Fachhochschule Vorarlberg

Betriebssysteme

Informatik

WS13/14

Prozesse: Scheduling Teil2

Armin Simma

Scheduler

Aufgaben des Schedulers (Dispatchers):

- Entscheidet auf Basis einer vorgegebenen Strategie, welcher Prozess als nächster ausgeführt wird, wenn mehrere Prozesse im Zustand „bereit“ sind.

Zwei grundlegende Varianten:

- **Nicht preemptives Scheduling (Run-to-Completion;** Event-Verarbeitung/Scheduling**):** Prozesse im Zustand „aktiv“ werden nur durch Warten auf Betriebsmittelanforderung (z.B. E/A) „blockiert“ (Stapel od. Batch-Betrieb ➔ traditionelle Betriebsart)
- **Preemptives Scheduling (Interrupt-Scheduling):** Prozesse können zu jedem Zeitpunkt vom BS unterbrochen (suspendiert) werden, so dass ein anderer Prozess zur Ausführung kommen kann.
  ➔ Kann zu kritischen Zuständen führen, die aufwändige Behandlung notwendig machen (-> Semaphore)

Anforderungen an einen Scheduling-Algorithmus

Die Qualität eines Scheduling-Algorithmus hängt im wesentlichen von folgenden Kriterien ab:

- **Fairness**: Jeder Prozess sollte einen gerechten Anteil an der Prozessorzeit erhalten
- **Effizienz**: Der Prozessor sollte möglichst immer vollständig ausgelastet sein (sofern Aufträge vorliegen)
  - Auch Geräte sollten so hoch wie möglich ausgelastet sein
- **Antwortzeit**: Die Antwortzeit für die interaktiv arbeitenden Benutzer sollte minimal sein

Anforderungen an einen Scheduling-Algorithmus

Die Qualität eines Scheduling-Algorithmus hängt im wesentlichen von folgenden Kriterien ab:

- **Verweilzeit** : Die Wartezeit auf die Ausgabe anliegender Stapelaufträge sollte ebenfalls minimal sein
  - **Durchlaufzeit** (Ende – Start von Stapelauftrag) sollte minimiert werden
- **Durchsatz**: Die Anzahl der Aufträge, die in einem bestimmten Zeitintervall ausgeführt werden, sollte maximal sein.

Aufgabe

- Überlege dir einen Scheduling-Algorithmus, der die Anforderungen (so gut es geht) erfüllt.
- Welche Hilfsmittel würdest du einsetzen?
- Es gibt im System wichtige und weniger wichtige Prozesse. Hilfsmittel?

Ideen zu „gutem“ Scheduling

- Hilfsmittel: Queue, Time-Slice=Zeitscheibe, …
- I/O-bound (E/A-lastige) Prozesse gegenüber CPU-bound (CPU-lastige auch rechen-lastige) bevorzugen
  - Warum?
- Prioritäten verwenden
  - Achtung: dass niederpriore auch drankommen
- kurze Prozesse bevorzugen

Ideen zu „gutem“ Scheduling

- I/O-bound (E/A-lastige) Prozesse gegenüber CPU-bound (CPU-lastige auch rechen-lastige) bevorzugen
  - **Warum**?
    - CPU ist Engpass
    - Da E/A-lastiger Prozess nur kurz die CPU benötigt, gib diesem (kurz) die CPU, dann arbeitet E/A-lastiger Prozess auf E/A (ohne CPU-Verbrauch) weiter
    - Würde man E/A-lastigen Prozess nicht bevorzugen, wäre das E/A-Gerät wenig ausgelastet, da E/A-lastiger Prozess „im Stau“ vor der CPU stehen müsste. -> schlechter **Gesamt**-durchsatz, schlechte durchschnittliche Antwortzeit
    - Vergleiche Bsp: Kaufhaus
      - CPU=Infoschalter
      - E/A-lastiger Prozess = Kunde der selbständig im Regal was suchen kann (und nur kurze Infos braucht)
      - CPU-lastigr Prozess = Kunde der lange Frage am Infoschalter hat

Prioritäten –
Hilfsmittel für Prozessmanagement

Zur Umsetzung der Anforderungen an die Scheduling-Algorithmen ist es sinnvoll, Prioritäten einzuführen.

- Jedem Prozess wird eine Priorität Q_i zugeordnet und der Scheduler gibt stets dem Prozess mit höchster Priorität ($Q_i > Q_j$) den Vorrang.
- Bei *n* unterschiedlichen Prioritäten wird die Ready-Queue nach den Prioritäten der Prozesse in *n* Ready-Queues aufgesplittet.
- Der als nächstes zu aktivierende Prozess wird aus der Ready-Queue mit den höchsten Prioritäten entnommen.

➡ **Problem**: Starvation: Prozesse mit niedriger Priorität könnten ***verhungern (starve)***, falls stets höherpriore Prozesse vorhanden sind.

Lösung: Die Prioritäten der Prozesse können sich mit der Zeit ihrer Abarbeitung ändern.

Scheduling-Algorithmen

„Ablaufplanung“ suggeriert planbare Betriebsmittel-vergabe;
Nur dann möglich, wenn Betriebsmittelbedarf vorab bekannt;
➔ optimaler Zeitplan kann angegeben werden.

In modernen Systemen: Hauptlast durch interaktiven Betrieb;

➔ ist vorab nicht bekannt.

An die Stelle der Planung tritt daher eine dynamische Zuteilungsstrategie, die jeweils von der aktuellen Lastsituation ausgeht. Bekannte Zuteilungsstrategien:

- First Come First Served (FCFS)
- Round Robin
- Shortest Process next (SPN)
- Shortest Remaining Processing Time (SRPT)
- Priority Scheduling (PS)
- Multilevel Feedback Queueing

First Come – First Served (FCFS)

- Non-Preemptives Verfahren, bei dem Aufträge in der Reihenfolge des Eintreffens im System bearbeitet werden:
  - bereiter Prozess stellt sich in der Ready-Queue hinten an
  - bei freiem Prozessor: ältester Prozess wird aktiviert

Vorteile

- Einfach zu implementieren
- Geringer Verwaltungsaufwand
- Fair

Nachteile

- Kurz laufende Prozesse müssen u.U. sehr lange warten
- Konvoi-Effekt: viele kurze Prozesse folgen einem langen
  - -> für Durchlaufzeit schlecht
  - für Durchsatz egal
- Ungeeignet für Dialogsysteme

Round Robin

- Preemptives Verfahren mit festen Zeitintervallen.
  - Kurze Zeitscheibe: viel Overhead durch Kontextwechsel
  - große Zeitscheibe: Round Robin degeneriert zu FCFS
  - Anhaltspunkt: 80% aller Jobs kleiner als ein Zeitintervall.

Vorteile

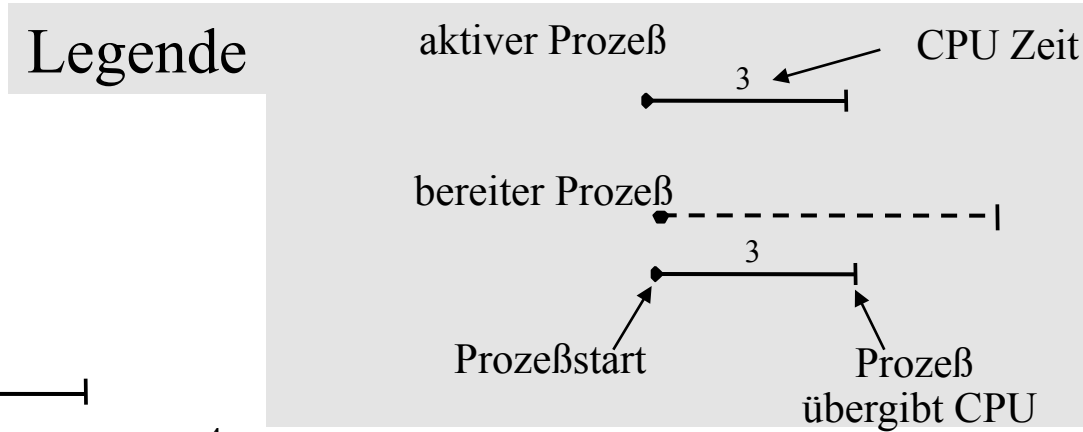
- Akzeptable Laufzeit für kurze Prozesse
- Einfache Implementierung
- Wenig Verwaltungsaufwand
- Fair

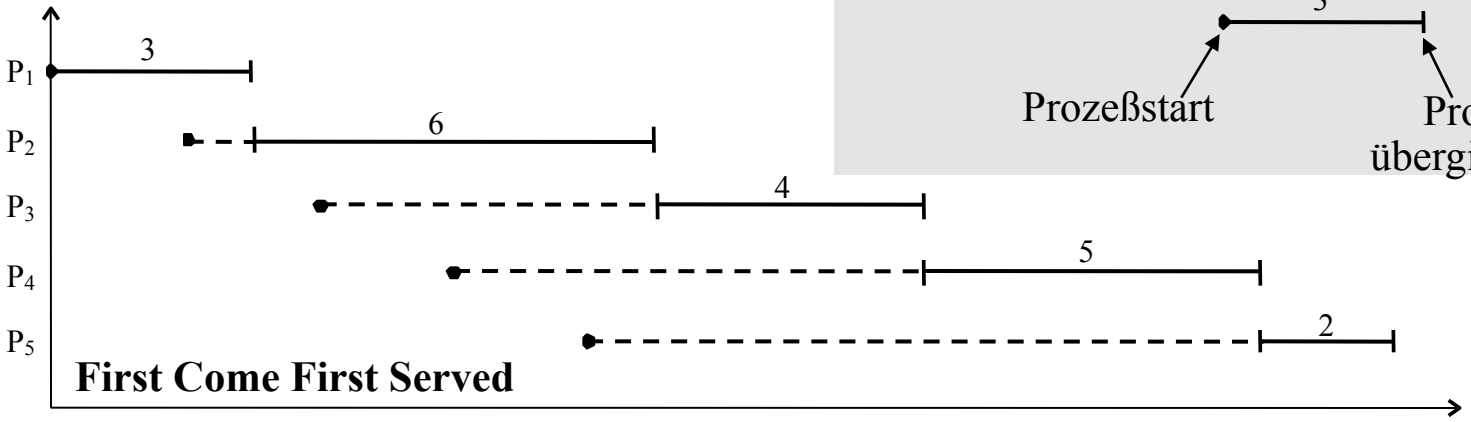
Nachteile

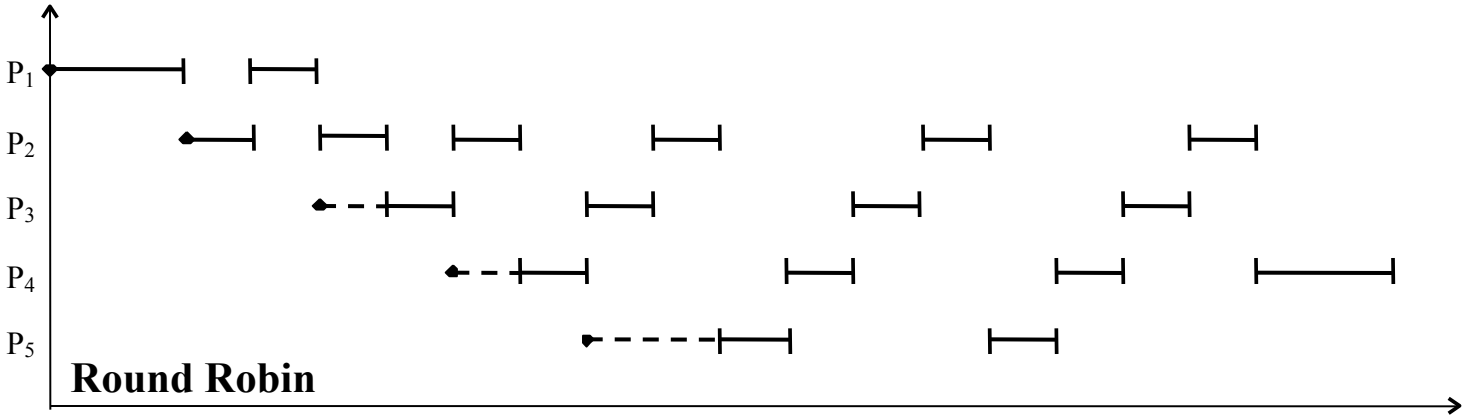
- Lang laufende Prozesse sind durch häufige Unterbrechungen eher im Nachteil

Vergleich: FCFS – Round Robin

Fachhochschule Vorarlberg







Shortest Process Next (SPN)

- Non-Preemptives Verfahren. Auftrag mit kürzester zu erwartender Abarbeitungsdauer wird ausgewählt. Abschätzung der Abarbeitungszeit durch Mittelwert bereits abgearbeiteter gleichartiger Aufträge

Vorteile

- Kurze Prozesse werden nicht benachteiligt

Nachteile

- Evtl. Verhungern von längeren Prozessen
- Non-Preemptiv, daher nur im batch-Betrieb sinnvoll
- Hoher Verwaltungsaufwand
- Bei falscher Schätzung
  - (reale Zeit >> geschätzte) Konvoi-Effekt
  - reale Zeit << geschätzte: schlechte Antwortzeit u. evtl. Verhungern

Shortest Remaining Processing Time (SRPT)

- Preemptives Analogon zu SPN. sobald neuer Prozess mit kürzerer zu erwartender Restabarbeitungsdauer eintrifft, wird dieser aktiviert. Im Gegensatz zu Round Robin gibt es keine weiteren Unterbrechungen
- auch als SRT abgekürzt

Vorteile

- Kurze Antwortzeiten
  - kurz nur, wenn Restzeit bis zum Zeitpunkt “Warte auf E/A” und nicht Gesamtzeit als Kriterium
  - Ev. Schlechte Durchlaufzeit
- Kurze Prozesse werden nicht benachteiligt

Nachteile

- Verhungern von längeren Prozessen
  - Noch kritischer als SPN
- Hoher Verwaltungsaufwand

Priority Scheduling (PS)

- Non-Preemptives Verfahren, das die Prozesse in Prioritätsklassen einteilt:
  - Realisierung durch mehrere Ready-Queues
  - **Problem**: evtl. Verhungern von Prozessen mit niedriger Prio.
  - **Lösung**: „Aging“ ➔ Priorität von wartenden Prozessen steigt

Vorteile

- Wichtige Aufgaben werden schnell erledigt

Nachteile

- Eventuell nicht fair wegen Verhungern niedrig priorer Prozesse
- Prioritätenvergabe unklar
- Höherer Verwaltungsaufwand

Multilevel Feedback Queuing

- Preemptives Verfahren; arbeitet sowohl mit Prioritätsklassen als auch mit Zeitscheiben
  - Prioritäten werden dynamisch vergeben; bei der ersten Aktivierung erhält jeder Prozess die höchste Prioritätsklasse; mit jeder Preemption wird er in die nächst niedrigere Ready-Queue eingestellt
  - Einrichten von Unterbrechungen im Round Robin Verfahren
  - Kurze Prozesse werden relativ schnell abgeschlossen
  - Lange Prozesse gelangen schließlich in die niedrigstpriore Queue und bleiben dort

Vorteile

- Keine Kenntnisse über vor-aussichtliche Abarbeitungszeit – trotzdem werden kurze Aufträge bevorzugt
- Dynamische Beurteilung des Verhaltens von Prozessen

Nachteile

- Hoher Verwaltungsaufwand
- evtl. Verhungern

Prinzipien der Ablaufsteuerung

- **Ziele**:
  - gleichmäßige Auslastung der Prozessoren und anderen Betriebsmittel (z.B: E/A-Geräte)
  - Geringer Verwaltungsaufwand (verursacht Overhead)
- **Lösungsansätze:**
  - Vernünftige Auftragsmischung (job mix) gewährleisten
    - Mindestens ein rechenintensiver Prozess (viel CPU-Zeit, wenig E/A)
    - Mehrere E/A-intensive Prozesse (gute Geräteauslastung)
  - Prioritätenzuordnung
    - Rechenintensive Prozesse: niedrige Priorität
    - E/A-intensive Prozesse: hohe Priorität (meist „blockiert“)
- **Problem:**
  - Optimale Auftragsmischung ist lastabhängig; nicht vorhersagbar (Nutzergesteuert)

Übungsaufgabe 1

- Ermitteln Sie den Trace für
  - First Come First Served (FCFS)
  - Round Robin (RR) mit q=1 **und** q=4
  - Shortest Process next (SPN)
  - Shortest Remaining Processing Time (SRPT)
  - Multilevel Feedback Queueing (MLFQ) mit q=1 **und** q=2^i

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Übungsaufgabe 1

- Lösung an der Tafel
- mitschreiben!

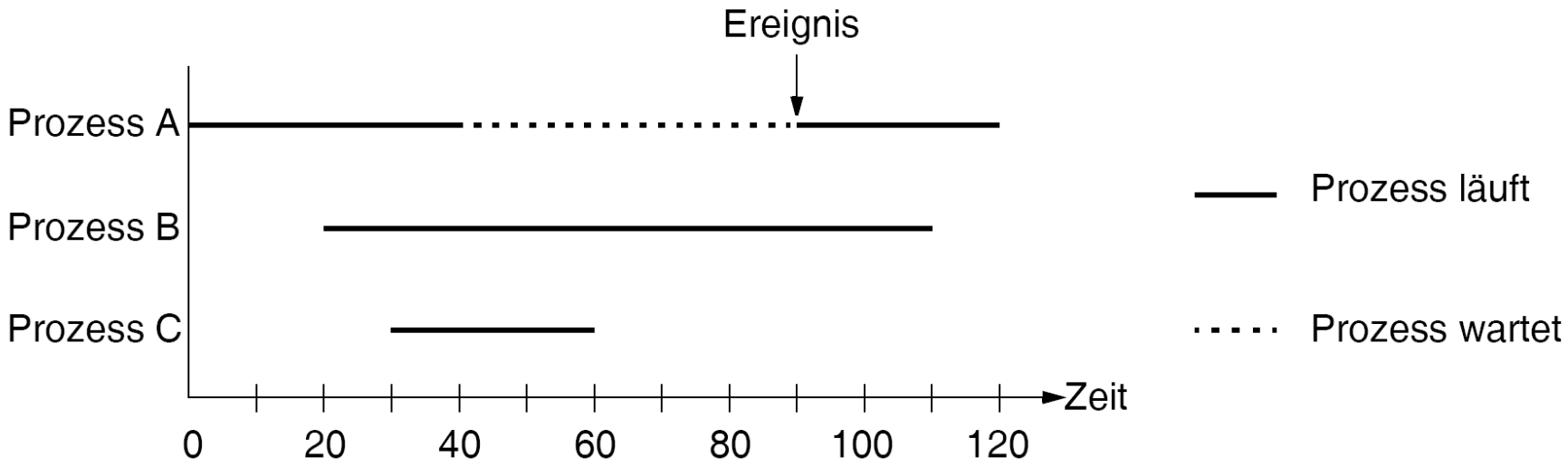
Gleichzeitige Ereignisse bei präemptiven Verfahren

- Was passiert, wenn **gleichzeitig** neuer Prozess ankommt **und** Zeitscheibe zu Ende

- Reihenfolge beim Ablauf Implementierungsabhängig
- Bei uns **immer**:
  - Zu jedem Zeitpunkt folgenden Ablauf in **genau dieser Reihenfolge**
  1. Falls neuer Prozess ankommt: hinten in Queue stellen
  2. Ist mind. ein bereiter Prozess in der Queue?
     - wenn ja und wenn Zeitscheibe zu Ende: Preemption: laufender Prozess hinten in Queue stellen
     - wenn nein: Weiterlaufen lassen

Übungsaufgabe 2

- Prozesse mit E/A-Wartezeiten (blockiert-Zustand)
- Angabe: Ablauf, wenn 3 CPUs vorhanden



Gleichzeitige Ereignisse bei präemptiven Verfahren

- Was passiert, wenn gleichzeitig **neuer Prozess P1** ankommt und **Zeitscheibe P2 zu Ende** und **E/A-Wartezeit P3** ist vorbei.

- Reihenfolge beim Ablauf Implementierungsabhängig
- Bei uns **immer**:
  - Zu jedem Zeitpunkt folgenden Ablauf in **genau dieser Reihenfolge**
  1. Falls neuer Prozess ankommt: hinten in Queue stellen
  2. Falls E/A-Wartezeit von Prozess zu Ende: hinten in Queue
  3. Ist mind. ein bereiter Prozess in der Queue?
     - wenn ja: Preemption: laufenden Prozess hinten in Queue stellen
     - wenn nein: Weiterlaufen lassen

Übungsaufgabe 2

- Lösung an der Tafel
- mitschreiben!

Statistiken

- Ankunftszeitpunkt (arrival time)
- Endzeitpunkt (finish time)
- Durchlaufzeit (turnaround time Tr)
- Bedienzeit (service time Ts)

} Folgeslide

- Wartezeit
- Antwortzeit (response time)
- Durchsatz (throughput)
- CPU-Auslastung (meist in % in bestimmtem Beobachtungszeitraum)

Statistik Übungsaufgabe 1

Fachhochschule Vorarlberg

	Process	A	B	C	D	E	
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	Mean
FCFS	Finish Time	3	9	13	18	20	
	Turnaround Time (T_r)	3	7	9	12	12	8.60
	T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$	Finish Time	4	18	17	20	15	
	Turnaround Time (T_r)	4	16	13	14	7	10.80
	T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$	Finish Time	3	17	11	20	19	
	Turnaround Time (T_r)	3	15	7	14	11	10.00
	T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN	Finish Time	3	9	15	20	11	
	Turnaround Time (T_r)	3	7	11	14	3	7.60
	T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT	Finish Time	3	15	8	20	10	
	Turnaround Time (T_r)	3	13	4	14	2	7.20
	T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59
HRRN	Finish Time	3	9	13	20	15	
	Turnaround Time (T_r)	3	7	9	14	7	8.00
	T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$	Finish Time	4	20	16	19	11	
	Turnaround Time (T_r)	4	18	12	13	3	10.00
	T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$	Finish Time	4	17	18	20	14	
	Turnaround Time (T_r)	4	15	14	14	6	10.60
	T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

Statistik Übungsaufgabe 2

- Übung

- Ankunftszeitpunkt (arrival time Ta)
- Endzeitpunkt (finish time Tf)
- Durchlaufzeit (turnaround time Tr = Tf-Ta)
- Bedienzeit (service time Ts) **ohne E/A-Wartezeit**
- zusätzlich: Bedienzeit+E/A-Wartezeit (total service time E/A T_{sea})

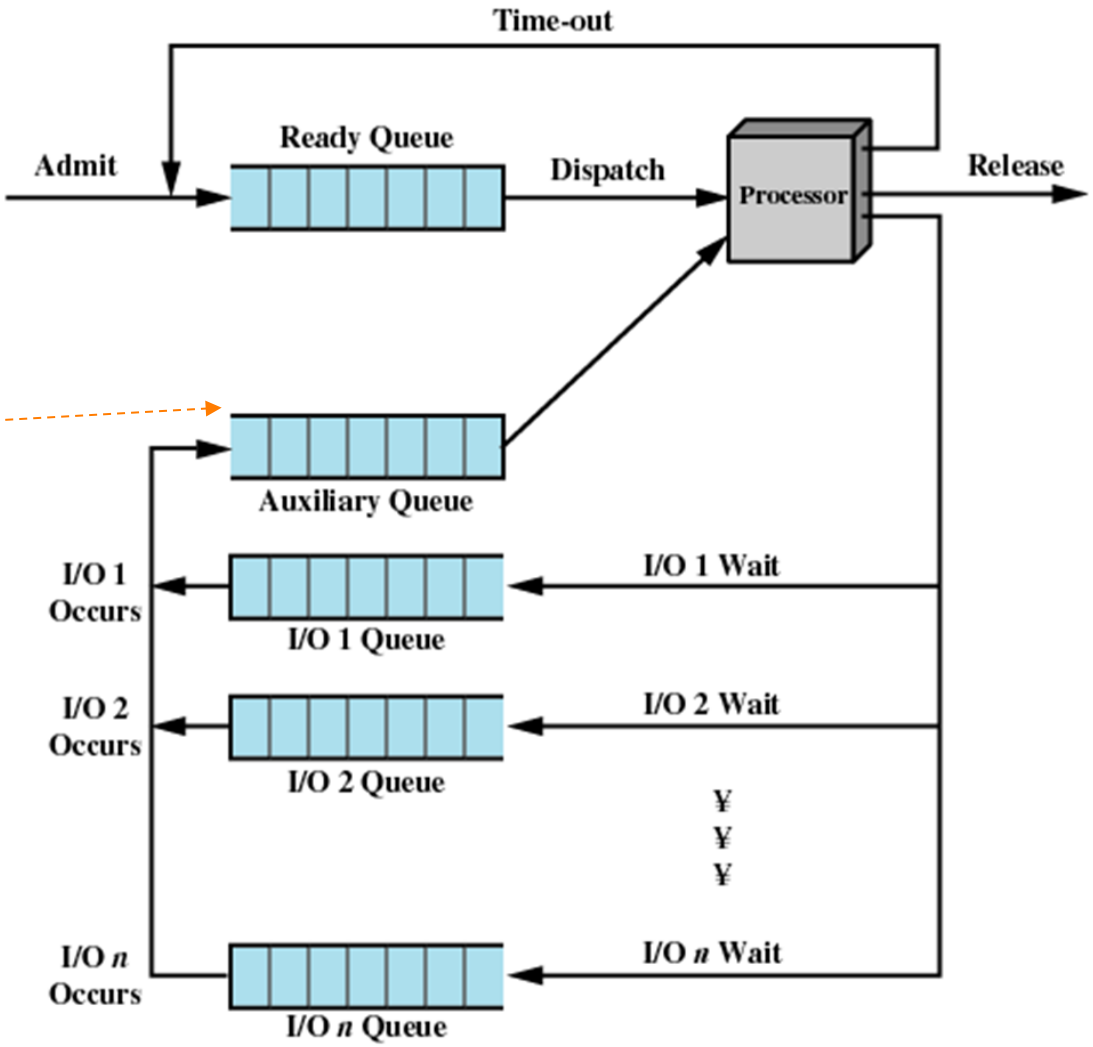
- sowohl Tr/Ts als auch Tr/T_{sea} ermitteln

Interaktive Prozesse

- Bei Interaktiven Prozessen (Dialogsystem) ist Antwortzeit besonders wichtig
- Strategien zur Verbesserung:
  - E/A-lastige Prozesse bevorzugen
    - Warten auf User-Interaktion ist E/A (Tastatur oder Maus)
  - bei RR: virtual RR
  - bei MLFQ: blockierter Prozess kommt nach Blockierung (bei Ende E/A-Wartezeit; bei INT) in höchstpriore Queue (Q0)
  - bei Priority Scheduling: blockierter Prozess kommt danach in höherpriore Queue (z.b. Windows)

Interaktive Prozesse

- virtual RR
  - da bei RR E/A-lastige Prozesse eher im Nachteil
- zusätzliche höherpriore bereit-Queue
  - blockierter Prozess kommt danach in höchstpriore Queue



Gleichzeitige Ereignisse bei präemptiven Verfahren

*Was passiert, wenn gleichzeitig **neuer Prozess P1** ankommt und **Zeitscheibe P2 zu Ende** und **E/A-Wartezeit P3** ist vorbei.*

- Reihenfolge beim Ablauf Implementierungsabhängig
- Bei uns **immer**:
  - Zu jedem Zeitpunkt folgenden Ablauf in **genau dieser Reihenfolge**
  1. Falls neuer Prozess ankommt: hinten in Queue stellen
  2. Falls E/A-Wartezeit von Prozess zu Ende: hinten in Queue
  3. Zeitscheibe zu Ende und mind. ein bereiter Prozess in der Queue?
     - wenn ja:
       1. Auswahl des nächsten Prozesses (und in temp speichern)
       2. Preemption: laufender Prozess hinten in Queue stellen
       3. temp-Prozess wird aktiv
     - wenn nein: Weiterlaufen lassen

MLFQ

- Warum wird ein gerade unterbrochener Prozess davon „gesperrt“, die CPU gleich wieder zu bekommen, auch wenn dieser Prozess eine (viel) höhere Priorität hat als die anderen (ready=bereiten) Prozesse
- Sinn von MLFQ ist **nicht**, daß
  - hohe Priorität bedeutet: Prozess darf **lange** laufen
- **sondern** Sinn von MLFQ ist,
  - hohe Priorität bedeutet: Prozess kommt **häufig** dran
  - Idee dahinter: E/A-bound-Prozesse sollen schnell (häufig) drankommen (Antwortzeit bei interaktiven Prozessen soll kurz sein)
  - Diese Idee sieht man besonders bei MLFQ mit q=2^i
    - Quantum ist für höherpriore Prozesse sogar **kleiner**

MLFQ Regeln

- wenn in irgendeiner Queue ein anderer (bereiter) Prozess ist, wird der aktive unterbrochen (egal ob der Aktive in einer höherprioren queue war).
- Der Unterbrochene (vorher aktive) kann nicht sofort wieder drangenommen werden (egal ob der Unterbrochene in einer niederprioren Queue ist als die anderen Bereiten). Der Unterbrochene kommt also in die entsprechende Queue (eins tiefer) und kommt erst bei der nächsten Unterbrechung (eines anderen Prozesses) eventuell wieder dran.

MLFQ Regeln

- wenn ein neuer Prozess ankommt, gilt die Reihenfolge bezüglich Einordnern in die höchste queue, wie in den Folien 29 angegeben.
  - Ein neu ankommender unterbricht aber einen aktiven niemals **innerhalb** seines Quantums (dasselbe bei RR). Wenn der aktive aber gerade (gleichzeitig mit der Ankunft des neuen) seine Zeitscheibe beendet hat , kommt der neue dran, falls der neue der vorderste in der höchsten queue ist.

MLFQ

- Wenn Prozess X der einzige in allen Queues ist und Quantum von X abgelaufen ist:
  - Das ist keine *Preemption* (=Verdrängung)
  - daher fällt X nicht in tiefere Queue