CS2123 Data Structures
Queues

Dr. Weining Zhang
Department of Computer Science
University of Texas at San Antonio
August 23, 2009

Outline

1 Queues
   • Concepts
   • Implementation of QueueADT
   • Priority Queue

2 Application of Queues
   • Event-Based Simulation

Tasks of the Week

• Case Study: Linked implementation of the QueueADT
• Analyze the efficiency of the linked queue implementation.
• Case Study: Linked implementation of the PriorityQueueADT
• Case Study: Event-driven simulation of a bank

Queue

A linear structure for which elements are added at the rear and removed from the front
• Provide First In First Out (FIFO) access
QueueADT

```java
public interface QueueADT<T> {
    public void enqueue(T element);
    public T dequeue();
    public T first();
    public boolean isEmpty();
    public int size();
    public String toString();
}
```

Other Implementation Options

- ArrayQueue: implement QueueADT using a generic array.
- Implement QueueADT by extending a list class, such as ArrayList or LinkedList (more to come in future)

PriorityQueue

- A queue in which elements are ordered according to a priority metrics
  - Typically, the priority is represented by a number and the priority queue can store elements in ascending order (that is, the smaller the number is the higher the priority is) or in descending order (that is the larger the number is the higher the priority is)
  - The major difference from QueueADT is the add method
    - Elements must be stored according to their priorities
**PriorityQueueADT**

```java
public interface PriorityQueueADT <T> {
    public void add(T element);
    public T removeMin();
    public T findMin();
    public void makeEmpty();
    public boolean isEmpty();
    public int size();
    public String toString();
}
```

**LinkedPriorityQueue**

- Implement PriorityQueueADT using a simple linked list
- Since new elements are added based on their priorities, there is no need to keep a reference to the rear
- Assume elements are compared by their priority measure, that is, e1.compareTo(e2) < 0 if e1 has a higher priority.
- Remove min is to remove the first element.

**Event-Based Simulation**

- A common strategy to represent and simulate real-world systems in computers
- Represent things happening at different time as Events
- System components are represented as event handlers
- Real-world systems are represented as simulation models
- Events are ordered by time
  - The time starts at 0 and increments whenever an event happens

**Event**

- Each event has a
  - Time: when it is supposed to happen
  - Type: what it represents
  - Target: whom the event is for

**Example**

The event that represents a customer arriving at the bank at time 100 will have

- time = 100.
- type = customer_arrival
- target = customer
Events are generated and processed by system components that act as event handlers.

- Event handler can do the following:
  - Generate optional initial event.
  - Process given event and optionally generate a new event.

A simulation model is an object that runs the simulation. It has:
- A clock keeping the current time, which is advanced to the time of the current event;
- A set of event handlers as system components;
- A priority queue storing future events ordered in time.

To run a simulation, the simulation model will:
- Create a set of initial events.
- Add them to the priority queue.
- Repeatedly dispatch the first event in the priority queue to its handler, until the simulation time is over.

As long as the stop time has not reached, repeat the following:
- Get the next event, and update the clock to the event time.
- Determine whom the event is for and decide if the event handler is available to handle the event.
  - If not, the event is either dropped or buffered for future process.
  - If a handler is available, give the event to its handler.
    - The handler can add a new event to the event queue, or ask other event generators to add a new event.

The model keeps a list of doors, a list of clerks, a list of served customers, the current time, and an event queue (a priority queue).

Customers arrive at different doors of a bank at different times.
- Each customer is an object that contains an arrival time and an amount of time needed to serve.
- Each door is represented by a cGenerator object, which contains a list of customers.

When a customer arrives, if no clerk is available, the customer leaves (no waiting list or buffer).

- Each clerk is an object:
  - It takes a different amount of time to serve different customers.
  - It can create new event for completion of a service.
Run a simulation

- initialize the event queue and current time
- remove first event, update current time, dispatch the event to a handler of that event, until event queue is empty
- To handle a customer_arrival event:
  - find an available clerk, add a new service_finished event to event queue
- To handle a service.finished event:
  - get the clerk from the event
  - get the customer from the clerk
  - set customer leaving time, and
  - add the customer to serviced list
  - make the clerk available