Technische Universität München
Informatik VIII
Netzarchitekturen und Netzdienste
Prof. Dr.-Ing. Georg Carle

Alexander Klein, Nils Kammenhuber

Übungen zur Vorlesung Diskrete Simulation

Übungsblatt 4, SS 2010

Abgabe: 7. Juli 2010 (bis 14 Uhr am Lehrstuhl I-8 in Finger 03.05, per E-Mail an `klein` und `hirvi@net.in.tum.de` oder in der Vorlesung). Die Aufgaben können jeweils von zwei Studenten gemeinsam bearbeitet werden.

In der letzten Übung haben Sie einen schlechten LCG-Zufallszahlengenerator durch einen guten Generator (Mersenne-Twister) ersetzt. Auf diesem Übungsblatt soll u. a. untersucht werden, welche Auswirkungen ein schlechter Zufallszahlengenerator auf die Simulationsergebnisse in unserer einfachen Simulation hat.

Aufgabe 1 — Implementierung eines LCG (20 Punkte)

Ziele: *Wiederholung des LCG-Konzeptes; Vorbereitung auf die nächste Aufgabe*

Schreiben Sie eine Klasse `RandomLCG`, welche einen LCG-Zufallszahlengenerator implementiert.

Beachten Sie beim Entwurf, dass die Klasse in bezug auf Ihren Event-Simulator aus den vorigen Übungsblättern die gleiche Schnittstelle bieten soll wie `java.lang.Random` bzw. die Mersenne-Twister-Klasse. Es reicht hierbei aus, die Funktionen `nextDouble()` und das Setzen des Seeds zu unterstützen; beachten Sie auch die folgende Aufgabe. Darüberhinaus sollte man natürlich die Parameter a, c und m einstellen können. Wenn kein Seed vorgegeben ist, soll standardmäßig als Seed $m - 1$ angenommen werden.

Aufgabe 2 — Auswirkungen von Zufallszahlengeneratoren (80 Punkte)

Ziele: *Kennenlernen des Statistik-Programms; Anwendung von Konfidenzintervallen; Wiederholung Nomenklatur Wartesysteme; Automatisierung bei umfangreicheren Simulationsprojekten; Stolperfallen im Zusammenhang mit Zufallszahlen und Simulationen; Exploratives Erkunden des Simulationsparam Raumes*

Nun wollen wir dieselbe Simulation mit verschiedenen Zufallszahlengeneratoren laufen lassen und die Ergebnisse untersuchen.

a) Ändern Sie Ihren Simulator so ab, dass man beim Start den Zufallsgenerator wählen und ihm *optional* den Seed und ggf. weitere Parameter (a, c, g) übergeben kann.

b) Ändern Sie Ihren Simulator ggf. so ab, dass er am Ende der Simulation lediglich das arithmetische Mittel und die Varianz (oder Standardabweichung) der Wartezeiten und das arithmetische Mittel und die Varianz (oder Standardabweichung) der Bedienzeiten ausgibt.

c) Schauen Sie sich die nächsten Teilaufgaben an und überlegen Sie sich, wie Sie die vielen verschiedenen Simulationsläufe und evtl. auch die Auswertung automatisieren können; z. B. Shell-Skript, weitere Klasse o. ä.
(Es muss kein Text abgegeben werden.)

d) Simulieren Sie jeweils $40\times$ ein $M/M/1/\infty$-Wartesystem mit $\lambda = \frac{8}{\mathrm{s}}$ und $\mu = \frac{10}{\mathrm{s}}$ für 5000 s. Verwenden Sie als Zufallszahlengenerator folgende Generatoren:

- LCG mit $a = 5, c = 1, m = 8192$; keine Seed-Vorgabe
- LCG mit $a = 5, c = 1, m = 8192$; Seeds von $1 \ldots 40$
- LCG mit $a = 6, c = 1, m = 8192$; Seeds von $1 \ldots 40$
- RANDU; Seeds von $1 \ldots 40$
- `java.util.Random`
- Mersenne-Twister; keine Seed-Vorgabe
- Mersenne-Twister; Seeds von $1 \ldots 40$

Vergleichen und interpretieren Sie die Ergebnisse der verschiedenen Zufallszahlengeneratoren.

e) *Teilaufgabe (e) wurde gestrichen.*

f) Ändert sich obiges Bild, wenn man die Simulationen jeweils viel länger laufen lässt? Experimentieren Sie nach eigenem Gutdünken herum. Sie können sich auch auf eine sinnvolle Teilmenge an Zufallsgeneratoren beschränken.