A hierarchical CPU scheduler for multimedia operating systems was proposed in [142]. The basic idea

of the start-time fair queuing (SFQ) algorithm is to organize the consumers of the CPU bandwidth in a

tree structure; the root node is the processor and the leaves of this tree are the threads of each application.

A scheduler acts at each level of the hierarchy. The fraction of the processor bandwidth, B, allocated to

the intermediate node i is

Bi

B

= wi

nj

=1 wj

(6.31)

with wj , 1   j   n, the weight of the n children of node i ; see the example in Figure 6.9.

When a virtual machine is not active, its bandwidth is reallocated to the other VMs active at the time.

When one of the applications of a virtual machine is not active, its allocation is transferred to the other applications running on the same VM. Similarly, if one of the threads of an application is not runnable,

its allocation is transferred to the other threads of the applications.

Call va(t) and vb(t) the virtual time of threads a and b, respectively, at real time t. The virtual time

of the scheduler at time t is denoted by v(t). Call q the time quantum of the scheduler in milliseconds.

The threads a and b have their time quanta, qa and qb, weighted by wa and wb, respectively; thus, in

our example, the time quanta of the two threads are q/wa and q/wb, respectively. The i -th activation

of thread a will start at the virtual time Sia

and will finish at virtual time Fia

. We call τ j the real time of

the j -th invocation of the scheduler.

An SFQ scheduler follows several rules:

R1. The threads are serviced in the order of their virtual start-up time; ties are broken arbitrarily.

R2. The virtual startup time of the i -th activation of thread x is

Six

(t) = max

v

τ j

, F(i−1)

x (t)

and S0

x

= 0. (6.32)

The condition for thread i to be started is that thread (i − 1) has finished and that the scheduler is

active.

R3. The virtual finish time of the i -th activation of thread x is

Fix

(t) = Six

(t) + q

wx

. (6.33)

A thread is stopped when its time quantum has expired; its time quantum is the time quantum of

the scheduler divided by the weight of the thread.

R4. The virtual time of all threads is initially zero, v0

x

= 0. The virtual time v(t) at real time t is

computed as follows:

v(t) =

Virtual start time of the thread in service at time t, if CPU is busy

Maximum finish virtual time of any thread, if CPU is idle.

(6.34)

In this description of the algorithm we have included the real time t to stress the dependence of all

events in virtual time on the real time. To simplify the notation we use in our examples the real time as

the index of the event. In other words, S6

a means the virtual start-up time of thread a at real time t = 6.

Example. The following example illustrates the application of the SFQ algorithm when there are two

threads with the weights wa = 1 and wb = 4 and the time quantum is q = 12 (see Figure 6.10.)

Initially S0

a

= 0, S0

b

= 0, va(0) = 0, and vb(0) = 0. Thread b blocks at time t = 24 and wakes up

at time t = 60.

The scheduling decisions are made as follows:

1. t = 0: We have a tie, S0

a

= S0

b , and arbitrarily thread b is chosen to run first. The virtual finish time

of thread b is

F0

b

= S0

b

+ q/wb = 0 + 12/4 = 3. (6.35)

2. t = 3: Both threads are runnable and thread b was in service; thus, v(3) = S0

b

= 0; then

S1

b

= max[v(3), F0

b

] = max (0, 3) = 3. (6.36)

But S0

a < S1

b , thus thread a is selected to run. Its virtual finish time is

F0

a

= S0

a

+ q/wa = 0 + 12/1 = 12. (6.37)

3. t = 15: Both threads are runnable, and thread a was in service at this time; thus,

v(15) = S0

a

= 0 (6.38)

and

S1

a

= max[v(15), F0

a

] = max[0, 12] = 12. (6.39)

As S1

b

= 3 < 12, thread b is selected to run; the virtual finish time of thread b is now

F1

b

= S1

b

+ q/wb = 3 + 12/4 = 6. (6.40)

4. t = 18: Both threads are runnable, and thread b was in service at this time; thus,

v(18) = S1

b

= 3 (6.41)

and

S2

b

= max[v(18), F1

b

] = max[3, 6] = 6. (6.42)

As S2

b < S1

a

= 12, thread b is selected to run again; its virtual finish time is

F2

b

= S2

b

+ q/wb = 6 + 12/4 = 9. (6.43)

5. t = 21: Both threads are runnable, and thread b was in service at this time; thus,

v(21) = S2

b

= 6 (6.44)

and

S3

b

= max[v(21), F2

b

] = max[6, 9] = 9. (6.45)

As S2

b < S1

a

= 12, thread b is selected to run again; its virtual finish time is

F3

b

= S3

b

+ q/wb = 9 + 12/4 = 12. (6.46)

6. t = 24: Thread b was in service at this time; thus,

v(24) = S3

b

= 9 (6.47)

S4

b

= max[v(24), F3

b

] = max[9, 12] = 12. (6.48)

Thread b is suspended till t = 60; thus, thread a is activated. Its virtual finish time is

F1

a

= S1

a

+ q/wa = 12 + 12/1 = 24. (6.49)

7. t = 36: Thread a was in service and the only runnable thread at this time; thus,

v(36) = S1

a

= 12 (6.50)

and

S2

a

= max[v(36), F2

a

] = max[12, 24] = 24. (6.51)

Then,

F2

a

= S2

a

+ q/wa = 24 + 12/1 = 36. (6.52)

8. t = 48: Thread a was in service and is the only runnable thread at this time; thus,

v(48) = S2

a

= 24 (6.53)

and

S3

a

= max[v(48), F2

a

] = max[24, 36] = 36. (6.54)

Then,

F3

a

= S3

a

+ q/wa = 36 + 12/1 = 48. (6.55)

9. t = 60: Thread a was in service at this time; thus,

v(60) = S3

a

= 36 (6.56)

and

S4

a

= max[v(60), F3

a

] = max[36, 48] = 48. (6.57)

But now thread b is runnable and S4

b

= 12.

Thus, thread b is activated and

F4

b

= S4

b

+ q/wb = 12 + 12/4 = 15. (6.58)

Several properties of the SFQ algorithm are proved in [142]. The algorithm allocates CPU fairly

when the available bandwidth varies in time and provides throughput as well as delay guarantees. The

algorithm schedules the threads in the order of their virtual start-up time, the shortest one first; the length

of the time quantum is not required when a thread is scheduled but only after the thread has finished its

current allocation. The authors of [142] report that the overhead of the SFQ algorithms is comparable

to that of the Solaris scheduling algorithm.