Bionic Buffalo Tech Note #44:
Using Executable Programs

last revised Thursday 1 October 1998

Supported Platforms

Most Bionic Buffalo products are available in "executable" form. We support the following platforms for most products:

- Solaris (version 2.6) (Sparc).
- Linux (Slackware 3.1, including kernel version 2.0) (x86).
- Windows 95 (any version) and NT (version 4.0) (both x86).

Although we support these specific versions, the software will almost always run on a much wider range of platforms. This is because we use very few special features of the operating systems in our code.

For example, the ELF binaries supported for Linux are compatible with those of many other Unix flavors. We don't use mechanisms such as System V STREAMS or IPC, which aren't always available.

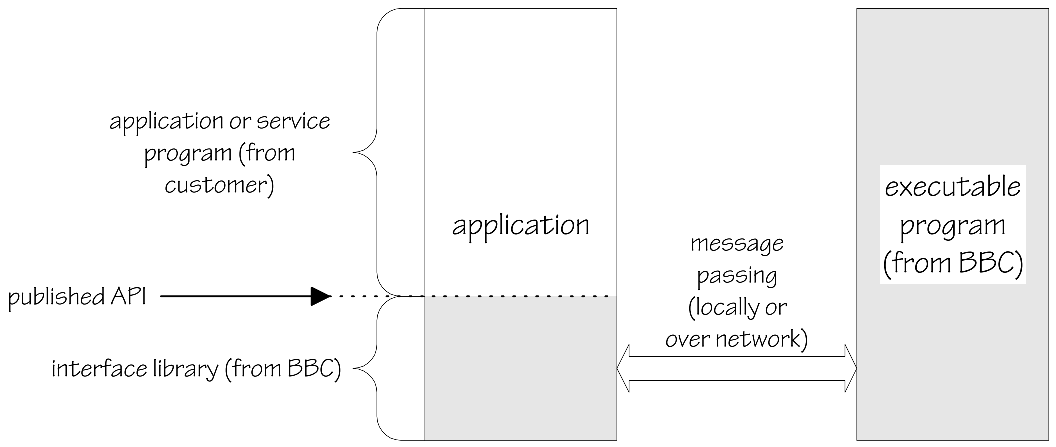
On all platforms for which we support executable code, we use the Berkeley sockets interface for interprocess communication and networking. Your system must have basic TCP/IP installed for most of our software to work.

Unless specifically indicated, our user interfaces are based on command lines and are non-graphical. As explained below, customers can provide their own graphical interfaces and friendly front ends.

Internally, our software uses the Unicode character set. We translate to and from the host character set at all OS interfaces. All of our programs support any language configured on the host platform, including those based on Asian characters.

Application Development

Although our "executable" programs come without sources, you can still write applications using the APIs they support. We provide libraries, which can be linked with your software, and which expose the APIs to your software.


application or service program (from customer)
published API
interface library (from BBC)
application
message passing (locally or over network)
executable program (from BBC)

The libraries are provided as C source code. We do not expect you to change the libraries, so they are uncommented and difficult to read. However, they can be compiled by almost any ANSI C compiler. Since you compile the libraries yourself, there is no specific version required for any particular development environment or toolset.

Each function call in the API is converted to a message, which is sent to the running executable program. A return message provides the output parameters. The libraries use the Berkeley sockets interface to send messages to our programs. You must link your application with the appropriate sockets libraries. Otherwise, we expect only common POSIX procedures in your platform's libraries.

Because we use socket calls, your applications can reside on separate machines from the running executable code. Our libraries take care of byte order and packing differences among platforms. TCP/IP must link the application platforms with the platforms running our executable programs.

The application libraries are provided for the supported platforms. They can be modified for unsupported platforms by the customer, allowing applications to reside on unsupported platforms while our executables run on supported systems. For example, you can host our DSM-CC on an x86 Linux machine, with your own video pump operating on a separate RISC platform.

Customer-Provided Functions

Some software requires the customer to provide certain services to the executable engine. For instance, the customer must provide the video pump for DSM-CC.

Customer-provided functions are built using routines included in the application libraries, which come with the executable code. The customer must provide a separate process for each function. Each service process repeatedly executes a two-step loop:

1. Receive an incoming message requesting a service operation

2. Perform the requested operation

The application libraries include sample source code to build these service processes. The customer must modify the sample code to provide the required service or function.

Evaluating the Porting Kits

The executable software we license is built from our own porting kits. Bionic Buffalo takes a porting kit and hosts the software on one of the supported platforms.

Customers who wish to evaluate the porting kits may do so by trying out the executable versions of the software. Although the porting kits are more flexible and usually have more features, the executable form can give a good idea of quality, capability, and performance.

The porting kits allow the customer to make hosting decisions differently than we have. For example, although we use Berkeley sockets for IPC, a customer might choose to use pipes, STREAMS, message passing, or other mechanisms. These decisions can have a radical affect on performance.

Porting kits allow other options which we do not support in our executable versions. A customer can use the porting kit to host the software on a multiprocessor platform, or using operating systems we do not support directly. Please consult the porting kit documentation for a better idea of the options available with a product.

Processes and Threads

Typically, each executable product runs as several different processes. (Although the porting kits support threading and reentrant code, we do not use it in our own executable forms.)

The architecture is done so that blocking occurs only when waiting for system resources. For this purpose, all resources of a single type are grouped together. For instance, a wait for any single disk file might block execution of a process which needs a different disk file. The executable forms of the software do not implement the parallelism available in the porting kit.

(Note that this grouping of resources may cause delays when networked or remote file systems are used. The executable code does not distinguish between local and remote files, so a network delay might temporarily block access to a local file.)

The application libraries do not implement explicit threading. However, they are thread-safe, so there may be multiple requests outstanding from one or more user application processes. User applications, unlike the executable code we provide, may be threaded. If there is no contention for resources, then the outstanding requests may be processed in parallel.

Some APIs directly support asynchronous request processing. As examples:

- the ***send_deferred*** operation of the ***CORBA::Request*** object allows any local or remote operation on a CORBA object to be performed asynchronously: the ***send_deferred*** operation returns immediately, and the result is retrieved later

- the ***DSM::Config::DeferredSync*** attribute defined by DSM-CC can be used to switch between synchronous and asynchronous behaviour for those operations which are defined to return ***void***

In these cases, the libraries support asynchronous invocation explicitly.

The application libraries support timeouts for synchronous calls. The caller can specify a limit, after which the operation will fail. This avoids the necessity of writing separate threads to prevent an application from blocking indefinitely if a return message should fail to arrive in a timely manner.

IMPLEMENTATION LIMITS

There are few practical implementation limits aside from those of the underlying platforms. We do not constrain the number of outstanding requests or other parameters, although the user may set limits on some values in order to avoid the pathological behaviour of certain operating systems under stress.

EMBEDDED SYSTEMS

There are a great number of embedded environments, including numerous operating systems and hardware platforms. Bionic Buffalo cannot support these directly with executable code. That is one reason we offer porting kits, which can be rehosted to different machines.

Users wishing to host our software on embedded systems (or on other platforms which we do not support directly with executables) have two choices:

- the customer may license a porting kit, and host (or have us host) the product on the target environment; or

- the customer may find a third party who has already hosted the product on the desired environment, and may license an object-code version from the third party (The third party must have an appropriate license agreement with Bionic Buffalo to allow this.)

In the future, we plan to support executable code on one or more public-domain embedded platforms. However, we do not currently offer this option.