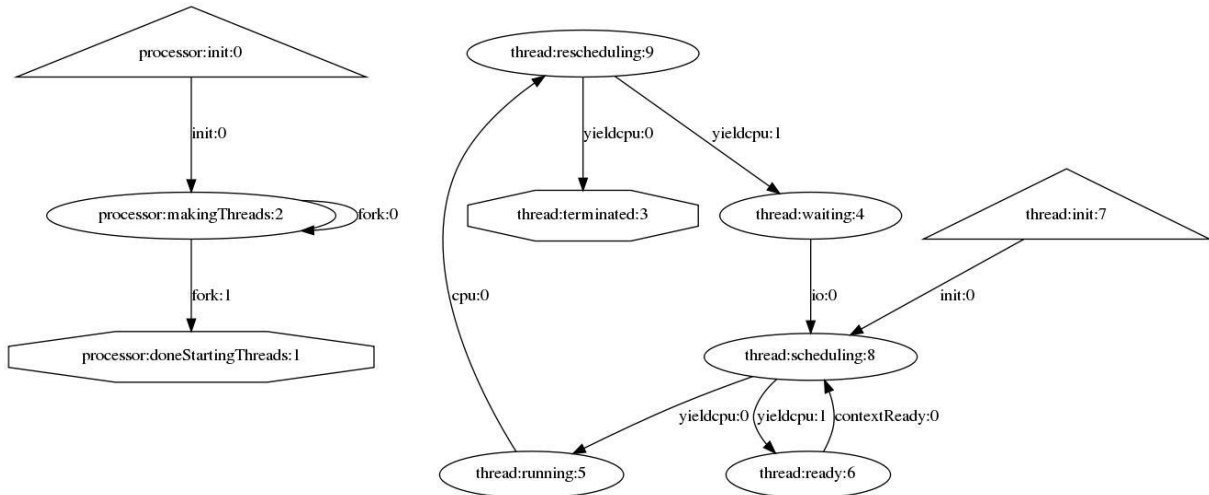


CSC 343 Operating Systems, Dr. Dale Parson, Fall 2014
State Machine Models of CPU Scheduling Algorithms
Located in ~parson/OpSys/state2codeV10

<http://acad.kutztown.edu/~parson/fcfs.jpg> (Substitute sjf or rr for fcfs)



fcfs.stm – First Come, First Served non-preemptive context scheduling

```

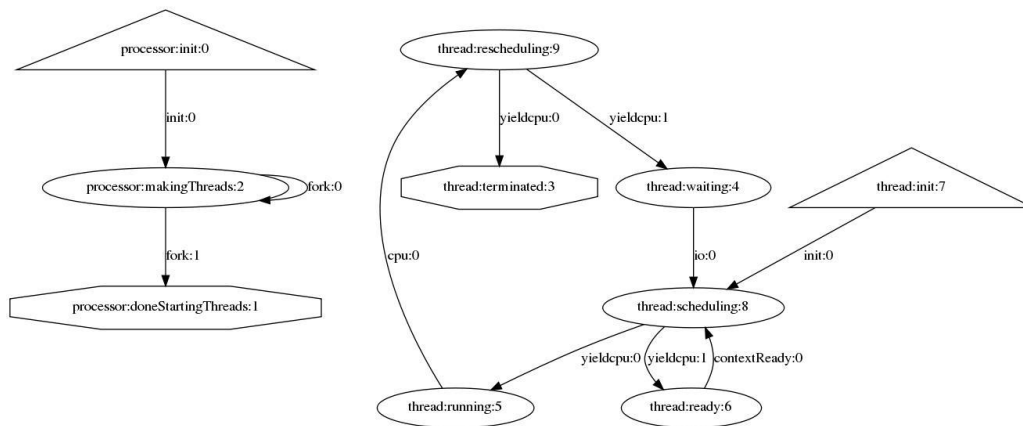
1  # CSC 343, Fall 2013, STUDENT NAME:
2  # fcfs.stm implements a first-come, first-served, non-preemptive scheduler as an
3  # example for assignment 2. D. Parson.
4
5  machine processor {
6    # Use this machine in all of your files in assignment 2 to start threads.
7    # It starts 10 threads, one every tick. I am starting them quickly so that
8    # algorithms like FCFS don't get swamped too much an early CPU-bound thread.
9    threadsToGo = 10 ;
10   start init, state makingThreads, accept doneStartingThreads ;
11   init -> makingThreads init()/@
12     processor.readyq = Queue(ispriority=False);
13     threadsToGo -= 1 ; fork()@
14   makingThreads -> makingThreads fork(pid, tid)[@threadsToGo > 0@]/@
15     threadsToGo -= 1 ; fork()@
16   makingThreads -> doneStartingThreads fork(pid, tid)[@threadsToGo == 0@]/
17 }
18
19 # For all parts of your assignment, half of every ten threads must be IO bound.
20 # The others are CPU bound. The I/O bound threads must request between
21 # 1 and 250 ticks using the exponential sampler, with the knee of the curve
22 # (half of the sampled values) at 25 ticks. The CPU bound threads must
23 # request between 100 and 1100 ticks using the revexponential sampler,
24 # with the knee of the curve (half of the sampled values) at 1000 ticks.
25 # All threads must continue to run until simulation time is >= 100,000.
26 machine thread {
27   machineid = -1, pid = -1, tid = -1, iobound = @False@, endtime = 100000 ;
28   # Python treats 0 as False (not iobound) and 1 as True (iobound).
29   # The transition out of state init initializes the above variables.
30   start init, state scheduling, state ready, state running, state waiting,

```

```

31     state rescheduling, accept terminated ;
32     init -> scheduling init()[]/@machineid, pid, tid = getpid();
33     iobound = True if ((pid % 2) == 1) else False ; yieldcpu()@
34     # ^^ The odd pids are IO bound.
35     # The others (50%) are CPU bound. This job mix stresses the scheduling
36     # algorithms better than a strictly IO-bound or CPU-bound mix.
37     scheduling -> running yieldcpu()[]/@processor.contextsFree > 0@]/@
38     processor.contextsFree -= 1 ;
39     ticks = sample(1, 250, 'exponential', 25) if iobound
40     else sample(100, 1100, 'revexponential', 1000);
41     msg('pid ' + str(pid) + ' tid ' + str(tid)
42     + ' about to CPU for ' + str(ticks) + ' ticks');
43     cpu(ticks)@
44     scheduling -> ready yieldcpu()[]/@processor.contextsFree == 0@]/@
45     # Put myself in processor's readyq with FIFO priority.
46     msg('pid ' + str(pid) + ' tid ' + str(tid)
47     + ' about to wait, ready for CPU');
48     processor.readyq.enq(thread); waitforEvent('contextReady', False)@
49     ready -> scheduling contextReady()[]/@yieldcpu()@
50     running -> rescheduling cpu()[]/@
51     processor.contextsFree += 1 ;
52     msg('thread ' + str(tid) + ' checking readyq '
53     + str(processor.readyq.len())
54     + ' with contextsFree ' + str(processor.contextsFree));
55     signalEvent(processor.readyq.deq(), 'contextReady')
56     if len(processor.readyq) > 0 else noop();
57     yieldcpu()@
58     rescheduling -> terminated yieldcpu()[]/@time() >= endtime@]/
59     rescheduling -> waiting yieldcpu()[]/@time() < endtime@]/@
60     # iodevice of -1 (process terminal) or one of the fastio devices.
61     iodevice = sample(-1, len(processor.fastio)-1, 'uniform');
62     msg('thread ' + str(tid) + ' blocking on IO unit ' + str(iodevice));
63     msg('pid ' + str(pid) + ' tid ' + str(tid)
64     + ' about to IO on dev ' + str(iodevice));
65     io(iodevice)@
66     waiting -> scheduling io()[]/@yieldcpu()@
67 }
68
69 processor

```



sjf.stm - non-preemptive Shortest Job First

```

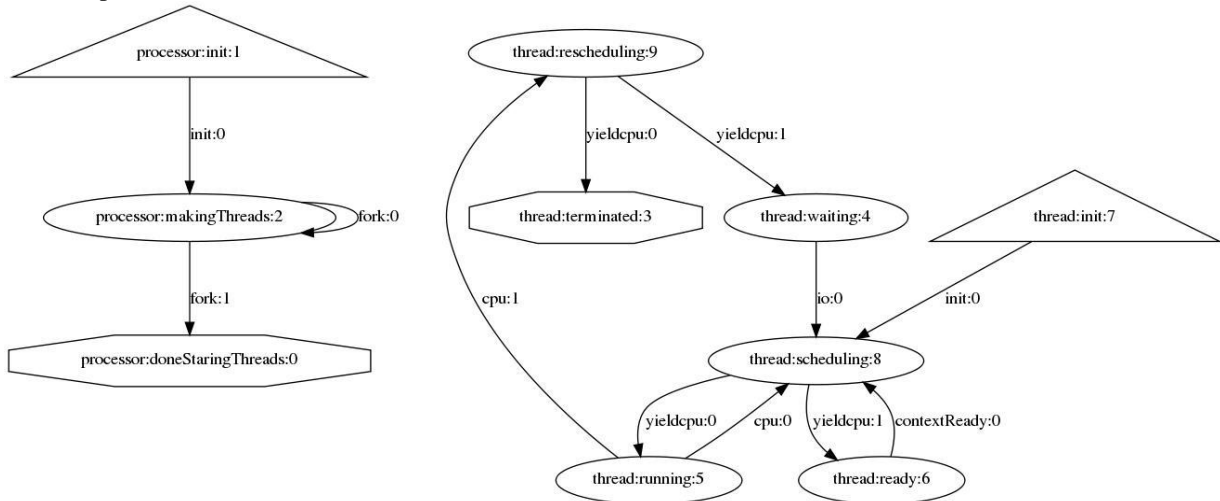
1 # CSC 343, Fall 2013, STUDENT NAME:
2 # sjf.stm implements a short-job first, non-preemptive scheduler as a
3 # partial solution of assignment 2. D. Parson.
4
5 machine processor {
6     # Use this machine in all of your files in assignment 2 to start threads.
7     # It starts 10 threads, one every tick. I am starting them quickly so that
8     # algorithms like FCFS don't get swamped too much an early CPU-bound thread.
9     threadsToGo = 10 ;
10    start init, state makingThreads, accept doneStartingThreads ;
11    init -> makingThreads init()[]/@
12        processor.readyq = Queue(ispriority=True);
13        threadsToGo -= 1 ; fork()@
14    makingThreads -> makingThreads fork(pid, tid)[@threadsToGo > 0@]/@
15        threadsToGo -= 1 ; fork()@
16    makingThreads -> doneStartingThreads fork(pid, tid)[@threadsToGo == 0@]/
17 }
18
19 # For all parts of your assignment, half of every ten threads must be IO bound.
20 # The others are CPU bound. The I/O bound threads must request between
21 # 1 and 250 ticks using the exponential sampler, with the knee of the curve
22 # (half of the sampled values) at 25 ticks. The CPU bound threads must
23 # request between 100 and 1100 ticks using the revexponential sampler,
24 # with the knee of the curve (half of the sampled values) at 1000 ticks.
25 # All threads must continue to run until simulation time is >= 100,000.
26 machine thread {
27     machineid = -1, pid = -1, tid = -1, iobound = @False@, endtime = 100000 ;
28     # Python treats 0 as False (not iobound) and 1 as True (iobound).
29     # The transition out of state init initializes the above variables.
30     start init, state scheduling, state ready, state running, state waiting,
31         state rescheduling, accept terminated ;
32     init -> scheduling init()[]/@machineid, pid, tid = getpid();
33         iobound = True if ((pid % 2) == 1) else False ;
34         # ^^^ The odd pids are IO bound.
35         # The others (50%) are CPU bound. This job mix stresses the scheduling
36         # algorithms better than a strictly IO-bound or CPU-bound mix.
37         # Set ticks when going into scheduling.
38         ticks = sample(1, 250, 'exponential', 25) if iobound
39             else sample(100, 1100, 'revexponential', 1000);
40         yieldcpu()@
41     scheduling -> running yieldcpu()[@processor.contextsFree > 0@]/@
42         processor.contextsFree -= 1 ;
43         msg('pid ' + str(pid) + ' tid ' + str(tid) + ' about to CPU for '
44             + str(ticks) + ' ticks'); cpu(ticks)@
45     scheduling -> ready yieldcpu()[@processor.contextsFree == 0@]/@
46         # Put myself in processor's readyq with sjf priority.
47         msg('pid ' + str(pid) + ' tid ' + str(tid)
48             + ' about to wait, ready for CPU ticks ' + str(ticks));
49         processor.readyq.enq(thread, ticks); waitForEvent('contextReady', False)@
50     ready -> scheduling contextReady()[]/@yieldcpu()@
51     # ^^^ Do not set ticks coming out of ready, not used yet.
52     running -> rescheduling cpu()[]/@
53         processor.contextsFree += 1 ;

```

```

54     msg('thread ' + str(tid) + ' checking readyq '
55         + str(len(processor.readyq))
56         + ' with contextsFree ' + str(processor.contextsFree));
57     signalEvent(processor.readyq.deq(), 'contextReady')
58     if len(processor.readyq) > 0 else noop();
59     yieldcpu(@
60     rescheduling -> terminated yieldcpu()[@time() >= endtime@]/
61     rescheduling -> waiting yieldcpu()[@time() < endtime@]/@
62     # iodevice of -1 (process terminal) or one of the fastio devices.
63     iodevice = sample(-1, len(processor.fastio)-1, 'uniform');
64     msg('thread ' + str(tid) + ' blocking on IO unit ' + str(iodevice));
65     msg('pid ' + str(pid) + ' tid ' + str(tid) + ' about to IO on dev '
66         + str(iodevice));
67     io(iodevice)@
68     waiting -> scheduling io()[]/@
69     ticks = sample(1, 250, 'exponential', 25) if iobound
70         else sample(100, 1100, 'revexponential', 1000);
71     yieldcpu(@
72 }
74 processor

```



rr.stm – Round Robin Preemptive Scheduling

```

1  # CSC 343, Fall 2013, STUDENT NAME:
2  # rr.stm implements a preemptive round-robin scheduler as a
3  # partial solution of assignment 2. D. Parson.
4  machine processor {
5      # Use this machine in all of your files in assignment 2 to start threads.
6      # It starts 10 threads, one every tick. I am starting them quickly so that
7      # algorithms like FCFS don't get swamped too much an early CPU-bound thread.
8      threadsToGo = 10 ;
9      start init, state makingThreads, accept doneStaringThreads ;
10     init -> makingThreads init()[]/@
11         processor.readyq = Queue(ispriority=False);
12         threadsToGo -- 1 ; fork()@
13     makingThreads -> makingThreads fork(pid, tid)[@threadsToGo > 0@]/@
14         threadsToGo -- 1 ; fork()@
15     makingThreads -> doneStaringThreads fork(pid, tid)[@threadsToGo == 0@]/
16 }
17 }

```

```

19 # For all parts of your assignment, half of every ten threads must be IO bound.
20 # The others are CPU bound. The I/O bound threads must request between
21 # 1 and 250 ticks using the exponential sampler, with the knee of the curve
22 # (half of the sampled values) at 25 ticks. The CPU bound threads must
23 # request between 100 and 1100 ticks using the revexponential sampler,
24 # with the knee of the curve (half of the sampled values) at 1000 ticks.
25 # All threads must continue to run until simulation time is >= 100,000.
26 # STUDENT: The quantum must be 125 ticks. Make sure that a thread never
27 # calls cpu() with more than quantum ticks; use Python's min(a, b) function.
28 # Make sure to keep any un-run ticks returns from sample() in a variable,
29 # and make sure that as long as the remaining ticks from the most
30 # recent sample() have not reached 0, that your thread gets back to the
31 # ready state (processor.readyq.enq) WITHOUT doing io(). It should request
32 # io() EXACTLY at the point that it has consumed all ticks supplied
33 # by the most recent sample() call, after which it can sample() a new
34 # CPU-burst number of ticks. Any given cpu() call must NEVER exceed
35 # the quantum limit.
36 machine thread {
37     quantum = 125, machineid = -1, pid = -1, tid = -1, iobound = @False@,
38     endtime = 100000 ;
39     # Python treats 0 as False (not iobound) and 1 as True (iobound).
40     # The transition out of state init initializes the above variables.
41     start init, state scheduling, state ready, state running, state waiting,
42     state rescheduling, accept terminated ;
43     init -> scheduling init()[]/@machineid, pid, tid = getpid();
44     iobound = True if ((pid % 2) == 1) else False ;
45     # ^^ The odd pids are IO bound.
46     # The others (50%) are CPU bound. This job mix stresses the scheduling
47     # algorithms better than a strictly IO-bound or CPU-bound mix.
48     # Set ticks when going into scheduling.
49     ticks = sample(1, 250, 'exponential', 25) if iobound
50     else sample(100, 1100, 'revexponential', 1000);
51     tickstorun = min(ticks, quantum);
52     tickstodefer = ticks - tickstorun;
53     yieldcpu()@
54     # ^^ pids that give a remainder of 5 for divide-by-10 are CPU bound.
55     scheduling -> running yieldcpu()[@processor.contextsFree > 0@]/@
56     processor.contextsFree -= 1 ;
57     msg('pid ' + str(pid) + ' tid ' + str(tid) + ' about to CPU for '
58     + str(tickstorun) + ' tickstorun ' + ' out of ' + str(ticks)
59     + ' ticks, tickstodefer = ' + str(tickstodefer));
60     cpu(tickstorun)@
61     scheduling -> ready yieldcpu()[@processor.contextsFree == 0@]/@
62     # Put myself in processor's readyq with rr priority.
63     msg('pid ' + str(pid) + ' tid ' + str(tid)
64     + ' about to wait, ready for CPU tickstorun ' + str(tickstorun)
65     + ' out of ' + str(ticks) + ' ticks, tickstodefer = '
66     + str(tickstodefer));
67     processor.readyq.enq(thread); waitForEvent('contextReady', False)@
68     ready -> scheduling contextReady()[]/@yieldcpu()@
69     # ^^ Do not set ticks; they have not all been used.
70     running -> scheduling cpu()[@tickstodefer > 0@]/@
71     processor.contextsFree += 1 ;
72     signalEvent(processor.readyq.deq(), 'contextReady')
73     if len(processor.readyq) > 0 else noop();
74     tickstorun = min(tickstodefer, quantum);

```

```

75     tickstodefer = tickstodefer - tickstorun;
76     yieldcpu()@
77     running -> rescheduling cpu()[@tickstodefer < 1@]/@
78     processor.contextsFree += 1 ;
79     signalEvent(processor.readyq.deq(), 'contextReady')
80     if len(processor.readyq) > 0 else noop();
81     yieldcpu()@
82     rescheduling -> terminated yieldcpu()[@time() >= endtime@]/
83     rescheduling -> waiting yieldcpu()[]/@
84     # Pick an iodevice of -1 (process terminal) or one of the fastio devices.
85     iodevice = sample(-1, len(processor.fastio)-1, 'uniform');
86     msg('thread ' + str(tid) + ' blocking on IO unit ' + str(iodevice));
87     msg('pid ' + str(pid) + ' tid ' + str(tid) + ' about to IO on dev '
88         + str(iodevice));
89     io(iodevice)@
90     waiting -> scheduling io()[]/@
91     ticks = sample(1, 250, 'exponential', 25) if iobound
92     else sample(100, 1100, 'revexponential', 1000);
93     tickstorun = min(ticks, quantum);
94     tickstodefer = ticks - tickstorun;
95     yieldcpu()@
96 }
97
98 processor

```