# Aufgabenstellung Warteschlangen:

Bezeichnung erfolgt nach Kendall-Symbolik A/B/m/n

* A und B kennzeichnen die Art der Verteilung der Ankünfte und der Bedienung der Kunden,  
  wichtige Verteilungen sind zB die Exponentialverteilung (M) und die diskrete Verteilung (D).
* m gibt die Anzahl der Bedienstationen an
* n gibt die Größe des Warteraums an.

Einfache Formeln ergeben sich für MM1∞, also für exponentialverteilte Ankünfte und Bedienungen und einen unendlich großen Warteraum, in dem keine Kunden abgewiesen werden müssen.

Dann gelten folgende Formeln:  
*Ta* Zwischenankunftszeit (die Zeit, die zwischen der Ankunft von 2 Kunden vergeht)  
 Ankunftsrate = 1/Ta (die Anzahl der Kunden, die pro Zeit im System ankommen)  
*Ts* Bedienzeit (die Zeit, die benötigt wird, um einen Kunden zu bedienen)  
 Bedienrate (die Anzahl an Kunden, die pro Zeit bedient werden können)  
 Auslastung des System (muss < 1 damit die Warteschlange nicht unaufhörlich wächst)

= Gesamte Auslastung des Systems, wenn darin mehrere Warteschlangen mit unterschiedlichen Bedienraten bedient werden

*Tv* Verweilzeit = Tw + Ts (Gesamtzeit, die der Kunde im System verbleibt)

Verweilzeit eines Kunden – dann ergeben sich evtl für jeden Kunden andere Tw

mittlere Wartezeit in der Schlange vor der Bedienung = ca der Mittelwert aller Tw, die man aus der Formel für Tv berechnet hat in dem man Tw=Tv-Ts verwendet

(Bemerkung Tw kann man auch umformen, dann erhält man verschiedene Formel in denen ,.. vorkommen - angeführte ist aber die allgemeinste, weil sie auch gilt wenn mehrere Schlangen in einer Warteschlange von mehrerern Bedienern bedient werden. Sonst muss man aufpassen, weil sie evtl nur für MM1∞ und eine einzige Schlange gelten)

Anzahl der Kunden, die gerade im System sind

Allgemeine Bemerkungen  
Folgende kompliziertere Fälle können eintreten:

1. Eine Warteschlange die sich auf zwei Bedienstationen aufteilt: wird halbiert (am einfachsten) oder im Verhältnis der Bedienraten aufgeteilt, so dass alle Bedienstationen gleich ausgelastet werden.
2. Eine Bedienstation muss zwei Warteschlangen mit unterschiedlichen Ankunftsraten und Bedienraten bedienen:  
    und berechnen und die allgemeine Formel für Tw verwenden. Dann ist für jeden Kunden Tv = Tw + Ts, mit dem allgemeinen Tw und seinem Ts.  
   Andere Möglichkeit: berechnen und mit der Formel für Tv die einzelnen Tw berechnen. Achtung: der Mittelwert der so berechneten Tw entspricht nur ungefähr dem Tw aus der allgemeinen Formel!
3. Eine Bedienstation hat nicht die gesamte Zeit zur Verfügung. Dann reduziert sich anteilsmäßig:  
   zB Bedienzeit=1 Minute, dann könnten 60 Kunden pro Stunde bedient werden dh . Wenn die Bedienung aber zB 20 Minuten pro Stunde für etwas andere benötigt wird reduziert sich auf also auf 40/h und man rechnet mit diesem weiter.

Was man wissen muss:

Benennen und erklären sie in den folgenden Formeln die Variablen!

## Beispielrechnung

Berechnen sie die für die Dimensionierung der Hardware wesentlichen Informationen der folgenden Konfiguration eines Servers, der sowohl Druckserver als auch Authentifizierungserver ist.

Dient hier als Beispiel, die Anmeldungen werden unter Umständen sogar viel mehr sein, zB bei uns ein paar hundert in der Früh in den ersten 10 Minuten, wobei man da schon bemerkt, dass es manchmal zu ziemlichen Antwortzeiten kommen kann. Bei Server-gespeicherten Profilen kann es noch schlimmer werden, weil da der Desktop zwischen Server und lokalem PC abgeglichen wird und es dabei zu ziemlich großen Transfermengen kommen kann zB wenn Leute ein Video am Desktop abspeichern und dieses dann bei jeder anmeldung vom Server zum PC geladen und am Ende vom PC zum Server transferiert wird, wenn man den PC wechselt.

Drucker:   
Eingangsrate: 20 Druckjobs / Stunde   
Bedienzeit: 2 Minuten

Authentifizierung:

Eingangsrate: durchschnittlich 360 Anfragen/ Stunde  
Bedienzeit: 3 Sekunden

## Lösung:

Drucker:

= 20

= 60/Bedienzeit = 60/2 = 30/Stunde

Authentifizierung:

= 360 / Stunde

= 3600/Bedienzeit = 3600/3 = 1200 / Stunde

**Die Auslastung des Servers beträgt ca 77%**

= 360+20 = 380/Stunde

**0,0066306 Stunden = 23,8 Sekunden**

Dieser Wert ist eindeutig zu lang, insbesondere für die Authentifizieung – auf einen eigenen Server auslagern wäre eine Möglichkeit (das macht man in der Regel auch so, oder man nimmt zwei Authentifizierungsserver, dann könnte man lambda halbieren, nochmal rechnen und schauen was passiert)

Jobs Im System (wichtig für die Speicherkapazität der Druckerwarteschlange):

= **3,29 Jobs im System**

Das sollte also von der Datenmenge her kein Problem sein.

# Simulation von Warteschlangen

Statt die Formeln zu verwenden, wird meist eine Simulation ausgeführt, dh man läßt das Programm mit sleepd warten und simuliert so die Ankünfte und Bedienungen, aber eben viel schneller in der Realität, in dem man den Zeitfaktor ändert. zB statt 1 Minute eine Millisekunde verwendet.

Man benötigt Zahlen, die einer gewissen Verteilung entsprechen.

Mathematisches Gesetz (Lemma=Hilfs-Aussage):

Aus einer gleichverteilten Folge von Zahlen kann eine beliebig anders verteilte Verteilungsfunktion erzeugt werden. Dazu verwendet man die Inversionsmethode (siehe weiter unten)

## Zufallszahlengenerator

Zuerst braucht man also eine gleichverteile Folge von Zufallszahlen.

Die Funktion rand() erzeugt so eine Folge (in C Zahlen zwischen 0 und 65536) dahinter steckt eine Funktion, die jeden Wert aus dem vorhergehenden durch Multiplikation, Addition und Modulo Bildung erzeugt. x=(a\*x+c) mod m, die Folge wiederholt sich nach m Durchläufen ist also nicht wirklich zufällig. Wenn man nur ein paar Zahlen kennt, weiß man normalerweise nicht was als nächstes kommt, aber eigentlich könnte man die ganze Zahlenfolge berechnen und wenn man ein paar Zahlen hat, könnte man schauen wo diese Folge in der ganzen Reihe vorkommt. Die Reihe ist allerdings recht lang zB 231 bei der Funktion in C.  
Wichtig ist auch mit welchem x man startet. Deshalb legt man den Startwert oft auf den Zeitpunkt zu dem das Programm gestartet wird zB mit srand(time(0)), wobei time(0) jeweils die Anzahl an Sekunden liefert, die 1.1.1970 vergangen sind, man fängt also irgendwo innerhalb der 231 Zahlen an.

Wenn man Zahlen in einem speziellen Zahlenbereich benötigt kann man zB

x=min + rand()%(max - min) verwenden

Benötigt man zB Wahrscheinlichkeiten, die ja zwischen 0 und 1 liegen sollen kann man

zB p=(float)rand()/MAX\_RAND verwenden

Bei anderen Rechenvorgängen (zB nur die niederwertigen Bits verwenden), um spezielle Teilbereiche zu bekommen muss man aufpassen, weil das Ergebnis evtl dann nicht mehr gleichverteilt ist.  
Es hat auch schon Zufallszahlengeneratoren (zB vor 30 Jahren bei einem von IBM im Bankwesen eingesetzten) gegeben, bei denen man auf Fehler in der Verteilung draufgekommen ist, die dann zB Muster in der Verteilung hatten.

## Diskrete Verteilung

So wie kommt man jetzt zu einer anderen Verteilungsfunktion aus den mit rand erzeugten Werten:

Einfachster Fall „diskrete Verteilung“ dh ein gewisser Prozentsatz der erzeugten Werte soll einen speziellen Wert erhalten.

Beispiel 25% der Werte sollen true sein, der Rest false

Das bekommt man recht einfach hin:

while()  
{

…

p=(float)rand()/MAX\_RAND;

if(p<0.25)

ergebnis=true;

else

ergebnis=false;

…

}

P bekommt alle möglichen Werte zwischen 0 und 1, wobei jeder Wert gleich wahrscheinlich ist, es werden also 25% der Werte im Intervall zwischen 0 und 0.25 leigen und 75% außerhalb.

## Andere Verteilungen:

Beispiel Exponentialverteilung

F(x)=1-e-λx

λ gibt die Anzahl der zu erwartenden Ereignisse in einer Zeiteinheit an, F(x) liefert die Wahrscheinlichkeit, dass in der Zeitspanne x ein Ereignis auftritt.

Beispiel:

λ = 2 / Stunde dann ist die Wahrscheinlichkeit das innerhalb einer halben Stunde ein Ereignis auftritt F(0,5)=1-e-2\*0,5=0,632 also knapp über 63 %. Die Wahrscheinlichkeit dafür, dass innerhalb von zwei Stunden ein Ereignis eintritt ist 98,2% dh es besteht zu 1,8% auch die Möglichkeit, das kein Ereignis auftritt.

Mit der Formel kann man also zu bekannten Zeitintervallen die Wahrscheinlichkeit für die Anzahl von Ereignissen berechnen. Für die Simulationsrechnung benötigt man aber Ereignisse deren Zeitabstände mit einer bestimmten Wahrscheinlichkeit variieren.

Das Lemma der Inversionsmethode besagt folgendes:   
Wenn y eine Verteilungsfunktion zu x ist, dann gehorcht umgekehrt x der gleichen Verteilungsfunktion, wenn y gleichverteilt ist.

Dazu formt man y=1-e-λx so um, dass x alleine auf einer Seite der Gleichung steht:  
x=-ln(1-y)/λ x ergibt jetzt exponentialverteilte Zeitintervalle wenn y gleichverteilt ist.

Das kann man sehr leicht programmieren, es ist nur eine Funktion mit einer Zeile und man kann sogar statt 1-y gleich nur mit y rechnen, weil wenn 1- y zufällige gleichverteilt zwischen 0 und 1 ist dann ist das auch y.

Wie sieht es mit der Normalverteilung aus: geht nur durch Potentialreihenzerlegung weil in F ein Inetgral vorkommt und verbraucht dementsprechend mehr Rechenzeit um die zufälligen Zeitintervalle zu berechnen

Andere Verteilungen sind auch komplexer oder gar nicht auf gleichverteilte Zufallszahlen rückführbar.

Es gibt zwei Möglichkeiten für die Simulation

## 1. Threads (also parallel)

Mit threads, ist es intuitiver, weil es die Realität abbildet

erster thread Ankunft:

n++, dh ein Klient ist angekommen

dann eine Zeit lang warten, bis der nächste Klient kommt: sleep(exporand(lambda))  
die Zeit entspricht der durchschnittlichen Zwischenankunftszeit +/- einer zufälligen Abweichung von dieser Zeit

zweiter thread Bedienung:

wenn n>0 ist n—(das muss ein atomarer Prozess sein)

dann eine zufällige Zeit lang warten: sleep(exporand(mu))

die Zeit entspricht der durchschnittlichen Bedienzeit +/- einer zufälligen Abweichung von dieser Zeit

Problematisch ist dabei das Sleep, weil die Zeitmessung unter Linux und Windows recht ungenau wird, wenn man kleine Intervalle betrachtet, und bei größeren Werte im Sleep dauert die Simualation länger. Mit halbwegs guter Genauigkeit kann man höchstens 20 Kunden pro Sekunde simulieren.

Man kann auch auf dummy Schleifen ausweichen, die nichts Sinnvolles machen außer Zeit zu verbrauchen, dabei muss man aufpassen, dass sei nicht vom Compiler wegoptimiert werden (globale Variablen in der Schleife inkrementieren, dann traut er sich normalerweise nicht!!!)

Und nicht vergessen das Überprüfen ob n>0 und das n- - zu atomaren Prozessen zu machen, also mit lock, synchronize, mutex oder semaphoren so sperren, dass keine negativen Kundenzahlen entstehen können!!!

## 2. Sequentiell

Man gibt eine gewisse Simualtionsdauer T vor, teilt diese in kleine regelmäßige Zeitabschnitte dt und beobachtet was in einem Zeitabschnitt geschieht:

while (gesamtzeit<T)

{

gesamtzeit=gesamtzeit+dt;

while (azeit <= gesamtzeit)

{

n++;

a=exporand(lambda);

azeit = azeit + a;

}

while (bzeit <= gesamtzeit && n>0)

{

n--;

b=exporand(mu);

bzeit = bzeit + b;

}

if(azeit>bzeit&&n==0) bzeit=azeit;

N = N + n;

steps++;

}

N = N / steps;

Erklärung:   
gesamtzeit gibt an wieviel von der gesamten Simualtionsdauer schon vergangen ist  
azeit … die Ankunftsintervalle zusammengezählt, da kann es keine Lücken geben  
  
erste while Schleife: Ankünfte. während des Intervalls dt können solange Leute ankommen, solange der Abstand zwischen zwei Ankünften klein genug sind

Zweite while Schleife: Bedienung es können solange Klienten bedient werden, bis der Abschluss der Bedienung über das gerade betrachtete Intervall hinaus dauern würde. Wenn kein Kunde im System ist, gibt es keine Bedienung, deshalb das if>0

if(azeit>bzeit …) hier wird berücksichtigt, dass ein Kunde sofort bedient werden kann, wenn er der erste ist.  
n enthält die Anzahl der Klienten in der Warteschlange und wird aufsummiert, um die mittlere Länge der Warteschlange zu berechnen.