ECS150 Discussion Section

Sophie Engle

(*January 28/30 2004*)

Process Scheduling

- **Introduction & Terminology**
  - Types of Schedulers
  - Scheduling Considerations & Performance
  - Algorithm Characteristics
- **Algorithms**
  - Shortest Job First
  - Highest Response Ratio Next
  - Selfish Round Robin
  - Multi-level Feedback
- **Evaluation**
  - Little's Law

Resources

- Some scheduling notes online from previous ECS150 course
  - http://nob.cs.ucdavis.edu/classes/ecs150-2000-winter/Pdf/scheduling.pdf

Process Scheduling

Boring Stuff

(Terminology)

Types of Schedulers

- Long-term Scheduler
  - Determines which jobs are admitted to the system for processing
- Medium-term Scheduler
  - When too many processes competing for memory, determines which get swapped in/out
- Short-term Scheduler*
  - Determines which process in memory (in ready queue) goes next

Scheduling Considerations

- What is the goal of a scheduler?
  - Throughput
  - Turnaround
  - Response
  - Resource use
  - Waiting time
  - Consistency
- Examples?

Scheduling Considerations

- What is the goal of a scheduler?
  - Throughput – work done in a given time
  - Turnaround – time to completion
  - Response – time from submission to response
  - Resource use – # of resources, waiting time
  - Waiting time – time process in ready queue
  - Consistency – runtime predictability
- Examples?

Scheduling Performance

- How measure scheduling performance?
  - Turnaround time (T)
    - Time process present in system
  - Waiting time (W)
    - Time process present and not running
  - Response ratio (R), Penalty ratio (P)
    - Factor by which processing rate reduced

Scheduling Performance

- How measure scheduling performance?
  - Turnaround time (T)
    - T = [finish time] – [arrival time]
  - Waiting time (W)
    - W = T – [service time]
  - Response ratio (R)
    - $R = \dfrac{T}{\text{service time}}$

Algorithm Characteristics

- Decision mode
  - Non-preemptive
    - A process runs until it blocks are completes (runs until no longer ready)
  - Preemptive
    - Operating system can interrupt currently running process to start another one

Algorithm Characteristics

- Priority function, p(a, r, t)
  - Assigns a priority to a process
  - Usually involves
    - a: service time so far
    - r: real time spent in system so far
    - t: total required service time
- Arbitration rule
  - Resolves ties when two processes have equal priority

Process Scheduling

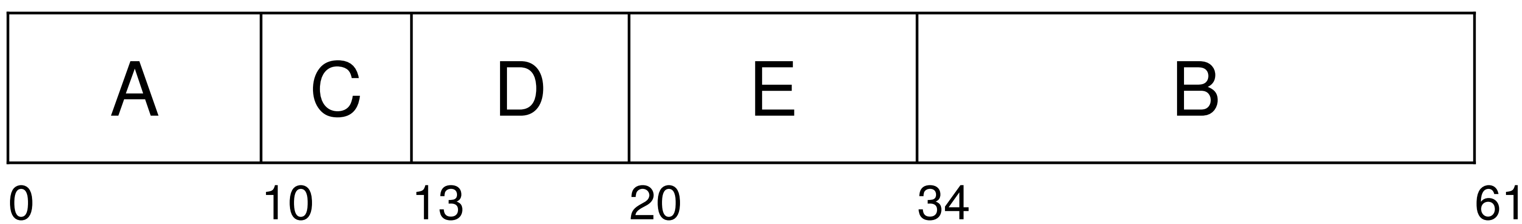
Scheduling Algorithms

Shortest Job Next

- **S**hortest **J**ob **N**ext, **F**irst (SJN, SJF)
  - Decision mode: non-preemptive
  - Arbitration rule: chronological or random
  - Priority function: p(a, r, t) = -t

Shortest Job Next

	Ready queue				
Process	**A**	**B**	**C**	**D**	**E**
Arrival time	0	1	2	3	4
Service time	10	29	3	7	12



For process D:

```
Turnaround Time (T) = 20 - 3 = 17
   Waiting Time (W) = 17 - 7 = 10
 Response Ratio (R) = 17 / 7 ≈ 2.3
```

Shortest Job Next

- Pro:
  - Gives smallest average turnaround time T out of all non-preemptive priority functions
- Con:
  - Need to know service time before process runs
  - No way to know service time without running the process!

Shortest Job Next

- Solution:
  - Compute expected time of next CPU-burst as an *exponential average* of previous bursts of process

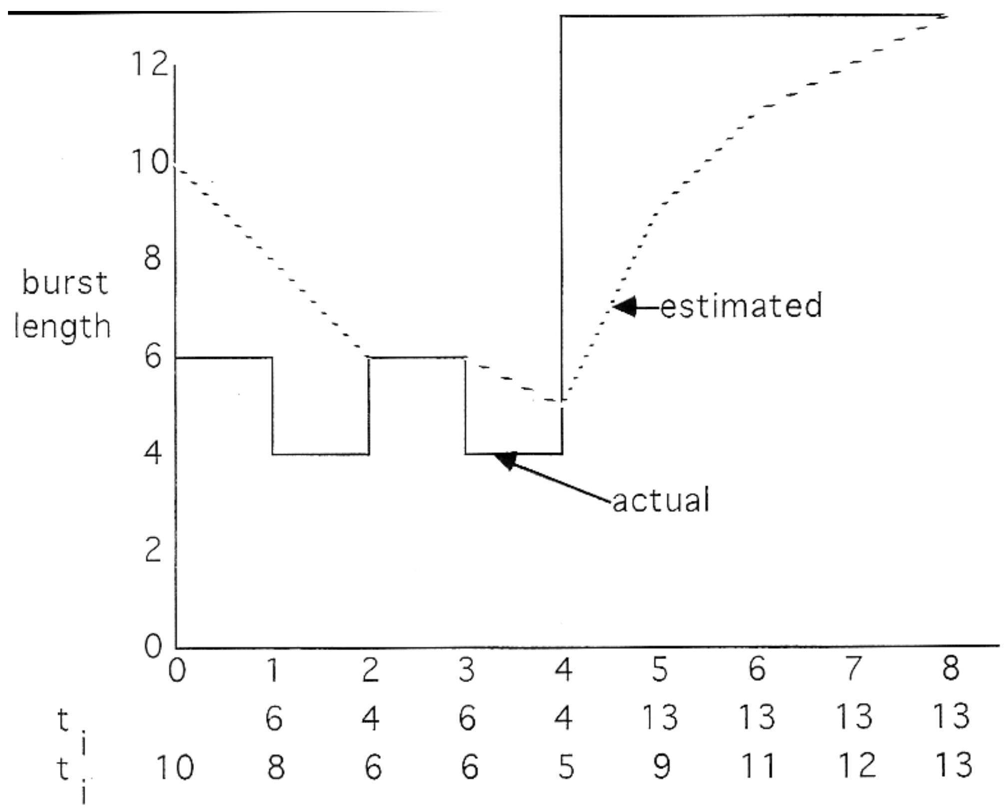
$$t_n = \text{length of } n\text{th CPU burst}$$

$$t_{n+1} = \text{expected length of next burst}$$

$$= a\,t_n + (1-a)\,t_n$$

where a is a parameter indicating how much to count past history (usually ½)

Shortest Job Next



Comparing exponential estimation with actual values: a = 1/2

Highest Response Ratio

- **H**ighest **R**esponse **R**atio **N**ext (HRRN, HRN)
  - Decision mode: non-preemptive
  - Arbitration rule: random or FIFO
  - Priority function: p(a, r, t) = (see below)

$$p = \frac{\text{estimated service time} + \text{waiting time so far}}{\text{estimated service time}}$$

SJN versus HRRN

- Shortest Job Next
  - Favors short jobs
  - Long jobs may have to wait a long time if short jobs appear frequently in the queue
- Highest Response Ratio
  - Still favors short jobs
  - More fair towards long jobs/processes
    - As long jobs wait their priority increases, giving them a chance to run

Selfish Round Robin

- Selfish Round Robin (SRR)
  - Decision mode: preemptive (at quantum)
  - Arbitration rule: first in, first out
  - Parameters:
    - a: rate priority of process in *new queue* increase
    - b: rate priority of process in *accepted queue* increase
    - q: quantum
  - Priority function: Let W be the time that a process must wait before entering the accepted queue:

$$p(r,W) = \begin{cases} br & r \le W \\ bW + a(r-W) & r > W \end{cases}$$

Selfish Round Robin

- General Idea:
  - New jobs are placed in the *new* queue with an initial priority of 0
    - Priority of job in *new* queue increase at rate *a*
  - Jobs move to the *accepted* queue when priority is equal to the priority of the *accepted* queue
    - Priority of jobs in the *accepted* queue increase at rate *b*
  - Jobs chosen from *accepted* queue in round robin fashion

Selfish Round Robin

quantum	running	next
0	A	A

New Queue ▶	B(0)
Accepted Queue ▶	A(2)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
1	A	A

New Queue ▶	B(3)	C(0)
Accepted Queue ▶	A(4)	

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
2	A	B

New Queue ▶	C(3) D(0)
Accepted Queue ▶	B(6) A(6)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
3	B	A

New Queue ▶	C(6) D(3) E(0)
Accepted Queue ▶	A(8) B(8)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
4	A	B

New Queue ▶	C(9) D(6) E(3)
Accepted Queue ▶	B(10) A(10)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
5	B	A

New Queue ▶ D(9) E(6)

Accepted Queue ▶ A(12) C(12) B(12)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
6	A	C

New Queue ▶	D(12) E(9)
Accepted Queue ▶	C(14) B(14) A(14)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
7	C	B

New Queue ▶	D(15) E(12)
Accepted Queue ▶	B(16) A(16) C(16)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
8	B	A

New Queue ▶	E(15)
Accepted Queue ▶	A(18) C(18) D(18) B(18)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
9	A	C

New Queue ▶	E(18)
Accepted Queue ▶	C(20) D(20) B(20) A(20)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
10	C	D

New Queue ▶	E(21)
Accepted Queue ▶	D(22) B(22) A(22) C(22)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Selfish Round Robin

quantum	running	next
11	D	B

New Queue ▶	
Accepted Queue ▶	B(24) A(24) C(24) E(24) D(24)

Let the new queue increase at a rate of a = 3, accepted at b = 2.

Multilevel Feedback

- **M**ultilevel **F**eedback **Q**ueues (MLF, MLFB)
  - Decision mode: preemptive (at quantum)
  - Arbitration rule: cyclic or chronological
  - Parameters: n levels each of priority T_p
  - Priority function: (see below)

$$p(a) = n - i, \quad 0 \le i < n$$

$$T_0(2^i - 1) \le a < T_0(2^{(i+1)} - 1)$$

$$\text{assuming } T_p = 2^p T_0$$

Multilevel Feedback

- General Idea:
  - n different queues exist with different priorities
  - Jobs start in uppermost level
    - After getting T_0 units of CPU time, job drop to next lower level
    - Jobs continue to drop until reach lowest queue
  - Favors short jobs by giving them more CPU time

Process Scheduling

Evaluation

Evaluation

- Deterministic modeling
  - Workout specific cases (like we did earlier)
- Simulation
  - Program a model, gather statistics
- Implementation
  - Implement algorithm on a system and observe
- Queuing Theory*
  - Represent system mathematically

Queuing Theory

- Little's Law
  - L: mean queue length
  - W: mean waiting time in queue
  - a: mean arrival rate for new jobs in queue

$$L = aW$$

  - Number of jobs leaving the queue is same as number arriving