something like this:

```
PATH=/usr/local/bin:/bin:/usr/bin:.
```

getenv provides access to any string in the user's environment. To find the current value of the PATH string, for example, we could write

```
char *p = getenv("PATH");
```

p now points to the string "/usr/local/bin:/bin:/usr/bin:.". Be careful with getenv: it returns a pointer to a statically allocated string that may be changed by a later call of the function.

system

The system function allows a C program to run another program (possibly an operating system command). The argument to system is a string containing a command, similar to one that we'd enter at the operating system prompt. For example, suppose that we're writing a program that needs a listing of the files in the current directory. A UNIX program would call system in the following way:

```
system("ls >myfiles");
```

This call invokes the UNIX command ls and asks it to write a listing of the current directory into the file named myfiles.

The return value of system is implementation-defined. system typically returns the termination status code from the program that we asked it to run; testing this value allows us to check whether the program worked properly. Calling system with a null pointer has a special meaning: the function returns a nonzero value if a command processor is available.

Searching and Sorting Utilities

```
void *bsearch(const void *key, const void *base,
              size_t nmemb, size_t size,
              int (*compar)(const void *,
                            const void *));
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

bsearch

The bsearch function searches a sorted array for a particular value (the "key"). When bsearch is called, the key parameter points to the key, base points to the array, nmemb is the number of elements in the array, size is the size of each element (in bytes), and compar is a pointer to a comparison function. The comparison function is similar to the one required by qsort: when passed pointers to the key and an array element (in that order), the function must return a negative, zero, or positive integer depending on whether the key is less than, equal to, or greater than the array element. bsearch returns a pointer to an element that matches the key; if it doesn't find a match, bsearch returns a null pointer.