V8 JavaScript Engine on Fedora 14

Finnbarr P. Murphy
(fpm@fpmurphy.com)

The recent announcement by Google of a new compilation infrastructure, aptly codenamed CrankShaft, for their V8 JavaScript engine prompted me to revisit this particular JavaScript engine and to compile and install it on a Fedora 14 X64 platform.

Crankshaft is the latest iteration of the V8 JavaScript (V8) engine and is somewhat similar to Mozilla's TraceMonkey JavaScript engine in that it attempts to aggressively optimize code that is regularly executed but avoid much optimization of code that is infrequently executed.

Currently the Mozilla Spidermonkey JavaScript shell is available as a standard RPM package for Fedora 14. In a previous post I showed how to build the Mozilla Tracemonkey JavaScript shell on Fedora 14. In this post I show how to build a V8 shell on Fedora 14.

Why build a V8 shell? Simply put, V8 is far faster than Tracemonkey. There are three key areas to it's superior performance:

- Fast Property Access
- Dynamic Machine Code Generation
- Efficient Garbage Collection

Read the V8 Design Elements web page for a detailed overview of these key areas.

V8 is written in C++ and makes extensive use of C++ namespaces. The V8 project is hosted at Google Code. Use the following *svn* command to anonymously check out the latest up-to-date stable version of the V8 source code using *svn*:

```
$ svn checkout http://v8.googlecode.com/svn/trunk/ v8-read-only
```

V8 uses the popular opensource SCons software construction tool for building the various components of the project. If SCons is not installed on your platform (which it is not by default), it is available as a RPM from the Fedora standard repositories. SCons requires at least Python 2.4. If you have not used SCons before, you will be pleasantly surprised with it. A hidden gem!

Enter the following commands to build a V8 shell:

```
$ cd v8-read-only
$ scons
```

Add *arch=x64 d8* argument to *scons* if you are on an X64 platform. Add *mode=debug* if you want a debugging version of the shell. The shell that will be built is a developer's shell called *d8*.

While SCons will successfully build the *d8* shell, the shell hangs when you attempt to use it. The solution appears to be to change all the *gcc* compiler optimization switches from *-O3* to *-O2* in the *SConstruct* file. SConstruct is similar in purpose to a Makefile and is the input file that SCons reads to control the build.

```
    'mode:release': {
```



```
      'CCFLAGS':      ['-O2']
    },
    .....
    'mode:release': {
      'CCFLAGS':      ['/O2'],
      ...
      },
```

Change all the *mode:release* CCFLAGS to *02*.

To clean the build system, invoke *scons* with the *–clean* option:

```
$ scons --clean
```

Another option available in the *d8* shell is support for command history and completions using the GNU *readline* library. To build the *d8* shell with command history support

```
$ scons arch=x64 console=readline d8
```

The current V8 shells do not support the *#!* (shebang AKA hashbang) interpreter directive functionality. The following modification to *../src/d8.cc* enables JavaScript scripts to be directly invoked using this functionality.

```
$ diff src/d8.cc.org src/d8.cc
699a700,737
>
>     FILE* file = i::OS::FOpen(argv[1], "rb");
>     if (file) {
>         int ch = fgetc(file);
>         int ch1 = fgetc(file);
>         if (ch == '#' &amp;amp;&amp;amp; ch1 == '!') {    // #! shebang
>             // figure out size of file
>             fseek(file, 0, SEEK_END);
>             int size = ftell(file);
>             rewind(file);
>
>             // go to start of second line
>             while((ch = fgetc(file)) != EOF) {
>               size--;
>               if (ch == '\n' || ch == '\r') break;
>             }
>
>             // copy remainder of script to char[]
>             char* chars = new char[size + 1];
>             for (int i = 0; i < size;) {
>                 int read = fread(&amp;chars[i], 1, size - i, file);
>                 i += read;
>             }
>             chars[size] = '&#92;&#48;';
>             fclose(file);
>
>             // now execute script in char[]
>             v8::HandleScope handle_scope;
>             v8::Handle<v8::String> file_name = v8::String::New("unnamed");
>             v8::Handle<v8::String> source = v8::String::New(chars);
>             if (ExecuteString(source, file_name, false, true)) {
>                 OnExit();
>                 return 0;
>             }
```



```
>         }
>         fclose(file);
>     }
>
```

Here is a simple JavaScript script (*fib.js*) that uses the *d8* shell (to which */usr/bin/d8js* is linked) and the *shebang* functionality. Just like a shell script, it can be invoked as *./fib.js* or with its full pathname.

```
#! /usr/bin/d8js
function fibonacci(num) {
   var curnum = 1
   var prevnum = 0;
   for(i=0;i<=num;i++){
      curnum = curnum + prevnum;
      prevnum = curnum;
   }
   print("Fibonacci number for: " + num + " is: " + curnum);
}
while (1) {
   write("Enter number greater than 1, or 0 to exit: ");
   if (( num = readline()) == 0)
       break;
   fibonacci(num);
}
```

Note the use of *write()* to output a prompt prior to reading in a value using *readline()*.

Here is how to build the regular V8 shell on an X64 platform:

```
$ scons arch=x64 sample=shell
```

Like the *d8* shell it does not support *shebang* functionaility nor does it support command history or command completion.

I modified ../samples/shell.cc to add *shebang* functionality and command history support. Initially I intended adding command completion support but dropped it due to lack of time. It turns out that command completion is implemented in an interesting way in the *d8* shell. It is actually implemented in JavaScript (see *GetCompletions* function *../src/d8.js*) and then converted into C-style char arrays (*natives*) using the Python script *../tools/js2c.py*. The c++ source file that is created by this tool and which is subsequently complied and linked in is *../obj/../libraries.cc*.

Here are the necessary diffs to *../samples/shell.cc* to support *shebang* and command line history:

```
diff samples/shell.cc.org samples/shell.cc.new
28c28,29
< #include <v8.h>
---
> #include "../src/d8.h"
>
34,36c35,45
< void RunShell(v8::Handle<v8::Context> context);
< bool ExecuteString(v8::Handle<v8::String> source,
<                    v8::Handle<v8::Value> name,
---
> #include <cstdio>
> #include <readline/readline.h>
```



```
> #include <readline/history.h>
>
> namespace v8 {
>
> namespace i = internal;
>
> void RunShell(Handle<Context> context);
> bool ExecuteString(Handle<String> source,
>                    Handle<Value> name,
39,45c48,159
< v8::Handle<v8::Value> Print(const v8::Arguments&amp; args);
< v8::Handle<v8::Value> Read(const v8::Arguments&amp; args);
< v8::Handle<v8::Value> Load(const v8::Arguments&amp; args);
< v8::Handle<v8::Value> Quit(const v8::Arguments&amp; args);
< v8::Handle<v8::Value> Version(const v8::Arguments&amp; args);
< v8::Handle<v8::String> ReadFile(const char* name);
< void ReportException(v8::TryCatch* handler);
---
> Handle<Value> Print(const Arguments&amp; args);
> Handle<Value> Read(const Arguments&amp; args);
> Handle<Value> Write(const Arguments&amp; args);
> Handle<Value> Load(const Arguments&amp; args);
> Handle<Value> Quit(const Arguments&amp; args);
> Handle<Value> Version(const Arguments&amp; args);
> Handle<String> ReadFile(const char* name);
> Handle<Value> ReadLine(const Arguments&amp; args);
> void ReportException(TryCatch* handler);
>
> // ----------- readline support
> class ReadLineEditor: public LineEditor {
>  public:
>   ReadLineEditor() : LineEditor(LineEditor::READLINE, "readline") { }
>   virtual i::SmartPointer<char> Prompt(const char* prompt);
>   virtual bool Open();
>   virtual bool Close();
>   virtual void AddHistory(const char* str);
> };
>
>
> const char* kHistoryFileName = ".v8_history";
> const char* kPrompt = "v8> ";
> Persistent<Context> utility_context_;
> Persistent<Context> evaluation_context_;
> LineEditor *LineEditor::first_ = NULL;
>
>
> LineEditor::LineEditor(Type type, const char* name)
>     : type_(type),
>       name_(name),
>       next_(first_) {
>   first_ = this;
> }
>
>
> LineEditor* LineEditor::Get() {
>   LineEditor* current = first_;
>   LineEditor* best = current;
>   while (current != NULL) {
>     if (current->type_ > best->type_)
>       best = current;
>     current = current->next_;
>   }
>   return best;
> }
>
>
> class DumbLineEditor: public LineEditor {
>  public:
```



```
>   DumbLineEditor() : LineEditor(LineEditor::DUMB, "dumb") { }
>   virtual i::SmartPointer<char> Prompt(const char* prompt);
> };
>
>
> i::SmartPointer<char> DumbLineEditor::Prompt(const char* prompt) {
>   static const int kBufferSize = 256;
>   char buffer[kBufferSize];
>   printf("%s", prompt);
>   char* str = fgets(buffer, kBufferSize, stdin);
>   return i::SmartPointer<char>(str ? i::StrDup(str) : str);
> }
>
>
> static ReadLineEditor read_line_editor;
>
> bool ReadLineEditor::Open() {
>   rl_initialize();
>   using_history();
>   return read_history(kHistoryFileName) == 0;
> }
>
>
> bool ReadLineEditor::Close() {
>   return write_history(kHistoryFileName) == 0;
> }
>
>
> i::SmartPointer<char> ReadLineEditor::Prompt(const char* prompt) {
>   char* result = readline(prompt);
>   return i::SmartPointer<char>(result);
> }
>
>
> void ReadLineEditor::AddHistory(const char* str) {
>   add_history(str);
> }
>
> static char* ReadToken(char* data, char token) {
>   char* next = i::OS::StrChr(data, token);
>   if (next != NULL) {
>     *next = '&#92;&#48;';
>     return (next + 1);
>   }
>
>   return NULL;
> }
>
>
> Handle<Value> ReadLine(const Arguments&amp; args) {
>   i::SmartPointer<char> line(i::ReadLine(""));
>   if (*line == NULL) {
>     return Null();
>   }
>   size_t len = strlen(*line);
>   if (len > 0 &amp;&amp; line[len - 1] == '\n') {
>     --len;
>   }
>   return String::New(*line, len);
> }
>
> // ----------- readline support
49,62c163
<   v8::V8::SetFlagsFromCommandLine(&amp;argc, argv, true);
<   v8::HandleScope handle_scope;
<   // Create a template for the global object.
<   v8::Handle<v8::ObjectTemplate> global = v8::ObjectTemplate::New();
<   // Bind the global 'print' function to the C++ Print callback.
```



```
<   global->Set(v8::String::New("print"), v8::FunctionTemplate::New(Print));
<   // Bind the global 'read' function to the C++ Read callback.
<   global->Set(v8::String::New("read"), v8::FunctionTemplate::New(Read));
<   // Bind the global 'load' function to the C++ Load callback.
<   global->Set(v8::String::New("load"), v8::FunctionTemplate::New(Load));
<   // Bind the 'quit' function
<   global->Set(v8::String::New("quit"), v8::FunctionTemplate::New(Quit));
<   // Bind the 'version' function
<   global->Set(v8::String::New("version"), v8::FunctionTemplate::New(Version));
---
>   V8::SetFlagsFromCommandLine(&amp;argc, argv, true);
63a165,167
>   // FPM HandleScope handle_scope;
>   HandleScope handle;
>   Handle<ObjectTemplate> global = ObjectTemplate::New();
64a169,176
>   global->Set(String::New("print"), FunctionTemplate::New(Print));
>   global->Set(String::New("read"), FunctionTemplate::New(Read));
>   global->Set(String::New("write"), FunctionTemplate::New(Write));
>   global->Set(String::New("load"), FunctionTemplate::New(Load));
>   global->Set(String::New("quit"), FunctionTemplate::New(Quit));
>   global->Set(String::New("version"), FunctionTemplate::New(Version));
>   global->Set(String::New("readline"), FunctionTemplate::New(ReadLine));
>
67c179
<   v8::Handle<v8::Context> context = v8::Context::New(NULL, global);
---
>   Handle<Context> context = Context::New(NULL, global);
69c181
<   v8::Context::Scope context_scope(context);
---
>   Context::Scope context_scope(context);
71a184,219
>   FILE* file = fopen(argv[1], "rb");
>   if (file) {
>         int ch = fgetc(file);
>         int ch1 = fgetc(file);
>         if (ch == '#' &amp;&amp; ch1 == '!') {    // #! shebang
>             // figure out size of file
>             fseek(file, 0, SEEK_END);
>             int size = ftell(file);
>             rewind(file);
>
>             // go to start of second line
>             while((ch = fgetc(file)) != EOF) {
>                size--;
>                if (ch == '\n' || ch == '\r') break;
>             }
>
>             // copy remainder of script to char[]
>             char* chars = new char[size + 1];
>             for (int i = 0; i < size;) {
>                 int read = fread(&amp;chars[i], 1, size - i, file);
>                 i += read;
>             }
>             chars[size] = '&#92;&#48;';
>             fclose(file);
>
>             // now execute script in char[]
>             HandleScope handle_scope;
>             Handle<String> file_name = String::New("unnamed");
>             Handle<String> source = String::New(chars);
>             if (ExecuteString(source, file_name, false, true)) {
>                 return 0;
>             }
>         }
>         fclose(file);
>     }
```



```
>
84,86c232,234
<       v8::HandleScope handle_scope;
<       v8::Handle<v8::String> file_name = v8::String::New("unnamed");
<       v8::Handle<v8::String> source = v8::String::New(argv[i + 1]);
---
>       HandleScope handle_scope;
>       Handle<String> file_name = String::New("unnamed");
>       Handle<String> source = String::New(argv[i + 1]);
92,94c240,242
<       v8::HandleScope handle_scope;
<       v8::Handle<v8::String> file_name = v8::String::New(str);
<       v8::Handle<v8::String> source = ReadFile(str);
---
>       HandleScope handle_scope;
>       Handle<String> file_name = String::New(str);
>       Handle<String> source = ReadFile(str);
107,114d254
<
< int main(int argc, char* argv[]) {
<   int result = RunMain(argc, argv);
<   v8::V8::Dispose();
<   return result;
< }
<
<
116c256
< const char* ToCString(const v8::String::Utf8Value&amp; value) {
---
> const char* ToCString(const String::Utf8Value&amp; value) {
124c264
< v8::Handle<v8::Value> Print(const v8::Arguments&amp; args) {
---
> Handle<Value> Print(const Arguments&amp; args) {
127c267
<     v8::HandleScope handle_scope;
---
>     HandleScope handle_scope;
133c273
<     v8::String::Utf8Value str(args[i]);
---
>     String::Utf8Value str(args[i]);
139c279
<   return v8::Undefined();
---
>   return Undefined();
146c286
< v8::Handle<v8::Value> Read(const v8::Arguments&amp; args) {
---
> Handle<Value> Read(const Arguments&amp; args) {
148c288
<     return v8::ThrowException(v8::String::New("Bad parameters"));
---
>     return ThrowException(String::New("Bad parameters"));
150c290
<   v8::String::Utf8Value file(args[0]);
---
>   String::Utf8Value file(args[0]);
152c292
<     return v8::ThrowException(v8::String::New("Error loading file"));
---
>     return ThrowException(String::New("Error loading file"));
154c294
<   v8::Handle<v8::String> source = ReadFile(*file);
---
>   Handle<String> source = ReadFile(*file);
156c296
<     return v8::ThrowException(v8::String::New("Error loading file"));
```



```
---
>     return ThrowException(String::New("Error loading file"));
161a302,318
> Handle<Value> Write(const Arguments&amp; args) {
>   for (int i = 0; i < args.Length(); i++) {
>     HandleScope handle_scope;
>     if (i != 0) {
>       printf(" ");
>     }
>     String::Utf8Value str(args[i]);
>     int n = fwrite(*str, sizeof(**str), str.length(), stdout);
>     if (n != str.length()) {
>       printf("Error in fwrite\n");
>       exit(1);
>     }
>   }
>   return Undefined();
> }
>
>
165c322
< v8::Handle<v8::Value> Load(const v8::Arguments&amp; args) {
---
> Handle<Value> Load(const Arguments&amp; args) {
167,168c324,325
<     v8::HandleScope handle_scope;
<     v8::String::Utf8Value file(args[i]);
---
>     HandleScope handle_scope;
>     String::Utf8Value file(args[i]);
170c327
<       return v8::ThrowException(v8::String::New("Error loading file"));
---
>       return ThrowException(String::New("Error loading file"));
172c329
<     v8::Handle<v8::String> source = ReadFile(*file);
---
>     Handle<String> source = ReadFile(*file);
174c331
<       return v8::ThrowException(v8::String::New("Error loading file"));
>       return ThrowException(String::New("Error loading file"));
176,177c333,334
<     if (!ExecuteString(source, v8::String::New(*file), false, false)) {
<       return v8::ThrowException(v8::String::New("Error executing file"));
---
>     if (!ExecuteString(source, String::New(*file), false, false)) {
>       return ThrowException(String::New("Error executing file"));
180c337
<   return v8::Undefined();
---
>   return Undefined();
186c343
< v8::Handle<v8::Value> Quit(const v8::Arguments&amp; args) {
---
> Handle<Value> Quit(const Arguments&amp; args) {
191c348
<   return v8::Undefined();
---
>   return Undefined();
195,196c352,353
< v8::Handle<v8::Value> Version(const v8::Arguments&amp; args) {
<   return v8::String::New(v8::V8::GetVersion());
---
> Handle<Value> Version(const Arguments&amp; args) {
>   return String::New(V8::GetVersion());
201c358
< v8::Handle<v8::String> ReadFile(const char* name) {
---
```



```
> Handle<String> ReadFile(const char* name) {
203c360
<   if (file == NULL) return v8::Handle<v8::String>();
---
>   if (file == NULL) return Handle<String>();
216c373
<   v8::Handle<v8::String> result = v8::String::New(chars, size);
---
>   Handle<String> result = String::New(chars, size);
223,228c380,383
< void RunShell(v8::Handle<v8::Context> context) {
<  // LineEditor* editor = LineEditor::Get();
<  //  printf("V8 version %s [console: %s]\n", v8::V8::GetVersion(), editor->name());
<
<   // printf("V8 version %s\n", v8::V8::GetVersion());
<   static const int kBufferSize = 256;
---
> void RunShell(Handle<Context> context) {
>   LineEditor* editor = LineEditor::Get();
>   printf("V8 version %s [console: %s]\n", V8::GetVersion(), editor->name());
>   editor->Open();
230,238c385,390
<     char buffer[kBufferSize];
<     printf("> ");
<     char* str = fgets(buffer, kBufferSize, stdin);
<     if (str == NULL) break;
<     v8::HandleScope handle_scope;
<     ExecuteString(v8::String::New(str),
<                   v8::String::New("(shell)"),
<                   true,
<                   true);
---
>     i::SmartPointer<char> input = editor->Prompt(kPrompt);
>     if (input.is_empty())
>       break;
>     editor->AddHistory(*input);
>     Handle<String> name = String::New("(v8)");
>     ExecuteString(String::New(*input), name, true, true);
239a392
>   editor->Close();
245,246c398,399
< bool ExecuteString(v8::Handle<v8::String> source,
<                    v8::Handle<v8::Value> name,
---
> bool ExecuteString(Handle<String> source,
>                    Handle<Value> name,
249,251c402,404
<   v8::HandleScope handle_scope;
<   v8::TryCatch try_catch;
<   v8::Handle<v8::Script> script = v8::Script::Compile(source, name);
---
>   HandleScope handle_scope;
>   TryCatch try_catch;
>   Handle<Script> script = Script::Compile(source, name);
258c411
<     v8::Handle<v8::Value> result = script->Run();
---
>     Handle<Value> result = script->Run();
268c421
<         v8::String::Utf8Value str(result);
---
>         String::Utf8Value str(result);
278,280c431,433
< void ReportException(v8::TryCatch* try_catch) {
<   v8::HandleScope handle_scope;
<   v8::String::Utf8Value exception(try_catch->Exception());
---
> void ReportException(TryCatch* try_catch) {
```



```
>   HandleScope handle_scope;
>   String::Utf8Value exception(try_catch->Exception());
282c435
<   v8::Handle<v8::Message> message = try_catch->Message();
---
>   Handle<Message> message = try_catch->Message();
289c442
<     v8::String::Utf8Value filename(message->GetScriptResourceName());
---
>     String::Utf8Value filename(message->GetScriptResourceName());
294c447
<     v8::String::Utf8Value sourceline(message->GetSourceLine());
---
>     String::Utf8Value sourceline(message->GetSourceLine());
307c460
<     v8::String::Utf8Value stack_trace(try_catch->StackTrace());
---
>     String::Utf8Value stack_trace(try_catch->StackTrace());
313a467,474
>
> }  // namespace V8
>
> int main(int argc, char* argv[]) {
>   int result = v8::RunMain(argc, argv);
>   v8::V8::Dispose();
>   return result;
> }
```

To support the changes name in *../samples/shell.cc,* the toplevel *SConstruct* file also has to be modified to add in *../src* as an include directory and to add *libreadline* to the list of libraries to be linked in.

```
$ diff SConstruct.org  SConstruct,new
453c453
<     'CPPPATH': [join(abspath('.'), 'include')],
---
>     'CPPPATH': [join(abspath('.'), 'include'), join(abspath('.'), 'src')]
461c461
<       'LIBS':         ['pthread'],
---
>       'LIBS':         ['pthread', 'readline'],
```

Here is another simple JavaScript script (*des.js*) which imports (loads) another Javascript file (*des-inc.js*) containing the actual DES encryption routines and then performs a TripleDES encryption on a user supplied string using a user supplied key. It uses */usr/bin/v8js* which is linked to to the *samples* shell.

```
#! /usr/bin/v8js
load("./des-inc.js");
function stringToHex (s) {
   var hexes = new Array ("0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f")
;
   var r="";
   for (var i=0; i<s.length; i++) {
       r += hexes [s.charCodeAt(i) >> 4] + hexes [s.charCodeAt(i) &amp;amp; 0xf];
   }
   return r;
}
var key = "1234567890123456ABCDEFGH";
var plaintext = "The quick brown fox jumped over the lazy dog";
while (true) {
   write("Enter 24 character key or 0 to exit [" + key + "]: ");
```



```
   input = readline();
   if (input == "0") quit();
   if (input.length == 0) break;
   if (input.length == 24) {
      key = input;
      break;
   }
}
while (true) {
   write("Enter plaintext or 0 to exit [" + plaintext + "]: ");
   input = readline();
   if (input == "0") quit();
   if (input.length == 0) break;
   if (input.length >= 24) {
      plaintext = input;
      break;
   }
}
var ciphertext = des(key, plaintext, 1, 0);
print("Ciphertext: " + stringToHex(ciphertext));
```

I did not write the JavaScript code for the DES encryption routines in *des-inc.js*. The original author of these routines was Paul Tero, a web programmer based in Brighton, England.

Here is sample output from *des.js*:

```
$ ./des.js
Enter 24 character key or 0 to exit [1234567890123456ABCDEFGH]: 123456789012345678901234
Enter plaintext or 0 to exit [The quick brown fox jumped over the lazy dog]:
Ciphertext: 01dfc133e39f20cce77732a2bb45838ee4020248ce8866b685d49042ac85234a26a08ed18cde40
e33c20206b54e6af9f
$
```

Well that is all for now. If I get some spare time, I would like to implement *readline* completions. The main problem will be modifying the *SConstruct* and *SConfig* build configurations files.

P.S. The V8 codebase is updated frequently so some work may be needed to integrate my changes into a more recent snapshot of the codebase. The trunk snapshot which I used was dated December 12th, 2010. If you want to use a tagged version rather than pull from the trunk, that is now possible, e.g. to get the v8-2.5.9 snapshot:

```
$ svn export http://v8.googlecode.com/svn/tags/2.5.9/ v8-2.5.9
```


