nach Spuren sortiert (Arm bewegt sich also immer von innen nach außen und wieder zurück). Ist im Gegensatz zur SSF-Methode fair und bevorzugt nicht die mittleren Spuren

117. **Warum ist eine uniforme Treiberschnittstelle (z.B. für block- bzw. zeichenorientierte Gräte) wichtig?**
- um den übergeordneten Ein- / Ausgabesystemen eine einheitliche Schnittstelle zur Verfügung zu stellen

118. **Warum ist es sinnvoll, Ein- und Ausgabedaten zu puffern?**
- Entkopplung von Benutzer- und Systemaktivitäten, Parallelität von Benutzerprozess und Gerät
- langsame Geräte sollen nicht unnötig Aufhalten

119. **Womit können Ein - / Ausgabekanäle in UNIX-Systemen verknüpft sein?**
- Dateien
- Terminals
- Kommunikationskanäle (quasi unendlicher Datenstrom)

120. **Welche besondere Rolle spielen die Kanäle 0, 1 und 2 in UNIX-Systemen?**
0 = Standardeingabe (standard in)
1 = Standardausgabe (standard out)
2 = Fehlerausgabe (standard error-out)

121. **Welche Aktionen muss ein Dateisystem bei Öffnen einer Datei durchführen?**
1. Gerät finden
   a) über Major Device Number → gibt an, um welchen Typ von Gerät es sich handelt
   b) über Minor Device Number → gibt an, um welches konkrete Gerät es sich handelt
2. Katalogeintrag der Datei finden
3. Zugriffsrechte überprüfen
4. Schreib-/Lese-Synchronisation
5. Lokalität der Datei feststellen
6. Verwaltungsdatenstrukturen anlegen, zur späteren Ein- / Ausgabe

122. **Welche Aktionen muss ein Dateisystem bei einer Ein- / Ausgabeoperation auf eine Datei ausführen?**
- logische Blocknummer ermitteln
- Block anfordern (über Cache oder bei *Cache Miss* direkt von der Platte)
- Offset im Block bestimmen und Daten übertragen
- Schreib- / Leseposition aktualisieren
- solange wiederholen, bis alle Daten übertragen sind

123. **Welche Problematik ist mit dem Cache des Dateisystems verbunden und wie kann man sie vermeiden bzw. entschärfen?**
- Bei Absturz des Rechners oder Stromausfall befinden sich möglicherweise noch ungeschriebene
  Daten im Cache die dann verloren gehen
- Lösung: entweder Schreiboperationen gar nicht cachen
  oder nur dateisystemrelevanten nicht, und/oder periodisch den Cache leerschreiben

124. **Erkläre die Funktionsweise eines Block Caches!**
- bei der Verwaltung des Caches ist es sinnvoll, immer nur die Blöcke im Cache zu halten, auf die häufig zugegriffen wird
- dabei sind immer die Blöcke im Cache, die vor kurzer Zeit oft genutzt wurden (Seite 13)

125. **Wie kann man einen LRU-Algorithmus für einen Block Cache realisieren?**
welche immer den am längsten nicht mehr zugegriffenen Block im Cache austauscht. (Seite 14)

126. **Diskutiere die Vor-/Nachteile von verketteten Blöcken zur Speicherung einer Datei auf der Platte!**
- keine externe Fragmentierung, aber schlechter wahlfreier Zugriff auf Dateiinhalte
+ sequenzieller Zugriff ist dagegen einfach
- schlechte Fehlereingrenzung aufgrund der verketteten Datenstruktur
- erhöhter Aufwand bei Dateizugriffen, weil sich innerhalb der Datenblöcke Verwaltungsinformationen befinden

127. **Diskutiere die Vor-/Nachteile von Indirektblöcken zur Speicherung einer Datei auf der Platte!**
- schlechte Fehlereingrenzung bei Verlust eines Indirektblockes
+ schneller wahlfreier Zugriff auf Dateiinhalte (sehr gut bei kleinen Dateien, vertretbarer Aufwand bei großen Dateien)
+ Ein einzelner Indirektblock kann ggf. durch weitere Mehrfachindirektblocken ergänzt werden (z.B. angewandt bei UNIX I-nodes)

128. **Diskutiere die Vor-/Nachteile von FATs zur Speicherung von Dateien auf einer Platte!**
- Fehlereingrenzung ohne weitere Maßnahmen kaum gegeben, aufgrund der zentralen Tabelle
+ schnellerer wahlfreier Zugriff auf Dateiinhalte als bei einer Blockliste, wenn die indirekte Liste komplett im Hauptspeicher gehalten werden kann. . . nur machbar für kleine Platten
+ Verwaltungsdaten werden getrennt von den Datenblocken gehalten, somit bessere Handhabung und weitere Fehlertoleranzmaßnahmen sind ggf. durch redundante Speicherung realisierbar

129. **Was versteht man unter interner Fragmentierung?**
- Verwendung von gleichgroßen Bereichen / Blöcken zur Speicherung von Daten
→ somit entstehen Lücken innerhalb der Blöcke, wenn Daten diesen nicht komplett ausfüllen

130. **Was versteht man unter externer Fragmentierung?**
- Verwendung verschieden großer Blöcke
→ somit entstehen keine Lücken innerhalb der Blöcke, sondern nur Lücken zwischen den Blöcken

131. **Eine Ausgabe (cout) wird getätigt, erscheint aber nicht auf dem Monitor. Woran kann das liegen?**
- Ausgabe des Prozess' wurde zwischengespeichert (gepuffert), aber noch nicht ausgegeben, bevor der Prozess beendet wurde

132. **Was bedeutet „volatile int a" und wo wird es verwendet??**
- durch das volatile wird dem Compiler gesagt, dass dieser Wert nicht im Register zwischengespeichert werden darf. Zu jeder Abfrage des Wertes muss dieser neu ausgewertet werden
→ z.B. beim lesenden Zugriff auf Statusregister als Wartebedingung (in einer Schleife)