# DES.R

March 2017 major changes (updated September 2017)

1. No longer keep the event set in sorted order. Too costly to do insertion, and anyway earliest event can be determined via which.min(), a C-level function that should be much faster. (There is also a provision for an "arrivals event set," for arrivals only, taking advantage of the ordered nature of pre-generated arrivals.)

2. Similarly, there is no dynamic resizing of the event set. Space is marked as either free or in use. This requires the user to provide an upper bound for the maximum number of events, a restriction, but should result in quite a performance boost.

3. In old version, used matrix instead of data frame, as latter was quite slow, and now back to matrix. Probably should go back to data frame, maybe data.table, but for now, at least add meaningful column names and use them.

All data is stored in an R environment variable that will be referred to as simlist below; an environment variable is used so that functions can change their simlist components, rather than reassigning.

The simlist will consist of the following components:

- currtime: current simulated time
- timelim: max simulated time
- timelim2: double timelim
- evnts: the events list, a matrix, one event per row; timelim2 value in first col means this row is free
- reactevent: event handler function, user-supplied; creates new events upon the occurrence of an old one; e.g. job arrival triggers either start of service for the job or queuing it; call form is reactevent(evnt,simlist)
- dbg: if TRUE, will print simlist$evnts after each call to simlist$reactevent(), and enter R browser for single-stepping etc.

The application code can add further application-specific data to simlist, e.g. total job queuing time.

Each event will be represented by a matrix row consisting of columns:

- occurrence time
- event type: user-defined numeric code, e.g. 1 for arrival, 2 for job completion, etc. (must be numeric as this is a matrix, but one can of course give names to the codes)
- application-specific information, if any

Library functions:

- newsim: create a new simlist
- schedevnt: schedule a new event
- getfreerow: find a free row in the event set
- getnextevnt: pulls the earliest event from the event set, updates the current simulated time, and processes this event; usually not called by users
- mainloop: as the name implies
# cancelevnt: cancel a previously-scheduled event
# newqueue: creates a new work queue
# appendfcfs: append job to a FCFS queue
# delfcfs: delete head of a FCFS queue
# exparrivals: convenience function if arrivals can all be
generated ahead of time

# event set:
# matrix in simlist
# one row for each event, rows NOT ordered by event occurrence time
# first two cols are event time, event type, then app-specific info,
# if any

# outline of a typical application:
# mysim <- newsim() create the simlist
# set reactevent in mysim
# set application-specific variables in mysim, if any
# set the first event(s) in mysim$evnts
# mainloop(mysim)
# print results

# create a simlist, which will be the return value, an R environment;
# appcols is the vector of names for the application-specific columns;
# maxesize is the maximum number of rows needed for the event set
newsim <- function(timelim,maxesize,appcols=NULL,aevntset=FALSE,dbg=FALSE) {
    simlist <- new.env()
    simlist$currtime <- 0.0  # current simulated time
    simlist$timelim <- timelim
    simlist$timelim2 <- 2 * timelim
    simlist$passedtime <- function(z) z > simlist$timelim
    simlist$evnts <-matrix(nrow=maxesize,ncol=2+length(appcols))  # event set
    colnames(simlist$evnts) <- c('evnttime','evnttype',appcols)
    simlist$evnts[,1] <- simlist$timelim2
    simlist$aevnts <- NULL  # will be reset by exparrivals()
    simlist$nextaevnt <- 1  # row number in aevnts of next arrival
    simlist$dbg <- dbg
    simlist
}

# schedule new event in simlist$evnts; evnttime is the time at
# which the event is to occur; evnttype is the event type; appdata is
# a vector of numerical application-specific data
schedevnt <- function(simlist,evnttime,evnttype,appdata=NULL) {
    evnt <- c(evnttime,evnttype,appdata)
    fr <- getfreerow(simlist)
    simlist$evnts[fr,] <- evnt
}

# find number of the first free row
getfreerow <- function(simlist) {
    evtimes <- simlist$evnts[,1]
    tmp <- Position(simlist$passedtime,evtimes)
    if (is.na(tmp)) stop('no room for new event')
    tmp
}
getnextevnt <- function(simlist) {
    # find earliest event
    etimes <- simlist$evnts[,1]
    whichnexte <- which.min(etimes)
    nextetime <- etimes[whichnexte]
    if (simlist$nextaevntset) {
        nextatime <- simlist$aevnts[simlist$nextaevnt,1]
        if (nextatime < nextetime) {
            oldrow <- simlist$nextaevnt
            simlist$nextaevnt <- oldrow + 1
            return(simlist$aevnts[oldrow,])
        }
    }
    head <- simlist$evnts[whichnexte,]
    simlist$evnts[whichnexte,1] <- simlist$timelim2
    return(head)
}

delevnt <- function(i,simlist) {
    # simlist$evnts <- simlist$evnts[-i,,drop=F]
    simlist$evnts[i,1] <- Inf
    simlist$emptyrow <- i
}

mainloop <- function(simlist) {
    simtimelim <- simlist$timelim
    while(TRUE) {
        head <- getnextevnt(simlist)
        etime <- head['evnttime']
        # update current simulated time
        if (etime > simlist$timelim) return()
        simlist$currtime <- etime
        # process this event (programmer-supplied ftn)
        simlist$reactevent(head,simlist)
        if (simlist$dbg) {
            print("event occurred:")
            print(head)
            print("events list now")
            print(simlist$evnts)
            browser()
        }
    }
}

binsearch <- function(x,y) {
    n <- length(x)
    lo <- 1
    hi <- n
    while(lo+1 < hi) {
        mid <- floor((lo+hi)/2)
        if (y == x[mid]) return(mid)
        if (y > x[mid]) lo <- mid + 1
        else hi <- mid
    }
    return(lo)
}
## if (y < x[mid]) hi <- mid else lo <- mid
##
## if (y <= x[lo]) return(lo)
## if (y < x[hi]) return(hi)
## return(hi+1)
##}

# removes the specified event from the schedule list
	cancelevnt <- function(rownum,simlist) {
		simlist$evnts[rownum,1] <- simlist$timelim2
	}

# the work queue functions below assume that queues are represented as
# matrices, one row per queued job, containing application-specific
# information about the job; the matrix is assumed stored in an
# environment, with the matrix being named m

# create and return new queue with ncol columns; the queue is an R
# environment, with the main component being m, the matrix representing
# the queue itself; ncol is up to the user, depending on how many pieces
# of information the user wishes to record about a job
	newqueue <- function(simlist) {
		if (is.null(simlist$evnts)) stop('no event set')
		n <- new.env()
		q$m <- matrix(nrow=0,ncol=ncol(simlist$evnts))
		n
	}

# appends jobtoqueue to the given queue, assumed of the above form;
# jobtoqueue is a vector of length equal to the number of columns in
# the queue matrix
	appendfcfs <- function(queue,jobtoqueue) {
		if (is.null(queue$m)) {
			queue$m <- matrix(jobtoqueue,nrow=1)
		} else {
			queue$m <- rbind(queue$m,jobtoqueue)
			return()
	}

# deletes and returns head of queue
	delfcfs <- function(queue) {
		if (is.null(queue$m)) return(NULL)
		nhead <- queue$m[1,]
		queue$m <- queue$m[1,,drop=F]
		return(nhead)
	}

# in many cases, we have exponential interarrivals that occur
# independently of the rest of the system; this function generates all
# arrivals at the outset, placing them in a separate arrivals event set

texparrivals <- function(simlist,meaninterarr,batchsize=10000) {
		if (!simlist$aevntset) stop("newsim() wasn’t called with aevntset TRUE")
		es <- simlist$evnts
		cn <- colnames(es)
		if (cn[3] != 'arrvtime') stop("col 3 must be "arrvtime"")
		if (cn[4] != 'jobnum') stop("col 3 must be "jobnum")
		erate <- 1 / meaninterarr
		s <- 0
		allarvs <- NULL
		while(s < simlist$timelim) {
		narvs <- rexp(batchsize,erate)
		arvs <- s + sum(arvs)
		s <- s + sum(arvs)
allarvs <- c(allarvs, arvs)

# may have overshot the mark
cuallarvs <- cumsum(allarvs)
allarvs <- allarvs[cuallarvs <= simlist$timelim]
nallarvs <- length(allarvs)
if (nallarvs == 0) stop('no arrivals before timelim')
cuallarvs <- cuallarvs[1:nallarvs]
maxesize <- nallarvs + nrow(es)
newes <- matrix(nrow=maxesize, ncol=ncol(es))
nonempty <- 1:nallarvs
newes[nonempty,1] <- cuallarvs
if (is.null(simlist$arrvevnt)) stop('simlist$arrvevnt undefined')
newes[nonempty,2] <- simlist$arrvevnt
newes[nonempty,3] <- newes[nonempty,1]
newes[nonempty,4] <- 1:nallarvs
newes[!nonempty,1] <- simlist$timelim2
colnames(newes) <- cn
simlist$aevnts <- newes