NAME

makePolicyArray, makeValueArray, destroyPolicyArray, destroyValueArray − create and destroy arrays to hold QNET policy and value data

SYNOPSIS

#include <qnet.h>

int* makePolicyArray(int *ndim*, **int** *nsrv*, **int** *trunc[]*);
void destroyPolicyArray(void* *p*, **int** *ndim*, **int** *nsrv*, **int** *trunc[]*);

double* makeValueArray(int *ndim*, **int** *nval*, **int** *trunc[]*);
void destroyValueArray(void* *p*, **int** *ndim*, **int** *nval*, **int** *trunc[]*);

DESCRIPTION

These functions create and destroy multidimensional arrays used to hold policy and value data for QNET programs. They handle the necessary conversion between QNET dimension conventions and C/C++ array conventions.

The function **makePolicyArray**() allocates memory for an *ndim*-dimensional policy array that has action information for *nsrv* servers. The truncation values for each queue are given by the array *trunc[]*. The function **destroyPolicyArray**() frees memory pointed to by *p* previously obtained by **makePolicyArray**().

The function **makeValueArray**() allocates memory for an *ndim*-dimensional array to hold differential cost value data with *nval* different values at each location in the state space. The truncation values for each queue are given by the array *trunc[]*. The function **destroyValueArray**() frees memory pointed to by *p* previously obtained by **makeValueArray**().

DISCUSSION

The **state** of a QNET program is determined by the number of jobs in each of the classes (or queues) used by the program. The **state space** is an *ndim*-dimensional space that is indexed by the number of jobs in each class. For example, if there are two classes and the first class has 5 jobs and the second class has 2 jobs then the ordered pair (5,2) completely specifies a state. In this case the state space is two-dimensional and the value of *ndim* would be 2.

Each state has an **action** associated with it that indicates which classes each of the servers will work on. A **policy** is the entire collection of actions for all states. This means that a policy contains the complete information about which class each server serves for every possible state of the system. For example, in a system with three classes and two servers the action for state (3,2,5) might be (1,3) which indicates that the first server works on class 1 while the second server works on class 3. This means that policy data is indexed by state and a multidimensional array is the appropriate structure to hold this information.

As QNET programs run they compute differential cost data. Each state in the system has one or more **cost values** associated with it and this data is stored in a multidimensional array indexed by state, just as the policy data is.

In C++ array indices start with 0 and so a 5-element array has indices 0 through 4. The convention in QNET programs is to describe the state space by the largest values each class can have, which in turns means the largest index that can be used. This means that QNET state space described by **truncation** values of 30, 30, and 30 is three-dimensional with indices in each dimension having values of 0 through 30 inclusive. The C++ array to hold this data should be 31 by 31 by 31.

If there is more than one server then there will be more than one action for each state and so one additional array dimension in the policy array is required. Similarly, if there is more than one cost value per state then an additional array dimension in the value array is required. While the functions described here can handle this, the user should cast pointers returned by **makePolicyArray**() and **makeValueArray**() appropriately.

RETURN VALUE

The allocation functions each return a pointer that must be cast to a pointer of appropriate level of indirection (dimension). In the case of policy data the level of indirection should be *ndim* if the number of servers *nsrv*=1 but should be increased by one if *nsrv*>1. Similarly, in the case of cost value data the level of

indirection should be *ndim* if *nval*=1 but should be *ndim*+1 if *nval*>1. The deallocation functions do not return any value.

EXAMPLE

Suppose a model has three classes and two servers and that the truncation values for the classes are 10, 20, and 20. Following the QNET convention that N[0] is not used (so N[1] is the truncation value for the first dimension), policy and value arrays for this situation can be allocated with

```
int N[] = {0, 10, 20, 20};
int**** action = (int****) makePolicyArray(3, 2, &N[1]);
double*** h = (double***) makeValueArray(3, 1, &N[1]);
```

Now consider the state x1=5, x2=2, x3=8 (the three queues each have 5, 2, and 8 jobs in them respectively). After these declarations, action[5][2][8] is a two-element array that can hold the policy data for the state (5,2,8); action[5][2][8][0] is the location for the first server's action and action[5][2][8][1] is the location for the second server's action. Notice that the references action[10][20][20][0] and action[10][20][20][1] are both valid. The differential cost value data can be stored in the array h; the value data for the example state can be referenced with h[5][2][8].

The memory allocated for the policy and value arrays can be returned to the system with the commands

```
destroyPolicyArray(action, 3, 2, &N[1]);
destroyValueArray(h, 3, 1, &N[1]);
```

AUTHOR