English Edition

Scheduling (3)


```
Sep 19 14:20:18 amd64 sshd[20494]: Accepted ... from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: ST...
Sep 20 01:00:01 amd64 /usr/sbin/cron[2... ...in/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[765...
Sep 20 02:00:01 amd64 /usr/sbin/c... ...evlogmgr -c 'age > "30d"')
Sep 20 02:00:01 amd64 syslog-n...
Sep 20 12:46:44 amd64 sshd[... ...ff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 sy...
Sep 20 12:48:41 amd64 ... ...:ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd... ...m ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 ... ...rom ::ffff:87.234.201.207 port 64242
Sep 20 15:27:...
Sep 20 16:3... ...esser from ::ffff:87.234.201.207 port 63375
Sep 20 1... ...pped 0
Sep 2... ...a for esser from ::ffff:87.234.201.207 port 63546
Se... ...(root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
...ATS: dropped 0
...878]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
...]: STATS: dropped 0
...: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
...g[7653]: STATS: dropped 0
...1269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
...slog-ng[7653]: STATS: dropped 0
...syslog-ng[7653]: STATS: dropped 0
...4 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
S... amd64 syslog-ng[7653]: STATS: dropped 0
Sep ...1 amd64 /usr/sbin... ...499]: (root) ...MD (/sbin/evlogmgr -c ...age > "30d"')
Sep ...0:01 amd64 syslog-n...7653]: STATS: drop...d 0
Sep 2...:23:21 amd64 syslog-ng...]: ST...S: drop...
Sep 23 01:00:01 amd64 /usr/sbin/cro...: (root... CMD ... "se...rit...=DE...G")
Sep 23 01:00:01 amd64 syslog-ng[7653]... ST...S: drop...d 0
Sep 23 02:00:01 amd64 /usr/sbi...5555... ...ot) CMD ... ...og... ...'ag... ...0d"')
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
```

Beispiele für Scheduler

Concrete scheduling methods

1. for batch systems
2. for interactive systems
3. for real time systems

Schedulers for interactive systems

```
Sep 19 14:20:18 amd64 sshd[20494]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[549[illegible]oot) CMD [illegible]/sbin/evlogmgr [illegible]c 'age > [illegible]0d"')
Sep 22 02:00:01 amd64 syslog-ng[7653]: [illegible]
Sep 22 20:23:21 amd64 syslog-ng[7653]: STA[illegible]pped
Sep 23 01:00:01 amd64 /usr/sbin/cron[24739]: [illegible]EBUG")
Sep 23 01:00:01 amd64 syslog-ng[7653]: [illegible] dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[25555]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[[illegible]554]: Accepted publickey for esser from [illegible]ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslo[illegible]
Sep 23 18:04:34 amd64 sshd[[illegible] port [illegible]2093
Sep 24 01:00:01 amd64 /usr/[illegible]erity=DEBUG[illegible]
Sep 24 01:00:01 amd64 syslog-ng[76[illegible]3]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
```

Interactive systems

- Typical: Interactive and background processes
- Desktop and server PCs
- Possibly several / many users, who share computing capacity
- Scheduler for interactive systems applicable in batch systems – bot not vice versa

Interactive systems

Schedulers for interactive systems

- Round Robin
- Priority Scheduler
- Lottery Scheduler
- Fair Share Scheduler

Round Robin / Time Slicing (1)

Like FCFS – but with preemption

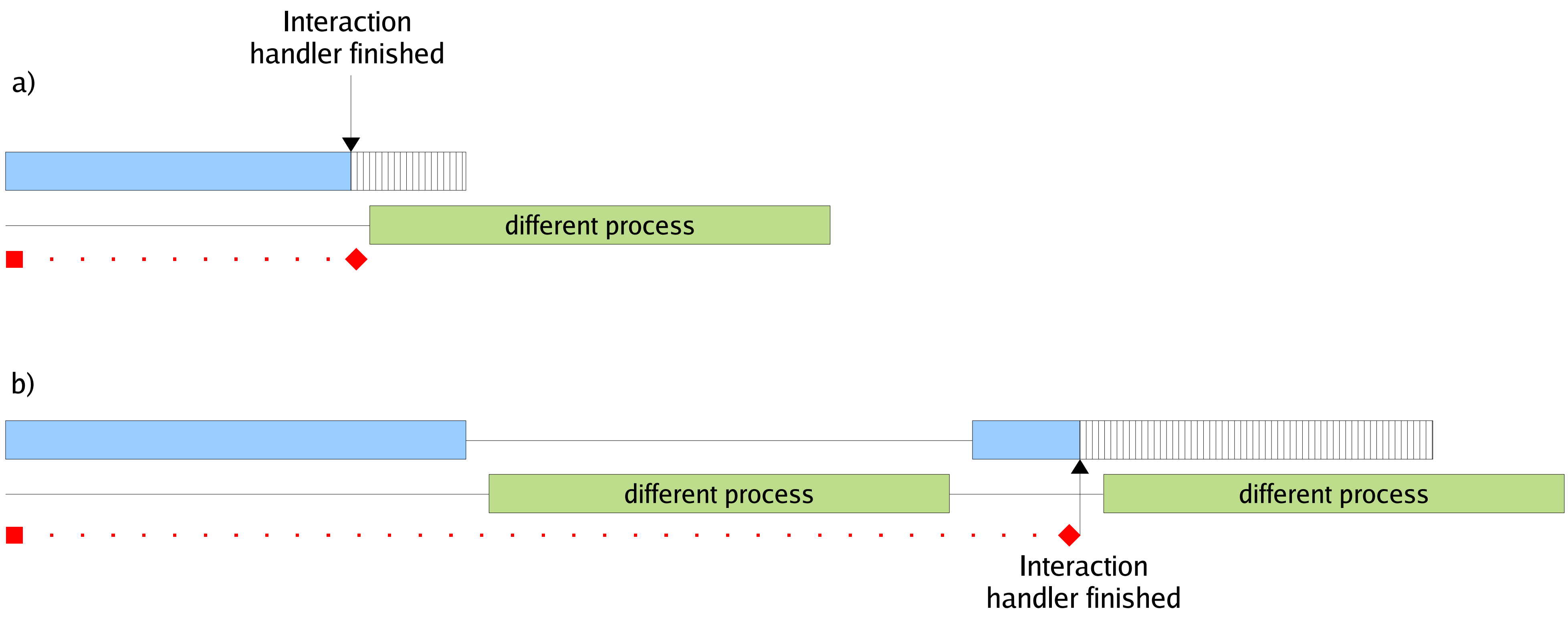
- All ready processes in one queue
- Assign to each thread a quantum (time slice)
- If process still active when time slice runs out:
  - preempt process, i.e. change its state to ready
  - Add process to queue's end
  - Activate next process in queue

Round Robin (2)

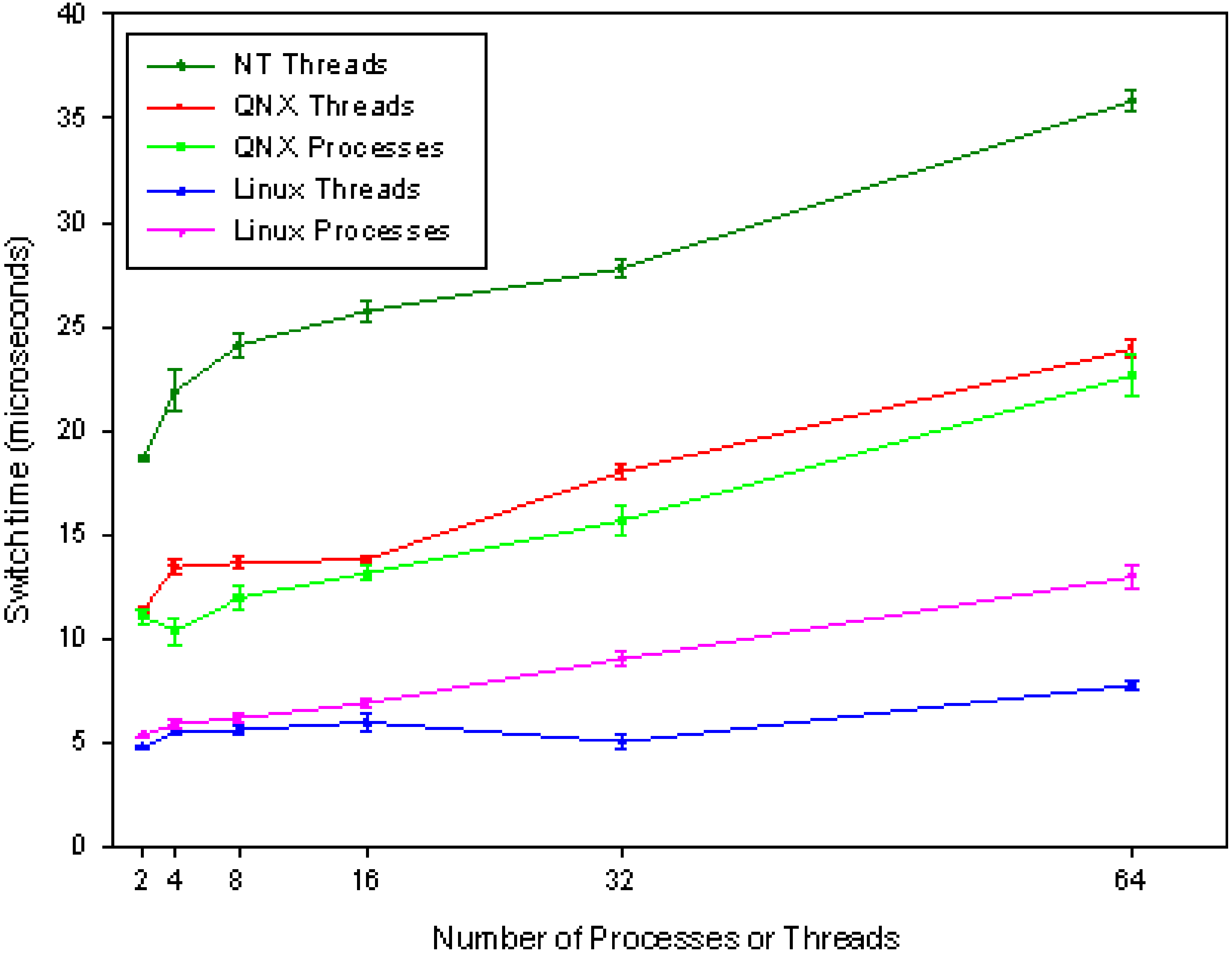
- Blocked processes that become ready, are also added at the end of the queue
- Criteria for choice of quantum length:
  - size must correspond to the time needed for a context switch
  - Large quantum: possibly long waiting times
  - Small quantum: short response times, but overhead because of frequent context switch

Round Robin (3)

- Often: Quantum *q* slightly larger than typical time needed for processing an interaction



Context Switch Latency



Pentium Pro
(200 MHz)

QNX 4.2
Red Hat Linux 5.1
Windows NT 4.0

Values from 1999

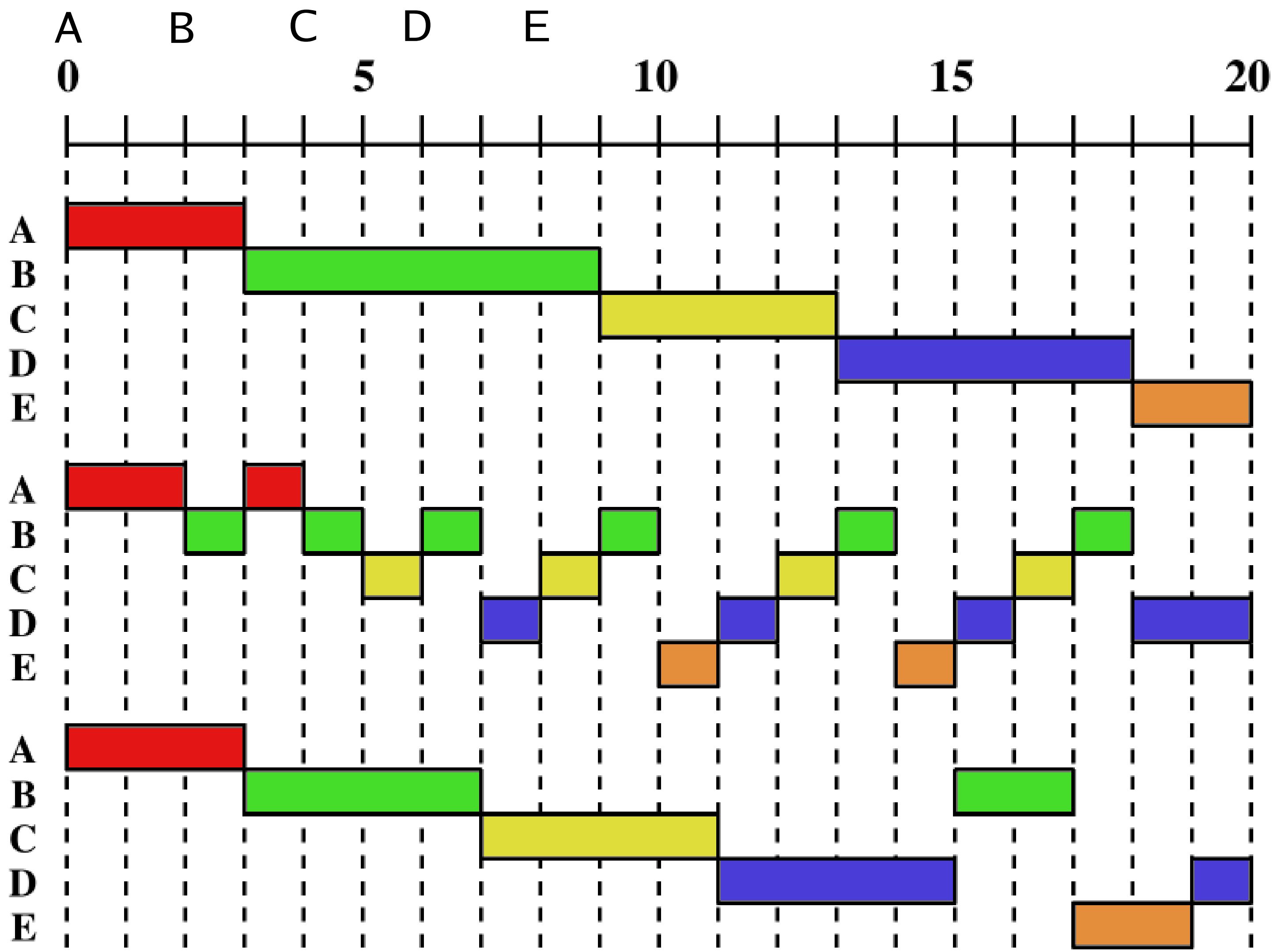
Round Robin Example

Arrival times:
A: 0, B: 2, C: 4,
D: 6, E: 8

For comparison:
First-Come-First Served (FCFS)

Round-Robin (RR), *q* = 1

Round-Robin (RR), *q* = 4



Picture: Stallings, S. 405

Virtual Round Robin (1)

- Round Robin unfair towards I/O-bound processes:
  - CPU-bound ones use full quantum,
  - I/O-bound ones only a fraction
- Idea: Remember unused quantum-part as process' credit
- Once the I/O process turns ready (I/O event occurred): use up remaining credit from last execution

Virtual Round Robin (2)

- Processes which spend their whold quantum are treated as in regular Round Robin:
  back to the queue
- Processes that block because of I/O and only used time $u < q$ of the quantum move to an auxiliary queue once they become ready
- Scheduler prefers processes in aux. queue
- Quantum for the process: $q-u$
  („gets what it deserves“, what was not used up the last time)

Virtual Round Robin

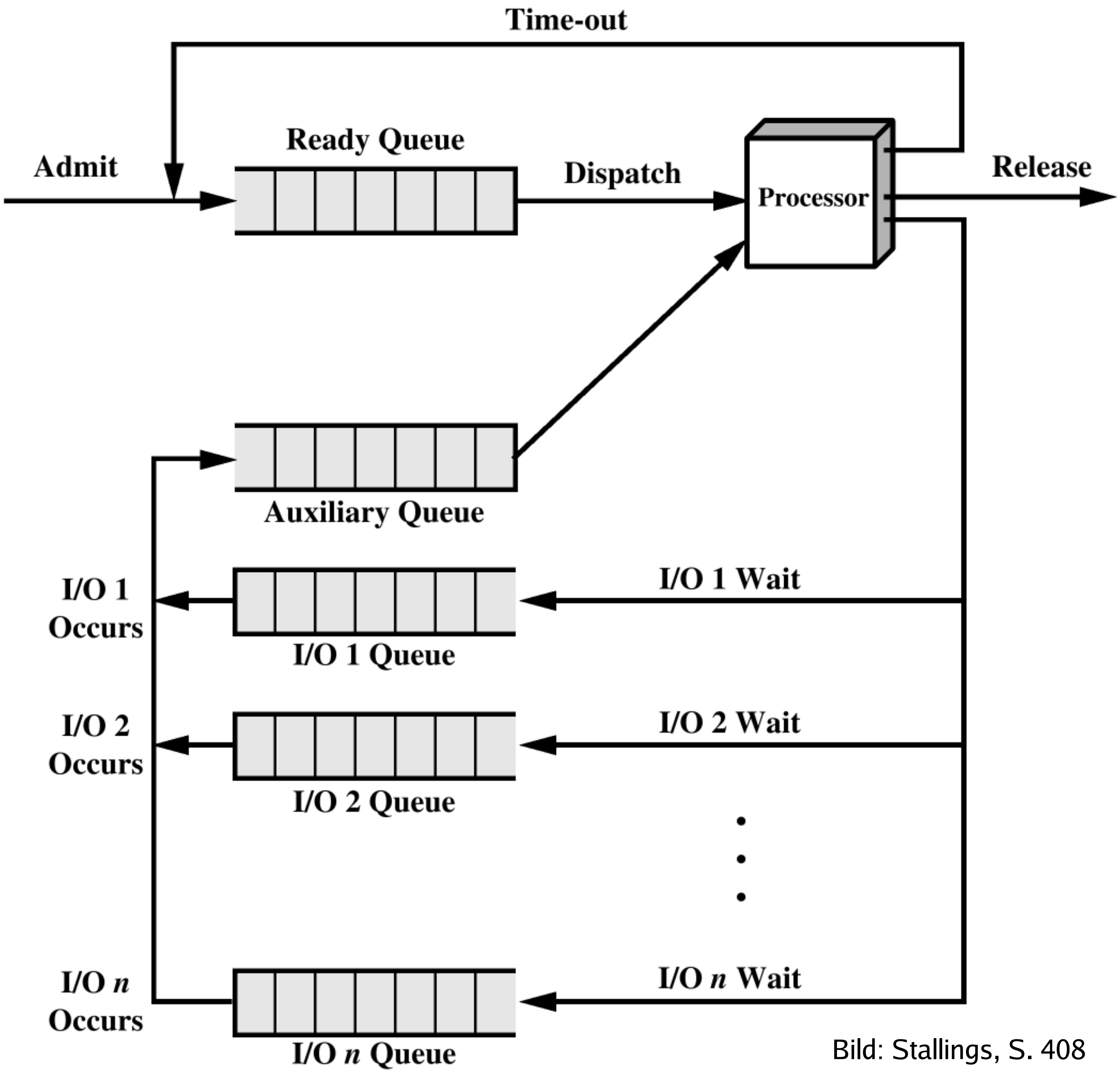


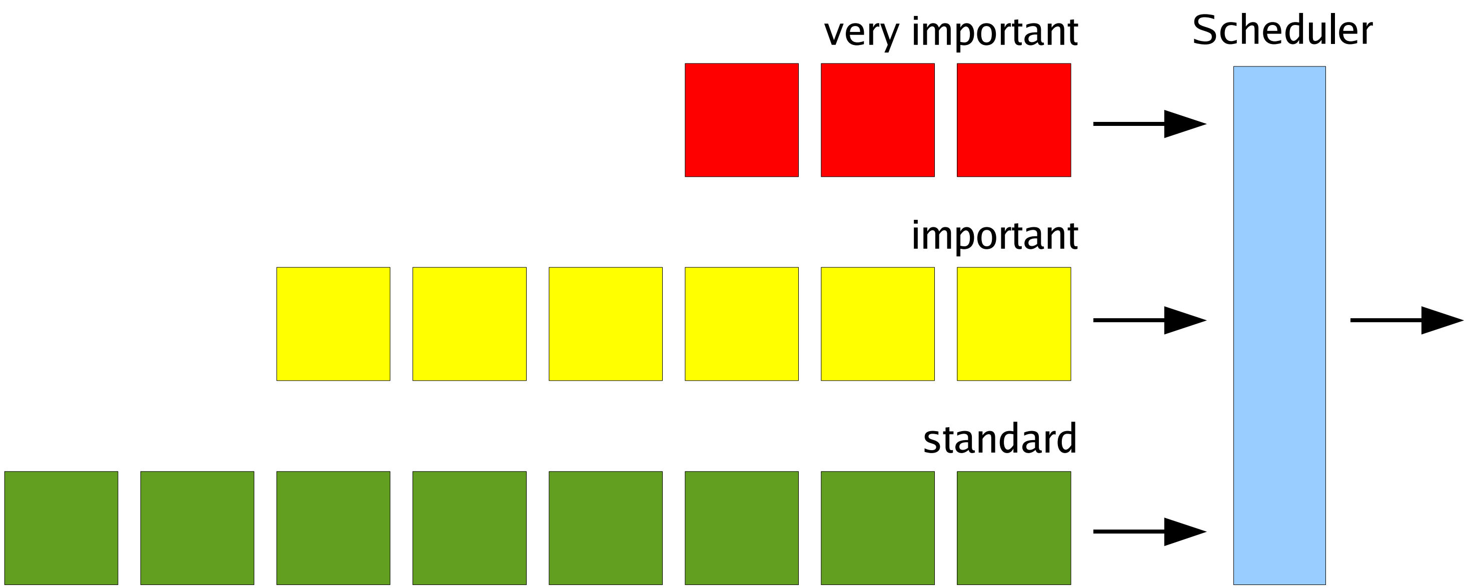
Bild: Stallings, S. 408

Priority Scheduler (1)

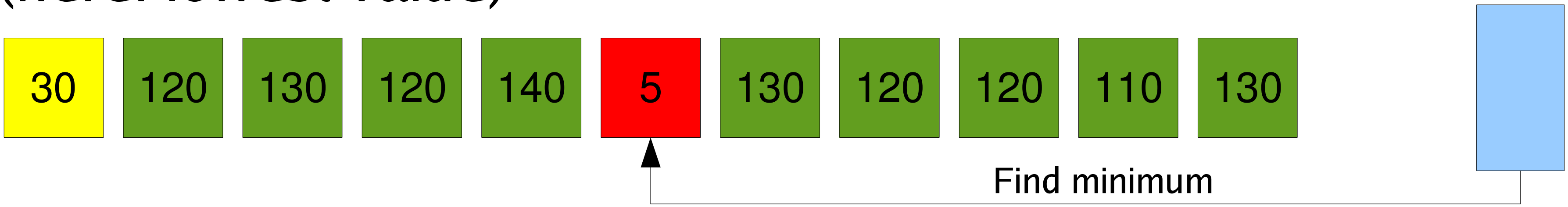
- Idea: either classify processes or assign a priority value to each process
- Scheduler prefers processes with higher priority (higher value or membership in higher class)
- assign at process creation (static) or let scheduler recalculate regularly (dynamic)
- Scheduling cooperative or preemptive

Priority Scheduler (2)

a) Several queues for priority classes



b) Scheduler searches for process with highest priority (here: lowest value)



Priority Scheduler (3)

Several queues

- Processes are assigned to priority classes and put into the corresponding queues
- Scheduler activates only processes in the highest non-empty queue
- Preemptive: stop process after a time quantum runs out
- Inside the queues: Round Robin

Priority Scheduler (3)

No hierarchies, but individual process priorities

- All processes in one process list
- Scheduler picks process with the highes priority value
- If several processes have the same (highest) priority, schedule them with Round Robin

Priority Scheduler (4)

Processes can starve ➞ Aging

Priority <u>inversion</u>:

- Process with high priority is blocked (needs a resource)
- Process with low priority holds this resource, but will not be activated by the scheduler (because there are other processes with higher priority)
- Both processes never run, since there are always processes of medium priority
- Solution: Aging

Priority Scheduler (5)

Aging:

- Permanently increase the priority of a process that is ready while it waits for the CPU
- Priorities of the active process and all not-ready (blocked) processes remain unchanged
- Result: Process that waits long will eventually gain sufficiently high priority to become active

Priority Scheduler (6)

Different quantum lengths

- Priority classes:
  1st priority = 1 quantum, 2nd priority = 2 quants,
  3rd priority = 4 quants, 4th priority = 8 quants
- Processes with high priority receive small quantum.
- If they give up the CPU before the quantum is used up, they stay in the priority class
- If they use up the quantum, scheduler will double the quant length and reduce priority – continue that, until process no longer uses up its quantum

Priority Scheduler Implementation

```
#!/usr/bin/python
# proc: [start, runtime, prio, used time]
processes = {
  1:[0,20,100,0],  2:[5,10,50,0],
  3:[6,30,100,0],  4:[7, 2,10,0] }
runtime = 40                # wie lange laufen lassen?

# Initialisierung
proccount = len (processes)
activity_log = []
seconds = 0

def find_min_priority():
  # Prozess mit der niedrigsten Prioritaet suchen
  minval = 9999
  for p in processes:
    # Test auf drei Bedingungen: Wert kleiner als Minimum,
    # Prozess schon erzeugt und noch nicht beendet
    [start,maxtime,prio,used] = processes[p]
    if prio < minval and start <= seconds and used < maxtime:
      minval = prio
      minproc = p
  return minproc
# end find_min_priority()
```

Priority Scheduler Implementation

```
def recalculate():
  # Prioritaeten neu berechnen
  processes[active][3] += 1   # Used-Time-Wert erhoehen
  return
# end recalculate()

def print_status():
  print "%4d |" % seconds,
  for p in processes:
    if p==active: st="*"
    else: st=" "
    [start,maxtime,prio,used] = processes[p]
    if (start <= seconds) and (used < maxtime):
      print "%2d/%2d %3d %s|" % (used, maxtime, prio, st),
    else: print "           |",
  return
# end print_status()
```

Priority Scheduler Implementation

```
# begin main()
print "Zeit |",
for p in processes:
  print "Prozess %2d |" % p,
print

for i in range(0,runtime):
  active = find_min_priority()
  print_status()
  print "-> Scheduler aktiviert P.",
    active
  activity_log.append(active)
  # aktiven Prozess ausfuehren
  seconds += 1  # Zeit hochzaehlen
  recalculate()

print
```

```
# Statistik ausgeben

print

for p in processes:
  [start,maxtime,prio,used] = \
    processes[p]
  st = ""  # leerer String
  print "Prozess %1d:" % p,
  for i in range(0,runtime):
    if   start>i:              st += " "
    elif activity_log[i]==p: st += "x"
    else:                      st += "-"
  print st

# end main()
```

Priority Scheduler Implementation

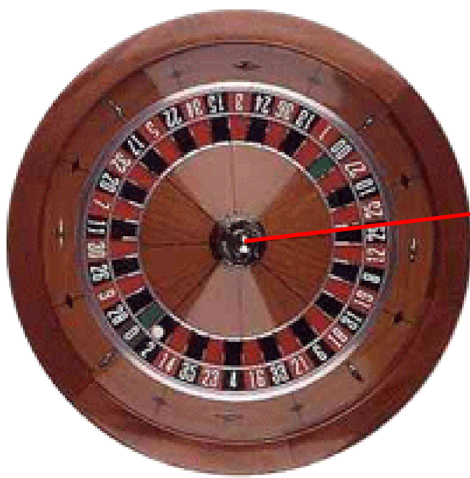
```
> ./prio-sched.py
Zeit | Prozess  1 | Prozess  2 | Prozess  3 | Prozess  4 |
   0 |  0/20 100 *|            |            |            | -> Scheduler aktiviert P. 1
   1 |  1/20 100 *|            |            |            | -> Scheduler aktiviert P. 1
   2 |  2/20 100 *|            |            |            | -> Scheduler aktiviert P. 1
   3 |  3/20 100 *|            |            |            | -> Scheduler aktiviert P. 1
   4 |  4/20 100 *|            |            |            | -> Scheduler aktiviert P. 1
   5 |  5/20 100  |  0/10  50 *|            |            | -> Scheduler aktiviert P. 2
   6 |  5/20 100  |  1/10  50 *|  0/30 100  |            | -> Scheduler aktiviert P. 2
   7 |  5/20 100  |  2/10  50  |  0/30 100  |  0/ 2  10 *| -> Scheduler aktiviert P. 4
   8 |  5/20 100  |  2/10  50  |  0/30 100  |  1/ 2  10 *| -> Scheduler aktiviert P. 4
   9 |  5/20 100  |  2/10  50 *|  0/30 100  |            | -> Scheduler aktiviert P. 2
  10 |  5/20 100  |  3/10  50 *|  0/30 100  |            | -> Scheduler aktiviert P. 2
...
  15 |  5/20 100  |  8/10  50 *|  0/30 100  |            | -> Scheduler aktiviert P. 2
  16 |  5/20 100  |  9/10  50 *|  0/30 100  |            | -> Scheduler aktiviert P. 2
  17 |  5/20 100 *|            |  0/30 100  |            | -> Scheduler aktiviert P. 1
  18 |  6/20 100 *|            |  0/30 100  |            | -> Scheduler aktiviert P. 1
...
  30 | 18/20 100 *|            |  0/30 100  |            | -> Scheduler aktiviert P. 1
  31 | 19/20 100 *|            |  0/30 100  |            | -> Scheduler aktiviert P. 1
  32 |            |            |  0/30 100 *|            | -> Scheduler aktiviert P. 3
  33 |            |            |  1/30 100 *|            | -> Scheduler aktiviert P. 3
...

Prozess 1: xxxxx------------xxxxxxxxxxxxxxx--------
Prozess 2:      xx--xxxxxxxx-----------------------
Prozess 3:       --------------------------xxxxxxxx
Prozess 4:        xx-------------------------------
```

Lottery Scheduler (1)

- Idea: processes receive „lottery tickets“ for access to resources
- Scheduler draws a ticket and activates the process which has the ticket
- Priorisation: Some processes receive more tickets than others

Lottery Scheduler (2)



Scheduler draws ticket No. 5

Process 1
Ticket 1,2,3,4

Process 2
Tickets 5,6

Process 3
Tickets 7,8,9

Process 4
Ticket 10

Lottery Scheduler (3)

- Groups and ticket sharing:
  - Collaboration of Client / Server
  - Client sends request to server, then gives its tickets to it and blocks
  - After handling the request, server returns tickets to the client and wakes it up
  - No clients?
    ➞ Server receives no tickets, never wins lottery (i.e. never becomes active)

Lottery Scheduler (4)

- Division of computing time only correct in a statistical sense
- In real-world situations different wait times occur
- The longer several processes run, the better (fairer) is the expected CPU sharing

Fair-Share Scheduling (1)

- Idea: Don't divide CPU times process-based, but on application or user level
- Normal schedulers view each process or thread separately
- Grouping of processes / threads
- Each computer user receives his „fair share“ of computing time
- is used in some Unix systems

Fair-Share Scheduling (2)

Algorithm by G. Henry (1984):

Process j in group k

$$\left.\begin{array}{l} CPU_j(i) = \dfrac{CPU_j(i-1)}{2} \\[2ex] GCPU_k(i) = \dfrac{GCPU_k(i-1)}{2} \end{array}\right\} P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4W_k}$$

- $CPU_j(i)$ CPU utilization by process j in intervall i
- $GCPU_k(i)$ CPU utilization by group k in intervall i
- $P_j(i)$ Priority of process j at the beginning of intervall i (low value = high priority)
- $Base_j$ Base priority of process j
- W_k Weight for group k; $0 < W_k \leq 1 ; \sum_k W_k = 1$

Fair-Share Scheduling (3)

Zeit	GROUP 1			GROUP 2					
	A			B			C		
	Priority	process CPU utiliz.	group CPU utiliz.	Priority	process CPU utiliz.	group CPU utiliz.	Priority	process CPU utiliz.	group CPU utiliz.
0	60	0 1 2 ... 60	0 1 2 ... 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 ... 60	0 1 2 ... 60	60	0	0
2	74	15 16 17 ... 75	15 16 17 ... 75	90	30	30	75	0	30

Fair-Share Scheduling (4)

Zeit	GROUP 1			GROUP 2					
	A			B			C		
	Priority	process CPU utiliz.	group CPU utiliz.	Priority	process CPU utiliz.	group CPU utiliz.	Priority	process CPU utiliz.	group CPU utiliz.
3	96	37	37	74	15	15 16 17 ... 75	67	0 1 2 ... 60	15 16 17 ... 75
4	78	18 19 20 ... 78	18 19 20 ... 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18