

```

1:
2: mmldes <- function() {
3:   initglobals()
4:   # create simulation
5:   newsim()
6:   # get things going, generating and scheduling first arrival event
7:   arvrtime <- rexp(1,rate=arvrate)
8:   schedevnt(arvrtime,arvtype,arvrtime)
9:   mainloop(10000.0)
10:  return(totwait/njobsdone)
11: }
12:
13: # application: M/M/1 queue, arrival rate 0.5, service rate 1.0; we must
14: # set the globals and the function reactevnt() for this application
15:
16: # initializes the global variables
17: initglobals <- function() {
18:   # globals
19:   # rates
20:   arvrate <- 0.5 # arrival rate
21:   srvrate <- 1.0 # service rate
22:   # event types
23:   arvtype <- 1 # arrival type
24:   srvdonetype <- 2 # service done type
25:   # server queue, consisting of arrival times of queued jobs
26:   srvc <- vector(length=0)
27:   # statistics
28:   njobsdone <- 0 # jobs done so far
29:   totwait <- 0.0 # total wait time so far
30: }
31:
32: # application-specific event processing function required by mainloop()
33: # in the general DES library above
34: reactevnt <- function(head) {
35:   if (head[2] == arvtype) { # arrival
36:     # if server free, start service, else add to queue
37:     if (length(srvq) == 0) {
38:       srvc <- head[3]
39:       srvdonetime <- sim$curtime + rexp(1,srvrate)
40:       schedevnt(srvdonetime,srvdonetype,head[3])
41:     } else srvc <- c(srvq,head[3])
42:     # generate next arrival
43:     arvrtime <- sim$curtime + rexp(1,arvrate)
44:     schedevnt(arvrtime,arvtype,arvrtime)
45:   } else { # service done
46:     # process job that just finished
47:     # do accounting
48:     njobsdone <- njobsdone + 1
49:     totwait <- totwait + sim$curtime - head[3]
50:     # remove from queue
51:     srvc <- srvc[-1]
52:     # more still in the queue?
53:     if (length(srvq) > 0) {
54:       # schedule new service
55:       srvdonetime <- sim$curtime + rexp(1,srvrate)
56:       schedevnt(srvdonetime,srvdonetype,srvq[1])
57:     }
58:   }
59: }
60:
61: # DES.R: R routines for discrete-event simulation (DES), with an example
62:
63: # each event will be represented by a vector; the first component will
64: # be the time the event is to occur; the second component will be the
65: # numerical code for the programmer-defined event type; the programmer
66: # may add further components that are application-specific
67:
68: # a list named "sim" holds the events list and other information; for
69: # convenience, sim has been stored as a global variable; some functions
70: # have side effects
71:
72: # create "sim"
73: newsim <- function() {
74:   sim <- list()
75:   sim$curtime <- 0.0 # current simulated time
76:   sim$evnts <- NULL # event list
77: }
78:
79: # insert event evnt into event list
80: insertevnt <- function(evnt) {
81:   # if the event list is empty, set it to consist of evnt and return
82:   if (is.null(sim$evnts)) {
83:     sim$evnts <- matrix(evnt,nrow=1)
84:     return()
85:   }
86:   # otherwise, find insertion point
87:   inspt <- binsearch(sim$evnts[,1],evnt[1])
88:   # now "insert," by reconstructing the matrix; we find what portion of
89:   # the current matrix should come before evnt and what portion should
90:   # come after it, then string everything together
91:   before <- if (inspt == 1) NULL else sim$evnts[1:(inspt-1),]
92:   nr <- nrow(sim$evnts)
93:   after <- if (inspt <= nr) sim$evnts[inspt:nr,] else NULL
94:   sim$evnts <- rbind(before,evnt,after)
95: }
96:
97: # schedule new event; evnttime is the time at which the event is to
98: # occur; evnttype is the event type; and appfields are the values of the
99: # programmer-defined fields, if any
100: schedevnt <- function(evnttime,evnttype,appfields=NULL) {
101:   evnt <- c(evnttime,evnttype,appfields)
102:   insertevnt(evnt)
103: }
104:
105: # start to process next event (second half done by application
106: # programmer via call to reactevnt() from mainloop())
107: getnextevnt <- function() {
108:   head <- sim$evnts[1,]
109:   # delete head
110:   if (nrow(sim$evnts) == 1) sim$evnts <- NULL else
111:     sim$evnts <- sim$evnts[-1,,drop=F]
112:   return(head)
113: }
114:
115: # main loop of the simulation
116: mainloop <- function(maxsimtime) {
117:   while(sim$curtime < maxsimtime) {
118:     head <- getnextevnt()
119:     # update current simulated time
120:     sim$curtime <- head[1]
121:     # process this event (programmer-supplied ftn)
122:     reactevnt(head)
123:   }
124: }
125:
126: # binary search of insertion point of y in the sorted vector x; returns
127: # the position in x before which y should be inserted, with the value
128: # length(x)+1 if y is larger than x[length(x)]
129: binsearch <- function(x,y) {
130:   n <- length(x)
131:   lo <- 1
132:   hi <- n
133:   while(lo+1 < hi) {
134:     mid <- floor((lo+hi)/2)
135:     if (y == x[mid]) return(mid)
136:     if (y < x[mid]) hi <- mid else lo <- mid
137:   }
138:   if (y <= x[lo]) return(lo)

```

```
139:   if (y < x[hi]) return(hi)
140:   return(hi+1)
141: }
142:
```