

Forum:[Hack Forums](#)

Thread: [I need help on using the Java Native Interface \(JNI\)](#)

BinaryBit - June 29, 2015, 1:05 pm

I want to make some mods for a popular game known as "Minecraft". However, I hate the Java programming language, but do know a lot about it. I also know a lot of the C programming language, and would like to make some native methods to process my functions in C. I tried looking online, but I could not understand any of the tutorials. If anyone here is experienced with Java, C, and the JNI, please help me on setting it up, and using it. Thank-you.

WildeThing - June 29, 2015, 2:14 pm

You're a fucking idiot if you think it's a good idea to have a Minecraft client that uses any form of native bridge. Minecraft is Java, you wanna write mods for it? Write them in Java. I'd love to see your attempts at writing process memory in the JVM address space or whatever fucking Windows C exploit your gonna pull for this one.

Anyways, it's really easy and I'm surprised a self-acclaimed C programmer like you can't do it. Fail. All I can say is: read the tutorials again, it's really very simple.

JavaProfit - June 29, 2015, 2:22 pm

Just learn Java, I don't understand why anyone would make mods with native code unless it is like some advanced graphics shit that you can't get lwjgl to do for you.

Besides that, there's the `System.loadLibrary` method which can load in your native mods (I think Java 8 allows static linking, but I would go with dynamic links cas you should do that). You can use `javah` to generate header files.

BinaryBit - June 30, 2015, 1:09 pm

(Jun 29, 2015 02:14 PM)WildeThing Wrote: [\[CITE\]](#) You're a fucking idiot if you think it's a good idea to have a Minecraft client that uses any form of native bridge. Minecraft is Java, you wanna write mods for it? Write them in Java. I'd love to see your attempts at writing process memory in the JVM address space or whatever fucking Windows C exploit your gonna pull for this one.

Anyways, it's really easy and I'm surprised a self-acclaimed C programmer like you can't do it. Fail. All I can say is: read the tutorials again, it's really very simple.

No need to be rude. Also, all of the libraries in Java lead up to C or C++. Without natives, you couldn't do anything in Java, really. You're not very smart for not thinking about that. I can't do it, because I cannot understand any of the tutorials online. And, if it's very simple, please help me out, as that is the point of this thread. Thank-you.

JavaProphet - June 30, 2015, 1:20 pm

JNI is very useful, however it's cons outweigh it's pros unless it is something that can be improved by faster native processing(ie a String Replace is 144x faster than Java in C), or if you need to do native calls(in Linux, `setuid`, etc). You may be able to use it in this case, but the overhead of JNI will be very bad for performance in such a high-throughput way. In rendering, you'll likely see issues with OpenGL context. You really should never mix and match unless it's either of these two cases.

If you still want to do it, I always thought it was harder than it really was. Run `javah` on a file containing methods with native flags, fill in a C/C++ file for them, compile, and then do `System.load/loadLibrary` on the file.

JNI has a considerable invocation overhead, you can see specific benchmarks online, or run them yourself to put them in perspective.

BinaryBit - June 30, 2015, 1:24 pm

(Jun 30, 2015 01:20 PM)JavaProphet Wrote: [\[CITE\]](#) JNI is very useful, however it's cons outweigh it's pros unless it is something that can be improved by faster native processing(ie a String Replace is 144x faster than Java in C), or if you need to do native calls(in Linux, setuid, etc). You may be able to use it in this case, but the overhead of JNI will be very bad for performance in such a high-throughput way. In rendering, you'll likely see issues with OpenGL context. You really should never mix and match unless it's either of these two cases.

If you still want to do it, I always thought it was harder than it really was. Run javah on a file containing methods with native flags, fill in a C/C++ file for them, compile, and then do System.load/loadLibrary on the file.

JNI has a considerable invocation overhead, you can see specific benchmarks online, or run them yourself to put them in perspective.

Thank-you for being the only helpful (and respectful) individual on HackForums to me. I never knew that it had an overhead. I tried your steps, and they worked. I will stick to Java for modifying Java software. Again, thanks a lot. You taught me something today.

peaksoonpeak - June 30, 2015, 1:50 pm

(Jun 30, 2015 01:20 PM)JavaProphet Wrote: [\[CITE\]](#) JNI is very useful, however it's cons outweigh it's pros unless it is something that can be improved by faster native processing(ie a String Replace is 144x faster than Java in C), or if you need to do native calls(in Linux, setuid, etc). You may be able to use it in this case, but the overhead of JNI will be very bad for performance in such a high-throughput way. In rendering, you'll likely see issues with OpenGL context. You really should never mix and match unless it's either of these two cases.

If you still want to do it, I always thought it was harder than it really was. Run javah on a file containing methods with native flags, fill in a C/C++ file for them, compile, and then do System.load/loadLibrary on the file.

JNI has a considerable invocation overhead, you can see specific benchmarks online, or run them yourself to put them in perspective. There are no string replaces functions in C.

It's up to the programmer himself to optimize the function for 'string replacing' which is really a subset of finding individual bytes in memory and comparing them with the comparison operand. memcmp is one way of coming by it, but using bitwise operations and assembly you can decrease the amount of time it takes to find it, also keep in mind with memcmp's return value. That also plays a part.

JavaProphet - June 30, 2015, 1:51 pm

@peak Yeah, that's implicit. I ported my C algorithm into Java as closely as possible, and it was 6x faster then.

peaksoonpeak - June 30, 2015, 2:07 pm

(Jun 30, 2015 01:51 PM)JavaProphet Wrote: [\[CITE\]](#) @peak Yeah, that's implicit. I ported my C algorithm into Java as closely as possible, and it was 6x faster then.

You have to also take into account the fact that all code that is ran by the machine, is understood by the machine. The JVM optimizes the byte code into it's machine code equivalent.

It's all dependant on the programmer himself to create a faster algorithm, not the language.

I've written sorting algorithms in assembler and c that far exceed the speed of java's equivalents.

JavaProfit - June 30, 2015, 3:27 pm

(Jun 30, 2015 01:51 PM)JavaProphet Wrote: [CITE](#) @peak Yeah, that's implicit. I ported my C algorithm into Java as closely as possible, and it was 6x faster then.

What algorithm are you talking about? If you just use mutable strings in Java it's going to be faster no matter what secret string replacement algo you use.

JavaProphet - June 30, 2015, 4:00 pm

Mutable strings are always faster, as it doesn't require as much reallocation, memory copying, etc. My algorithm ported from C to Java only worked on char arrays, and I just wrote a wrapper for immutable strings. I've personally come more and more to loathe immutable strings, as they are a real performance drain. A 1kb immutable string is horrible to do parsing on. All C 'strings' are mutable, unless you enforce immutability on your own code.

C replace:

Spoiler ([Click to View](#))Code char* replace(char* input, char* rf, char* rt) {

```
    size_t isl = strlen(input);
    size_t rfsl = strlen(rf);
    size_t rtsl = strlen(rt);
    size_t ml = 0;
    size_t mc = 0;
    int i = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                mc++;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
    }
    i = 0;
    ml = 0;
    char* output = calloc(isl - ((rfsl - rtsl) * mc), sizeof(char));
    int mi = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                size_t os = mi - ml + 1;
                strcpy(output + os, rt);
                if (ml > rtsl) {
                    memset(output + os + (ml - rtsl), 0, ml - rtsl);
                }
                mi += ml - rtsl - 1;
                ml = 0;
            }
        }
    }
}
```

```

        continue;
    }
} else if (ml > 0) {
    ml = 0;
}
output[mi++] = input[i];
}
return output;
}

```

Java replace(custom):

Spoiler ([Click to View](#))

```

Code public static char[] replace(char[] input, char[] rf, char[] rt) {
    int isl = input.length;
    int rfsl = rf.length;
    int rtsl = rt.length;
    int ml = 0;
    int mc = 0;
    for (int i = 0; i < isl; i++) {
        boolean im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                mc++;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
    }
    ml = 0;
    char[] output = new char[isl - ((rfsl - rtsl) * mc)];
    int mi = 0;
    for (int i = 0; i < isl; i++) {
        boolean im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                int os = mi - ml + 1;
                System.arraycopy(rt, 0, output, os, rt.length);
                if (ml > rtsl) {
                    int ofs = ml - rtsl;
                    int oi = os + ofs;
                    if (oi + ofs < output.length) System.arraycopy(new char[ofs], 0, output, oi, ofs);
                }
                mi += ml - rtsl - 1;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
    }
}

```

```

        output[mi++] = input[i];
    }
    return output;
}

```

peaksoonpeak - June 30, 2015, 4:04 pm

(Jun 30, 2015 04:00 PM)JavaProphet Wrote: [CITE](#) Mutable strings are always faster, as it doesn't require as much reallocation, memory copying, etc. My algorithm ported from C to Java only worked on char arrays, and I just wrote a wrapper for immutable strings. I've personally come more and more to loathe immutable strings, as they are a real performance drain. A 1kb immutable string is horrible to do parsing on. All C 'strings' are mutable, unless you enforce immutability on your own code.

C replace:

Spoiler ([Click to View](#))Code char* replace(char* input, char* rf, char* rt) {

```

    size_t isl = strlen(input);
    size_t rfsl = strlen(rf);
    size_t rtsl = strlen(rt);
    size_t ml = 0;
    size_t mc = 0;
    int i = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                mc++;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
    }
    i = 0;
    ml = 0;
    char* output = calloc(isl - ((rfsl - rtsl) * mc), sizeof(char));
    int mi = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                size_t os = mi - ml + 1;
                strcpy(output + os, rt);
                if (ml > rtsl) {
                    memset(output + os + (ml - rtsl), 0, ml - rtsl);
                }
                mi += ml - rtsl - 1;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {

```

```

        ml = 0;
    }
    output[mi++] = input[i];
}
return output;
}

```

Java replace(custom):

Spoiler ([Click to View](#))

```

Code public static char[] replace(char[] input, char[] rf, char[] rt) {
    int isl = input.length;
    int rfsl = rf.length;
    int rtsl = rt.length;
    int ml = 0;
    int mc = 0;
    for (int i = 0; i < isl; i++) {
        boolean im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                mc++;
                ml = 0;
                continue;
            }
        }else if (ml > 0) {
            ml = 0;
        }
    }
    ml = 0;
    char[] output = new char[isl - ((rfsl - rtsl) * mc)];
    int mi = 0;
    for (int i = 0; i < isl; i++) {
        boolean im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                int os = mi - ml + 1;
                System.arraycopy(rt, 0, output, os, rt.length);
                if (ml > rtsl) {
                    int ofs = ml - rtsl;
                    int oi = os + ofs;
                    if (oi + ofs < output.length) System.arraycopy(new char[ofs], 0, output, oi, ofs);
                }
                mi += ml - rtsl - 1;
                ml = 0;
                continue;
            }
        }else if (ml > 0) {
            ml = 0;
        }
        output[mi++] = input[i];
    }
    return output;
}

```

```
}
```

There are many redundancies in your C representation.

calculating null terminated strings in the function itself is redundant (pass them onto the stack).

Your replace function should

1. Locate the string you want to replace in memory
2. Overwrite that string.

Instead of reallocating memory for the buffer block (What if this is a local buffer? undefined behaviour...)

You would want to copy the entire string onto the heap, along with the new size + (New string to replace length - initial string to replace length)

Something along the lines of this

```
Code BOOL CompareMemoryWithMask(unsigned char* ucSignature, DWORD
dwAddress,DWORD dwSig)
```

```
{
    for (; dwSig > 0; --dwSig, ++ucSignature, ++dwAddress)
    {
        if (*ucSignature != '?' && *ucSignature != *(unsigned char*)dwAddress)
            return FALSE;
    }
    return TRUE;
}
```

```
DWORD FindMemory(unsigned char* ucSignature, DWORD dwModuleBase, DWORD
dwModuleSize,DWORD dwSig)
```

```
{
    DWORD i = 0;
    for (i = 0; i < dwModuleSize; i++)
    {
        if (CompareMemoryWithMask(ucSignature,dwModuleBase+i,dwSig))
        {
            return dwModuleBase+i;
        }
    }
    return 0;
}
```

Compile with intrinsic option, or you can create an intrinsic function.

For the signature you'd pass the string you want to find the buffer, the module size is the buffer size, the module base is the start of the buffer, dwSig is the string length you want to find in the buffer.

JavaProphet - June 30, 2015, 4:08 pm

(Jun 30, 2015 04:04 PM)peaksoonpeak Wrote: [\[CITE\]](#) There are many redundancies in your C representation.

calculating null terminated strings in the function itself is redundant (pass them onto the stack).

Your replace function should

1. Locate the string you want to replace in memory
2. Overwrite that string.

Instead of reallocating memory for the buffer block (What if this is a local buffer? undefined behaviour...)

You would want to copy the entire string onto the heap, along with the new size + (New string to replace length - initial string to replace length)

Not a bad idea. I'm still new to C. Thanks!

EDIT: What exactly do you mean by calculating the null terminated string? The lengths?
How exactly do I copy a larger/smaller string onto the heap? Avoiding overflows?

peaksoonpeak - June 30, 2015, 4:15 pm

(Jun 30, 2015 04:08 PM)JavaProphet Wrote: [\[CITE\]](#) Not a bad idea. I'm still new to C. Thanks!

EDIT: What exactly do you mean by calculating the null terminated string? The lengths?
How exactly do I copy a larger/smaller string onto the heap? Avoiding overflows?

I've posted code above for finding a said string in a said buffer.

For example, let's take this buffer

"IAMTHEBUFFER"

I'd like to replace THE with A

So the function above will find 'THE' at Buffer+0x03.

You will then allocate memory with malloc with the size defined as,

dwNewBufferSize = dwCurrentBufferSize + (Length of A - Length of THE)

dwNewBufferSize = dwCurrentBufferSize - 2

You then allocate memory, copy the first 3 bytes of the initial buffer (iam) then copy the new string (a) then copy the rest.

memcpy(lpNewBuffer, Buffer, (DWORD)((Buffer) - (Buffer+0x03)); Remember, buffer+0x03 was the location of the string in the buffer. We want to copy the first portion of the buffer into the newly allocated block.

Then you copy the new string

memcpy(lpNewBuffer+((DWORD)((Buffer) - (Buffer+0x03)), NewStringWhatever, lengthofnewstring);

In this line of code we will be copying 'A' to lpnewbuffer + the previous portion

Then you finally copy the rest.

The first portions of the initial buffer were IAMTHE so what we are going to do is define the location of the initial buffer as,

LPVOID lpTempBuffer = (LPVOID)(Buffer+strlen("IAMTHE"));

and the length we copy over

DWORD dwTempBuffer = ((DWORD)Buffer + strlen(Buffer)) - (DWORD)lpTempBuffer

```
memcpy(lpNewBuffer+((DWORD)(Buffer)-(Buffer+0x03))+Lengthofnewstring, lpTempBuffer,
dwTempBuffer);
```

JavaProphet - June 30, 2015, 4:18 pm

@peaksoonpeak I do this. I posted an old function, perhaps you looked at that? I updated it quickly after I posted. I loop over the function, and get the match count, then multiply it by the equivalent of THE-A length, then allocate it all at once.

Anyways, here's what I have now:

```
Code char* replace(char* input, char* rf, char* rt) {
    size_t isl = strlen(input);
    size_t rfsl = strlen(rf);
    size_t rtsl = strlen(rt);
    size_t ml = 0;
    size_t mc = 0;
    int i = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                mc++;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
    }
    i = 0;
    ml = 0;
    char* output = malloc(isl - ((rfsl - rtsl) * mc));
    output[0] = '\0';
    int mi = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                size_t os = mi - ml + 1;
                strcpy(output + os, rt);
                mi += ml - rtsl - 1;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
        output[mi++] = input[i];
    }
    output[mi] = '\0';
    return output;
}
```

```
}
```

peaksoonpeak - June 30, 2015, 4:25 pm

(Jun 30, 2015 04:18 PM)JavaProphet Wrote: [\[CITE\]](#) @peaksoonpeak I do this. I posted an old function, perhaps you looked at that? I updated it quickly after I posted. I loop over the function, and get the match count, then multiply it by the equivalent of THE-A length, then allocate it all at once.

Anyways, here's what I have now:

```
Code char* replace(char* input, char* rf, char* rt) {
    size_t isl = strlen(input);
    size_t rfsl = strlen(rf);
    size_t rtsl = strlen(rt);
    size_t ml = 0;
    size_t mc = 0;
    int i = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                mc++;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
    }
    i = 0;
    ml = 0;
    char* output = malloc(isl - ((rfsl - rtsl) * mc));
    output[0] = '\0';
    int mi = 0;
    for (; i < isl; i++) {
        bool im = input[i] == rf[ml];
        if (im) {
            ml++;
            if (ml == rfsl) {
                size_t os = mi - ml + 1;
                strcpy(output + os, rt);
                mi += ml - rtsl - 1;
                ml = 0;
                continue;
            }
        } else if (ml > 0) {
            ml = 0;
        }
        output[mi++] = input[i];
    }
    output[mi] = '\0';
    return output;
}
```

As to your PM i stay away from skype, come join IRC irc.#.net everyone there is helpful :)

my alias is error

edit:
try

chat.#.net - kiwi IRC web client.

JavaProfit - June 30, 2015, 4:46 pm

(Jun 30, 2015 04:43 PM)JavaProphet Wrote: [\[CITE\]](#) @Systemic They're all acronyms, it's just how I prefer to write code. I never intended on publishing it in any manner.

If I really need an efficient String operation in Java, I just start working with char[]. Unfortunately the array system is not as fleshed out as C, due to a lack of pointers.

No problem there, it was just a suggestion. char[] that isn't copied on a method like ".toCharArray()" is pretty much your best bet.

Java arrays are fine, they were designed to be safe.

JavaProphet - June 30, 2015, 4:49 pm

(Jun 30, 2015 04:46 PM)Systemic Wrote: [\[CITE\]](#) No problem there, it was just a suggestion. char[] that isn't copied on a method like ".toCharArray()" is pretty much your best bet.

Java arrays are fine, they were designed to be safe.

Safe yes, but I have only begun to use C, and when I write Java, I miss pointers already.

I'm considering learning D too. It's almost C-level Java.

And alright peak.

Sparta. - June 30, 2015, 11:52 pm

(Jun 30, 2015 01:09 PM)BinaryBit Wrote: [\[CITE\]](#) No need to be rude. Also, all of the libraries in Java lead up to C or C++. Without natives, you couldn't do anything in Java, really. You're not very smart for not thinking about that. I can't do it, because I cannot understand any of the tutorials online. And, if it's very simple, please help me out, as that is the point of this thread. Thank-you.

And without ASM you can't use C or C++, retard! Hurr!

Fucking moron. Shut the fuck up.

Every single response you give to people adds a new cancer cell to my brain just from trying to process your stupidity.

Why in all fucks do you bash on a language you don't know SHIT about?

If you can't even understand the most simplistic tutorials online (although they are shitty, I may say myself), then you are the dumb fuck here.

tldr stop posting here.

P.S. You can achieve anything in Java without using the JNI. You're an absolute fucking moron if you believe otherwise. It's just the way it is accomplished, that's all.

BinaryBit - July 1, 2015, 12:37 am

(Jun 30, 2015 11:52 PM)Sparta. Wrote: [\[CITE\]](#) And without ASM you can't use C or C++, retard! Hurr!

Fucking moron. Shut the fuck up.

Every single response you give to people adds a new cancer cell to my brain just from trying to process your stupidity.

Why in all fucks do you bash on a language you don't know SHIT about?

If you can't even understand the most simplistic tutorials online (although they are shitty, I may say myself), then you are the dumb fuck here.

tldr stop posting here.

P.S. You can achieve anything in Java without using the JNI. You're an absolute fucking moron if you believe otherwise. It's just the way it is accomplished, that's all.

Dude, are you high on drugs, or just have high-functioning autism? You seem like one of those highly-raged little kids in your everyday voice communication in Team Fortress 2. I like how you lowered my rep 2 more times because you are so butthurt. I don't even know what you are saying anymore. I hope to god that you are trolling because, if not, you are obviously disabled (no offense to you, but I said it for truth's sake).

MrOrang - July 1, 2015, 2:41 am

Please no flaming and if this turns into a C thread I'll havw to move it.

BinaryBit - July 1, 2015, 8:33 pm

(Jul 1, 2015 02:41 AM)Looka Wrote: [\[CITE\]](#) Please no flaming and if this turns into a C thread I'll havw to move it.

This is not a C thread, it is a thread on the Java Native Interface which involves using C, but is still Java. Also, I hate flaming, and that enraged disabled user had to do it. Also, could you consider getting a Java Native Interface (JNI) subsection created, since the JNI is both revolved around C and Java? Thank-you.

Sparta. - July 1, 2015, 9:40 pm

(Jul 1, 2015 08:33 PM)BinaryBit Wrote: [\[CITE\]](#) This is not a C thread, it is a thread on the Java Native Interface which involves using C, but is still Java. Also, I hate flaming, and that enraged disabled user had to do it. Also, could you consider getting a Java Native Interface (JNI) subsection created, since the JNI is both revolved around C and Java? Thank-you.

He isn't even a moderator/administrator you fucking idiot.

JNI is apart of Java, so it stays in the Java section. Anything C related goes to the C section.

You are the most fallacious and ignorant person here by far. You claim in the OP that you "know a lot about Java"; then why the fuck are you asking for help? ?

JavaProfit - July 2, 2015, 3:25 pm

(Jul 1, 2015 09:40 PM)Sparta. Wrote: [\[CITE\]](#) He isn't even a moderator/administrator you fucking idiot.

JNI is apart of Java, so it stays in the Java section. Anything C related goes to the C section.

You are the most fallacious and ignorant person here by far. You claim in the OP that you "know a lot about Java"; then why the fuck are you asking for help? ?

He's gotta make sure the entire HF 1337 SQUAD knows that he's too smart for them so he has to ask things that he can answer himself.

