

# Re: A solution for the allocation failures problem

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2008-01/msg04364.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2008-01/msg04364.html)

---

- *From:* [vippstar@xxxxxxxxxx](mailto:vippstar@xxxxxxxxxx)
  - *Date:* Tue, 29 Jan 2008 04:26:56 -0800 (PST)
- 

On Jan 29, 1:47 pm, jacob navia <[ja...@xxxxxxxxxx](mailto:ja...@xxxxxxxxxx)> wrote:

1:  
It is not possible to check EVERY malloc result within complex software.

It is, with proper design it becomes easier too.

3:  
A solution like the one proposed by Mr McLean (aborting) is not possible for software quality reasons. The program must decide if it is possible to just abort() or not.

That was not a solution.

Solution:

1) At program start, allocate a big buffer that is not used elsewhere in the program. This big buffer will be freed when a memory exhaustion situation arises, to give enough memory to the error reporting routines to close files, or otherwise do housekeeping chores.

That does not guarantee anything, it depends on the OS, and the implementation of free/malloc.

Also it makes your program slower, and what if the big buffer allocation fails?

Would you exit because your program failed to allocate resources it would not use?

What if no allocation fails? The buffer is only a waste of resources.

2) xmalloc()

```
static int (*mallocfailedHandler)(int);
void *xmalloc(size_t nbytes)
{
  restart:
```

## Re: A solution for the allocation failures problem

```
void *r = malloc(nbytes);
if (r)
return r;
// Memory exhaustion situation.
// Release some memory to the malloc/free system.
if (BigUnusedBuffer)
free(BigUnusedBuffer);
```

free it anyway, free(NULL); does nothing.

```
BigUnusedBuffer = NULL;
if (mallocfailedHandler == NULL) {
// The handler has not been set. This means
// this application does not care about this
// situation. We exit.
fprintf(stderr,
"Allocation failure of %u bytes\n",
nbytes);
```

Undefined behavior, you pass size\_t where a variadic function expects unsigned int.

```
fprintf(stderr,"Program exit\n");
exit(EXIT_FAILURE);
}
```

What about previous allocations that have been done with xmalloc?  
memory leak.

```
// The malloc handler has been set. Call it.
if (mallocfailedHandler(nbytes)) {
goto restart;
}
// The handler failed to solve the problem.
// Exit without any messages.
exit(EXIT_FAILURE);
```

Memory leak because you don't give the user a chance to free previous allocations

The recovery handler is supposed to free memory, and reallocate the BigUnusedBuffer, that has been set to NULL;

So if the allocation for BigUnusedBuffer succeeds but the allocation after the callback fails, we will enter a loop of free-ing/allocating the big buffer, great.

Re: A solution for the allocation failures problem

mr Jacob, I suggest you read a book or two on program structure & designing.