```
/* newqueue.c - newqueue */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  newqueue  -  Allocate and initialize a queue in the global queue table
 *------------------------------------------------------------------------
 */
qid16   newqueue(void)
{
        static qid16    nextqid=NPROC;  /* Next list in queuetab to use */
        qid16           q;              /* ID of allocated queue        */

        q = nextqid;
        if (q >= NQENT) {               /* Check for table overflow     */
                return SYSERR;
        }

        nextqid += 2;                   /* Increment index for next call*/

        /* Initialize head and tail nodes to form an empty queue */

        queuetab[queuehead(q)].qnext = queuetail(q);
        queuetab[queuehead(q)].qprev = EMPTY;
        queuetab[queuehead(q)].qkey  = MAXKEY;
        queuetab[queuetail(q)].qnext = EMPTY;
        queuetab[queuetail(q)].qprev = queuehead(q);
        queuetab[queuetail(q)].qkey  = MINKEY;
        return q;
}
```

4.10 Perspective

Using a single data structure for process lists makes it possible to create general-purpose linked list manipulation functions, which reduce the size of the code by avoiding duplication. Using an implicit data structure with relative pointers reduces the memory used. For small embedded systems, compacting code and data is necessary. What about systems that have plenty of memory? Interestingly, a general principle applies: unless care is taken, successive generations of software expand to fill whatever memory is available. Thus, thinking carefully about a design is always important: there are never sufficient resources to justify wasteful inefficiency.