Topic 16 Queues

"FISH queue: n.

[acronym, by analogy with FIFO (First In, First Out)] 'First In, Still Here'. A joking way of pointing out that processing of a particular sequence of events or requests has stopped dead. Also FISH mode and FISHnet; the latter may be applied to any network that is running really slowly or exhibiting extreme flakiness."

-The Jargon File 4.4.7

Queues

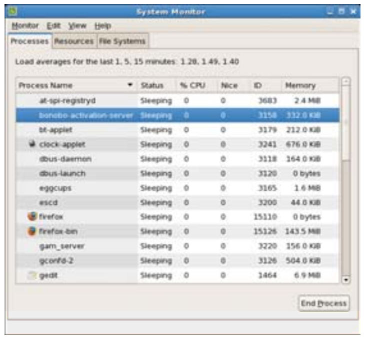
- Similar to Stacks
- Like a line
  - In Britain people don't "get in line" they "queue up".

Queue Properties

- Queues are a first in first out data structure
  - FIFO (or LILO, but that sounds a bit silly)
- Add items to the end of the queue
- Access and remove from the front
  - Access to the element that has been in the structure the ***longest*** amount of time
- Used extensively in operating systems
  - Queues of processes, I/O requests, and much more

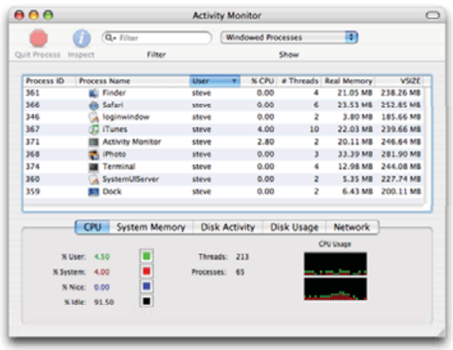
Queues in Operating Systems

- On a computer with 1 CPU, but many processes how many processes can actually use the CPU at a time?
- One job of OS, schedule the processes for the CPU
- issues: fairness, responsiveness, progress



Load averages for the last 1, 5, 15 minutes: 1.28, 1.49, 1.40

Process Name	Status	% CPU	Nice	ID	Memory
at-spi-registryd	Sleeping	0	0	3683	2.4 MiB
bonobo-activation-server	Sleeping	0	0	3158	332.0 KiB
bt-applet	Sleeping	0	0	3179	212.0 KiB
clock-applet	Sleeping	0	0	3241	676.0 KiB
dbus-daemon	Sleeping	0	0	3118	164.0 KiB
dbus-launch	Sleeping	0	0	3120	0 bytes
eggcups	Sleeping	0	0	3165	1.6 MiB
escd	Sleeping	0	0	3200	44.0 KiB
firefox	Sleeping	0	0	15110	0 bytes
firefox-bin	Sleeping	0	0	15126	143.5 MiB
gam_server	Sleeping	0	0	3220	156.0 KiB
gconfd-2	Sleeping	0	0	3126	504.0 KiB
gedit	Sleeping	0	0	1464	6.9 MiB



Show: Windowed Processes

Process ID	Process Name	User	% CPU	# Threads	Real Memory	VSIZE
361	Finder	steve	0.00	4	21.05 MB	238.26 MB
366	Safari	steve	0.00	6	23.53 MB	252.85 MB
346	loginwindow	steve	0.00	2	3.80 MB	185.66 MB
367	iTunes	steve	4.00	10	22.03 MB	239.66 MB
371	Activity Monitor	steve	2.80	2	20.11 MB	246.64 MB
368	iPhoto	steve	0.00	3	33.39 MB	281.90 MB
374	Terminal	steve	0.00	4	12.98 MB	244.08 MB
360	SystemUIServer	steve	0.00	2	5.35 MB	227.74 MB
359	Dock	steve	0.00	2	6.43 MB	200.11 MB

CPU | System Memory | Disk Activity | Disk Usage | Network

% User:	4.50	🟩
% System:	4.00	🟥
% Nice:	0.00	🟦
% Idle:	91.50	⬛

Threads: 213
Processes: 65

Queue operations

- `add(Object item)`
  - a.k.a. `enqueue(Object item)`
- `Object get()`
  - a.k.a. `Object front(), Object peek()`
- `Object remove()`
  - a.k.a. `Object dequeue()`
- `boolean isEmpty()`
- Specify in an interface, allow varied implementations

Queue interface, version 1

```
public interface Queue
{     //place item at back of this Queue
      enqueue(Object item);

      //access item at front of this queue
      //pre: !isEmpty()
      Object front();

      //remove item at front of this queue
      //pre: !isEmpty()
      Object dequeue();

      boolean isEmpty();
}
```

Implementing a Queue

- Given the internal storage container and choice for front and back of queue what are the Big O of the queue operations?

	ArrayList	LinkedList (Singly Linked)	LinkeList (Doubly Linked)
enqueue			
front			
dequeue			
isEmpty			

Attendance Question 1

- If implementing a queue with a singly linked list with references to the first and last nodes (head and tail) which end of the list should be the front of the queue in order to have all queue operations O(1)?

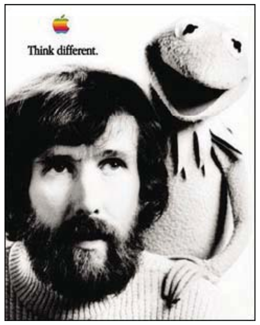
A. The front of the list should be the front of the queue

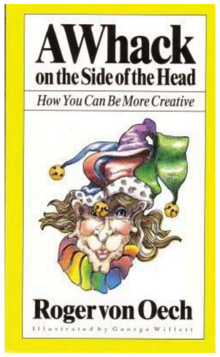
B. The back of the list should be the front of the queue.

C. D. E. I don't know, but I am sure looking forward to taking 307 again some time.

Alternate Implementation

- How about implementing a Queue with a native array?
  - Seems like a step backwards





Application of Queues

- Radix Sort
  - radix is a synonym for *base.* base 10, base 2
- Multi pass sorting algorithm that ***only*** looks at individual digits during each pass
- Use queues as *buckets* to store elements
- Create an array of 10 queues
- Starting with the least significant digit place value in queue that matches digit
- empty queues back into array
- repeat, moving to next least significant digit

Radix Sort in Action: 1s

- original values in array

  113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12
- Look at ones place

  113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12
- Queues:

0	70, 40	5	
1		6	86
2	12, 252, 12	7	37, 7
3	113, 93	8	
4		9	9, 79

Radix Sort in Action: 10s

- Empty queues in order from 0 to 9 back into array

  70, 40, 12, 252, 12, 113, 93, 86, 37, 7, 9, 79
- Now look at 10's place

  70, 40, 12, 252, 12, 113, 93, 86, 37, _7, _9, 79
- Queues:

0	_7, _9	5	252
1	12, 12, 113	6	
2		7	70, 79
3	37	8	86
4	40	9	93

Radix Sort in Action: 100s

- Empty queues in order from 0 to 9 back into array

  7, 9, 12, 12, 113, 37, 40, 252, 70, 79, 86, 93
- Now look at 100's place

  __7, __9, _12, _12, 113, _37, _40, 252, _70, _79, _86, _93
- Queues:

0	_7, _9, _12, _12, _40, _70, _79, _86, _93	5	
1	113	6	
2	252	7	
3		8	
4		9	

Radix Sort in Action: Final Step

- Empty queues in order from 0 to 9 back into array

  7, 9, 12, 12, 40, 70, 79, 86, 93, 113, 252

Radix Sort Code

```
public static void sort(int[] list){
    ArrayList<Queue<Integer>> queues = new ArrayList<Queue<Integer>>();
    for(int i = 0; i < 10; i++)
        queues.add( new LinkedList<Integer>() );
    int passes = numDigits( list[0] );
    int temp;
    for(int i = 1; i < list.length; i++){
        temp = numDigits(list[i]);
        if( temp > passes )
            passes = temp;
    }
    for(int i = 0; i < passes; i++){
        for(int j = 0; j < list.length; j++){
            queues.get(valueOfDigit(list[j], i)).add(list[j]);
        }
        int pos = 0;
        for(Queue<Integer> q : queues){
            while( !q.isEmpty())
                list[pos++] = q.remove();
        }
    }
}
```