

Re: Efficient scather-gather-copy

Source: <http://coding.derkeiler.com/Archive/Assembler/comp.lang.asm.x86/2006-04/msg00160.html>

- *From:* Hynek Schlawack <spamtrap@xxxxxxxxxxx>
 - *Date:* Fri, 14 Apr 2006 11:39:57 +0200
-

spamtrap@xxxxxxxxxxx writes:

It's UNIX and yet filtering is simply too slow. I'd rather use memcpy() than external programs.

I can't see that memcpy() would buy you much here, but perhaps I'm misunderstanding the problem.

Filtering through other programs would lead ultimately also to copying chunks of data, so I guess that wouldn't help me much.

In order to alloc() the buffer you need, you'll have to make two passes through the buffer: one to count the newlines and one to do the copying. The 2nd pass might be faster if it is cached, but a single pass will almost certainly be faster overall.

Yeah, in this way I like the idea to simply allos twice the size of the buffer.

Well, I'm getting it as a buffer and anything that remotely could involve access to file systems is out of question.

Do you know the size of the buffer? Or the size of the data within the buffer?

Yes²

If so the worst case scenario would be a buffer full of newlines and the output would have length (size*2) so if resources permit, just alloc() a buffer of that size. The pseudo-C code could be something like

Re: Efficient scather-gather-copy

```
char *xlate (char *src, size_t len)
{
char *dst = malloc (len * 2);
char *d, *end = src + len;
for (d = dst; src < end; src++)
{
if (*src == '\n')
*d++ = '\r';
*d++ = *src;
}
return realloc (dst, (d - dst));
}
```

[Obviously with some error checking!] See what asm the compiler outputs and try to optimise that — probably the loop, though it's already pretty simple and the C compiler should make a pretty good job of it anyway.

I guess too. As said, my original idea was to build a copy plan and boost it with some CPU magic. I guess this * 2 is my way to go.

Imagine dozens of mails per `_second_` that want to be processed. Probably, `memcpy()` might be fast enough on a decent machine, however it would definitely be a bottleneck. So I want to make it really as fast as possible, as there isn't unfortunately much I can optimize algorithm-sided.

If you have access to the input stream, perhaps look at injecting the `\r`'s there on-the-fly?

I haven't. :(The stupid part is that the `\r` get removed and I have to put them back. :(

Thanks to everyone very much for your help!
-hs

.