

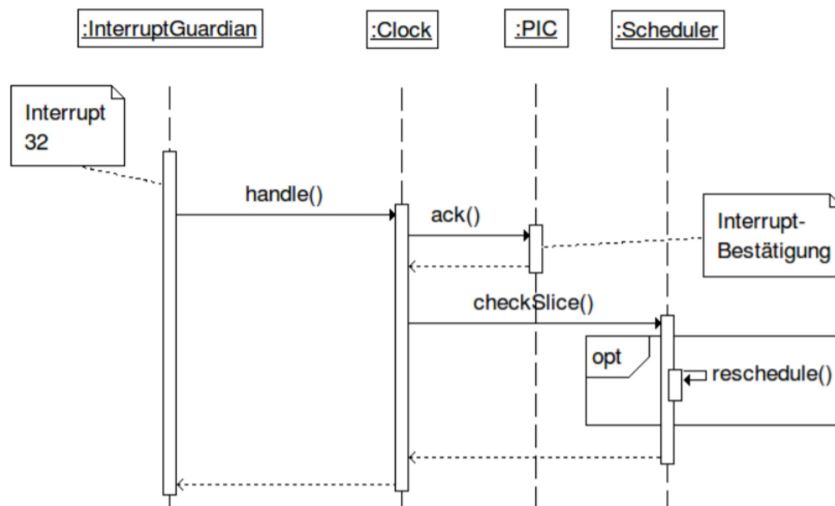
# Übung 5

## 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?

- Interrupt-Sperren sperren Interrupts ?!
- Einsatz in kritischen Abschnitten
  - o in denen darf sich zu jeder Zeit nur ein Prozess befinden

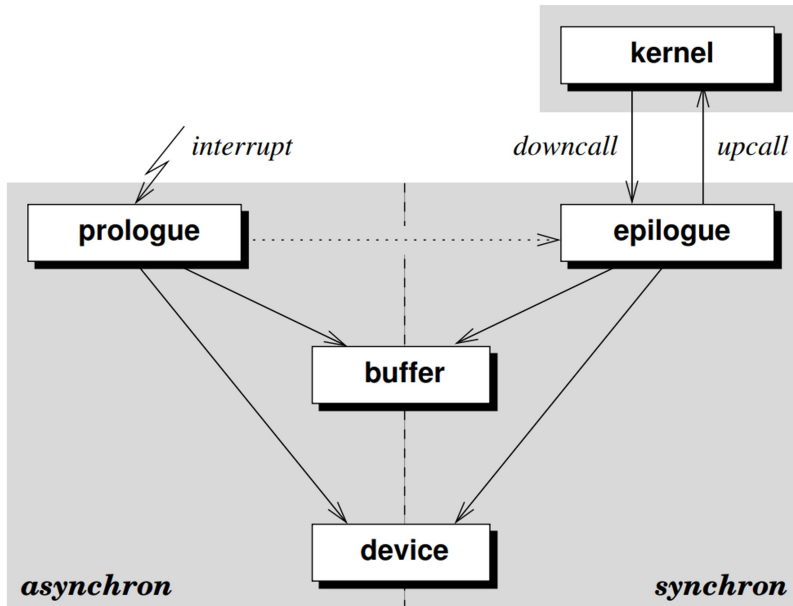
### 3. Wozu dienen bei der Interrupt-Behandlung Prolog und Epilog?

- Kern unterscheidet zwei logische Arbeitsmodi
- "normaler" Modus
  - o wenn Kernaufrufe durch Prozesse getätigt werden
  - o synchron zum laufenden Prozess
- "ausnahmebedingter" Modus
  - o wenn ein Prozess unterbrochen wird
  - o 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

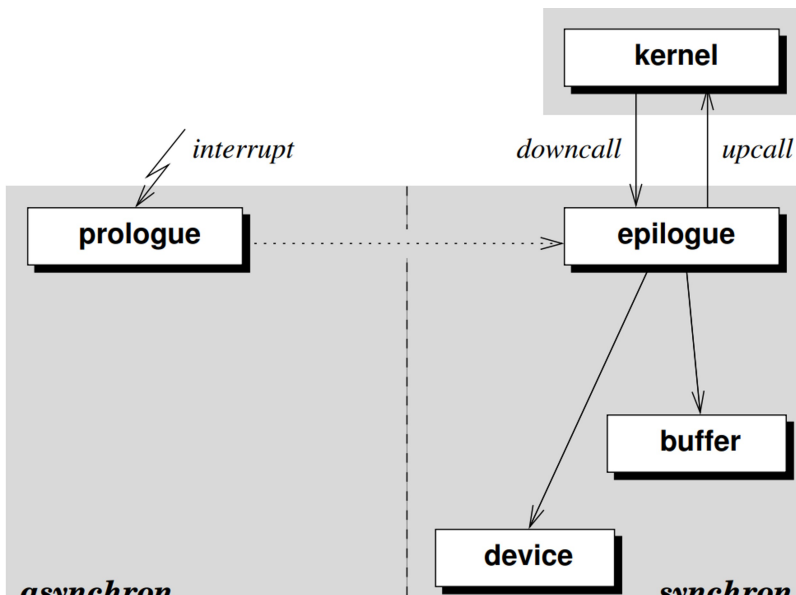
### 4. Worin unterscheiden sie sich?

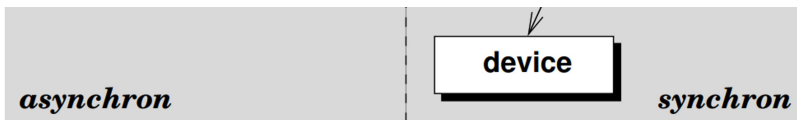
- Prolog
  - o wird mit HW-Priorität des Interrupts gestartet
    - kann den Kern jederzeit während einer normalen Arbeitsphase unterbrechen
      - kurze Reaktionszeit
    - darf deshalb keine Operationen des Kernel-Monitors direkt aufrufen

- 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
    - Epilog 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?

- gute Frage ...

#### 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
  - o können uns auf Wartebedingungen konzentrieren
  - o müssen uns nicht um kritische Abschnitte kümmern
  - o obwohl die Operationen durch präemptive Prozesse ausgeführt werden
- Bedingungsvariablen dienen zum Warten
  - o Prozess wartet auf die Erfüllung einer Bedingung
    - indem wait() auf einer Bedingungsvariablen ausgeführt wird
    - Prozess blockiert und gibt Monitor frei
  - o 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

#### 8. Was ist eine Semaphore?

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

#### 9. Was ist private Vererbung?

- normal  
class Student : public Mensch { }
- private Vererbung  
class Student : private Mensch { }
  - o dadurch werden alle vererbten public Attribute und Methoden aus Mensch zu private