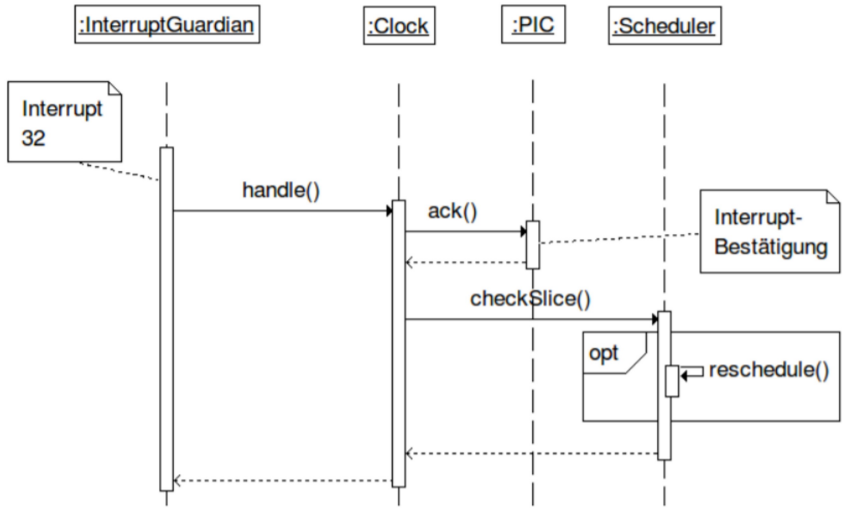
Übung 5

3. Theoretische Fragen

0. Worum ging es dieses mal?

- nichtblockierende Synchronisation
- semaphorenbasierte Synchronisation für Anwendungen

1. Wie erfolgt die Interrupt-Behandlung bisher? Wenn ein Interrupt eintritt, ab wann können neue Interrupts auftreten?



- neue Interrupts können in handle auftreten
- erst ab checkSlice sind Interrupts blockiert

2. Wozu dienen die Interrupt-Sperren (IntLock) und wann ist ihr Einsatz bisher notwendig?

- Einsatz in kritischen Abschnitten -> Schutz
    - in denen darf sich zu jeder Zeit nur ein Prozess befinden

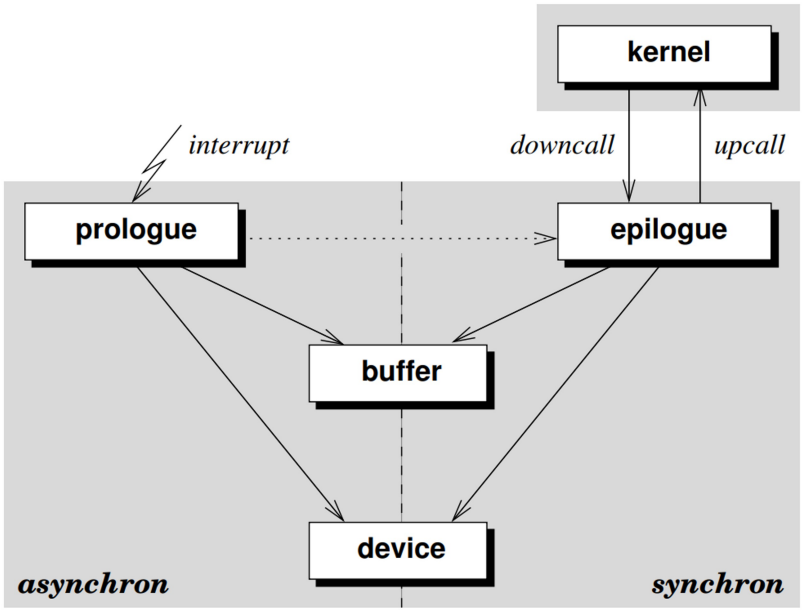
3. Wozu dienen bei der Interrupt-Behandlung Prolog und Epilog?

- Kern unterscheidet zwei logische Arbeitsmodi
- "normaler" Modus
    - wenn Kernaufrufe durch Prozesse getätigt werden
    - synchron zum laufenden Prozess
- "ausnahmebedingter" Modus
    - wenn ein Prozess unterbrochen wird
    - asynchron zum laufenden Prozess
- zu jedem Zeitpunkt kann sich maximal ein Prozess im Kern befinden
- Aktivitäten des Interrupt-Handlers werden analog zu den Arbeitsmodi des Kern aufgeteilt
  -> asynchroner Prolog
  -> synchroner Epilog

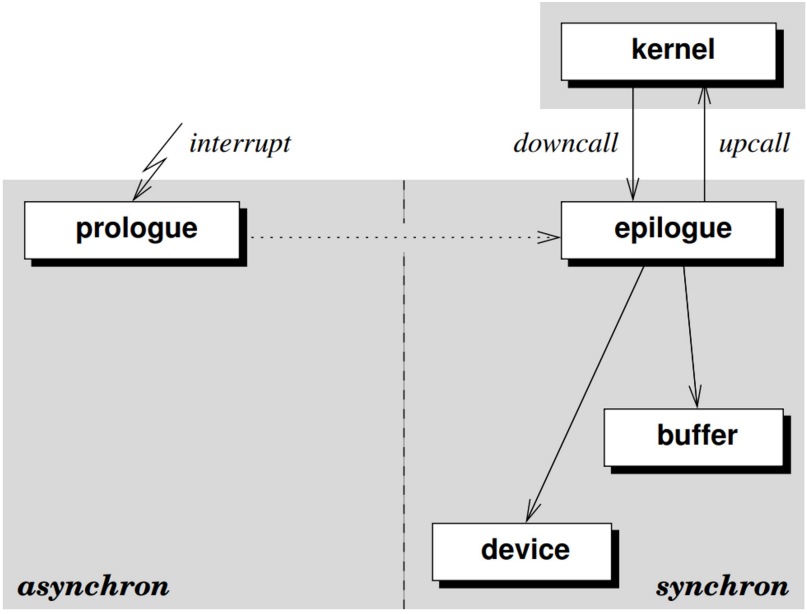
4. Worin unterscheiden sie sich?

- Prolog
    - wird mit HW-Priorität des Interrupts gestartet
        - kann den Kern jederzeit während einer normalen Arbeitsphase unterbrechen
            - kurze Reaktionszeit
        - darf deshalb keine Operationen des Kernel-Monitors direkt aufrufen
        - Interrupts niedrigerer Priorität sind während der Ausführung gesperrt

        - Interrupts höherer Priorität können jederzeit auftreten
    - behandelt den Interrupt am Gerät
        - Ein- und Ausgabedaten sind ggf. zu puffern
        - Interrupt quittieren



- Epilog
    - führt eine Nachbehandlung des Interrupts aus
        - schlafende Prozesse aufwecken
        - Rescheduling ausführen
    - nur bei Bedarf aktiviert
        - auf Anforderung des Prologs
    - Ausführung findet innerhalb des Kernel-Monitors statt
        - Epiloge durch Prologe unterbrechbar
    - darf nur ausgeführt werden, wenn Kernel-Monitor frei ist
    - jedem Interrupt ist ein Objekt zugeordnet
        - das atomar in Warteschlange eingefügt wird
        - wenn Kernel-Monitor besetzt
        - erweitertes Gate-Objekt
    - Warteschlange wird erst abgearbeitet, wenn
        - alle Prologe beendet sind
        - Kernel-Monitor schließlich verlassen wird



5. Welchen Vorteil bringt die Aufteilung?

- Konsistenz ist sichergestellt
    - Korrektheit der speichernden Daten

6. Was ist ein Monitor?

- engl. to monitor: überwachen, kontrollieren
- sichert für eine Menge von Methoden automatischen wechselseitigen Ausschluss zu
- für Operationen des Monitors gelten dadurch ähnliche Prozesse wie für kooperative Prozesse
    - können uns auf Wartebedingungen konzentrieren
    - müssen uns nicht um kritische Abschnitte kümmern
    - obwohl die Operationen durch präemptive Prozesse ausgeführt werden
- Bedingungsvariablen dienen zum Warten
    - Prozess wartet auf die Erfüllung einer Bedingung
        - indem wait() auf einer Bedingungsvariablen ausgeführt wird
        - Prozess blockiert und gibt Monitor frei
    - andere Prozesse können durch signal() anzeigen, dass
        - wartenden Prozess weiterarbeiten kann
        - da seine Wartebedingung nun erfüllt ist

7. Wozu wird er verwendet? Welches Konzept ersetzt er?

- dient der Synchronisation von Prozessen
- muss sich nicht mehr um kritische Abschnitte kümmern

8. Was ist eine Semaphore?

- abstrakter Datentyp
- wird zur Steuerung von Prozessen verwendet
- zählende Semaphoren
    - zur Betriebsmittelverwaltung und Realisierung synchronisierter Datentypen
    - werden mit Zählwert vorinitialisiert und arbeiten als Signalzähler
    - wait()
        - blockiert den aufrufenden Prozess, wenn Zähler == 0
        - ansonsten wird Zähler dekrementiert und Prozess läuft weiter
    - signal()
        - weckt den nächsten schlafenden Prozess auf
        - wenn kein Prozess schläft, wird Zähler inkrementiert
    - Signalzähler stellt sicher, dass kein signal-Aufruf verloren geht
    - schlafende Prozesse werden idR in einer Warteschlange verwaltet
        - meiste Semaphorenimplementierung sind dadurch fair
- binäre Semaphoren
    - zum Schutz kritischer Abschnitte
    - ähnliche Funktion wie Sperren
    - idR jedoch blockierend und fair

9. Was ist private Vererbung?

- normal
  class Student : public Mensch { }
- private Vererbung
  class Student : private Mensch { }
    - dadurch werden alle vererbten public Attribute und Methoden aus Mensch zu private
- erbende Klasse nutzt Methoden und Attribute der Basisklasse, ohne nach außen als Unterklasse dieser Klasse zu gelten
- Vererbung zwecks Teilen gemeinsamer Implementierung