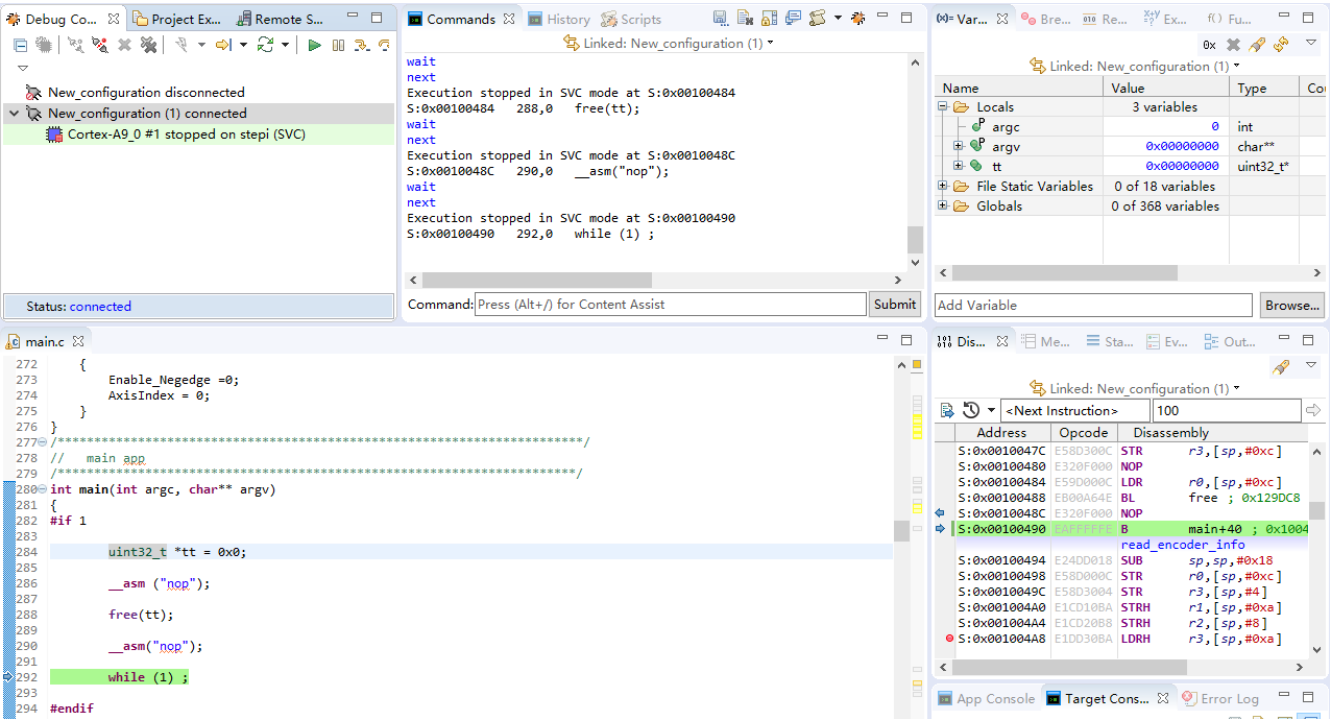


newlib库移植前后的堆分配和释放函数的对比验证测试

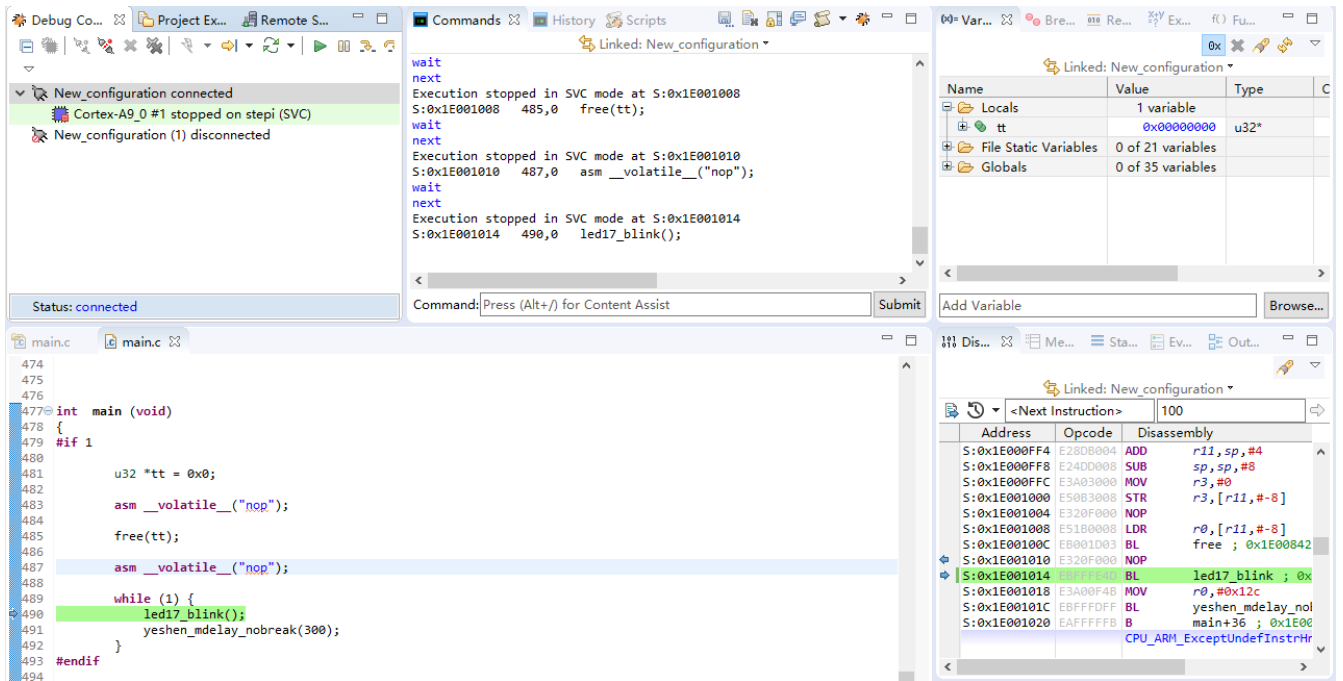
一、释放空指针

原版：



上图是未移植newlib版的释放空指针的情况，可以看到释放空指针不会跑飞，然后正在往下运行。

移植newlib版：

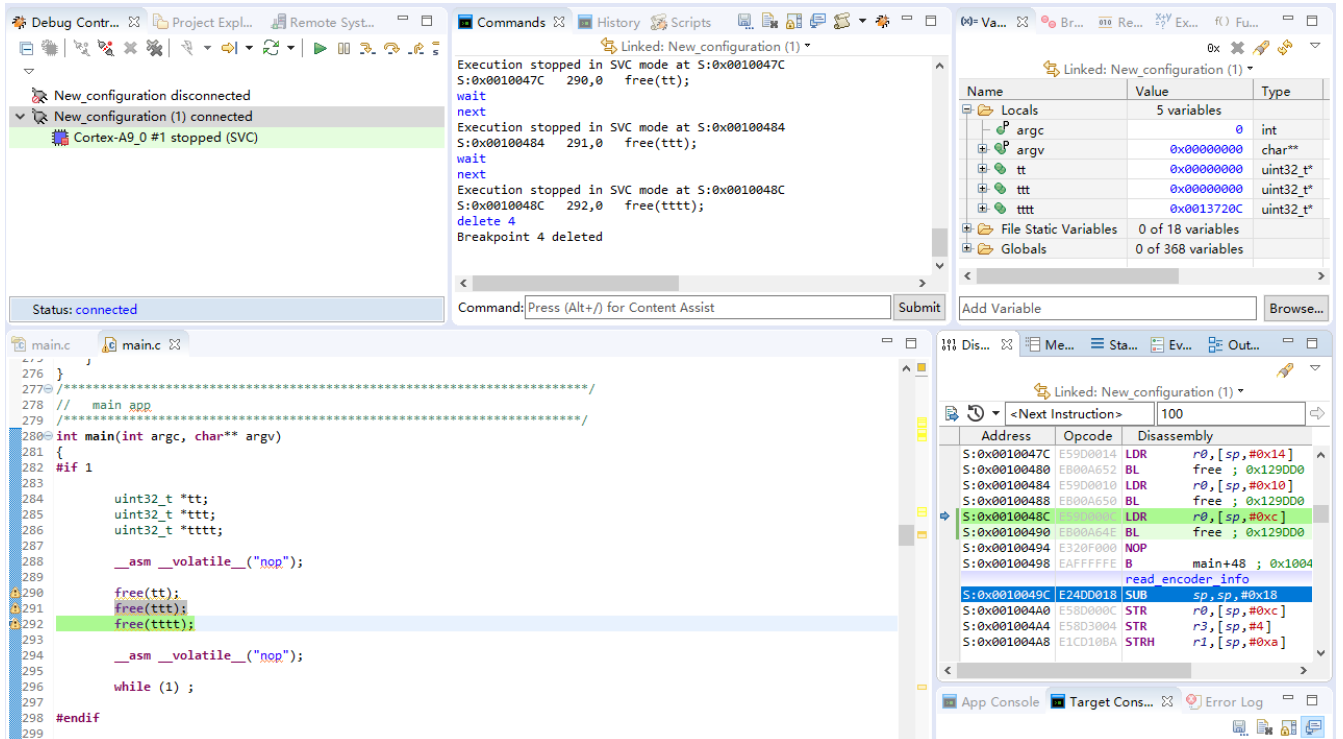


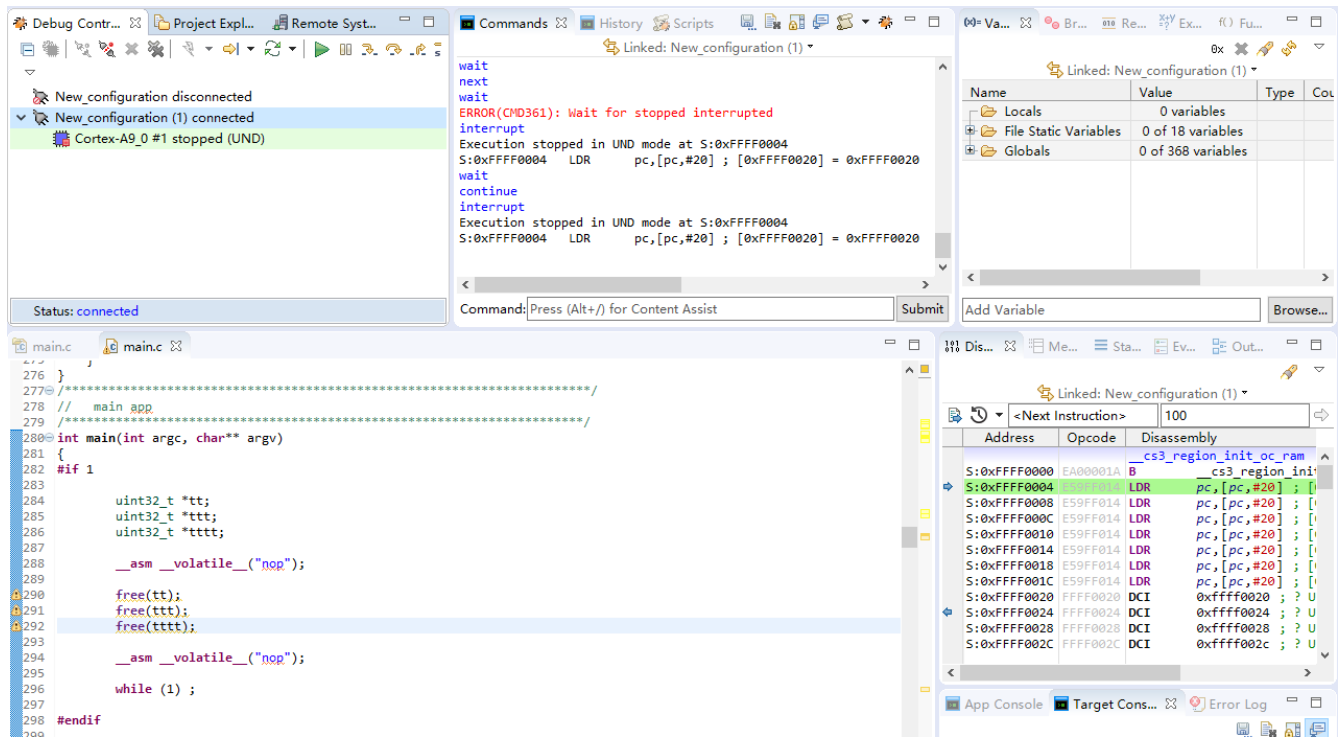
上图是移植newlib版的释放空指针的情况，可以看到释放空指针也不会跑飞，然后正在往下运行。

结论：从实验结果和newlib的源代码中分析到释放空指针不会有任何效果，直接返回，也不会发生错误，两个版本释放空指针行为一致。

二、释放未初始化指针

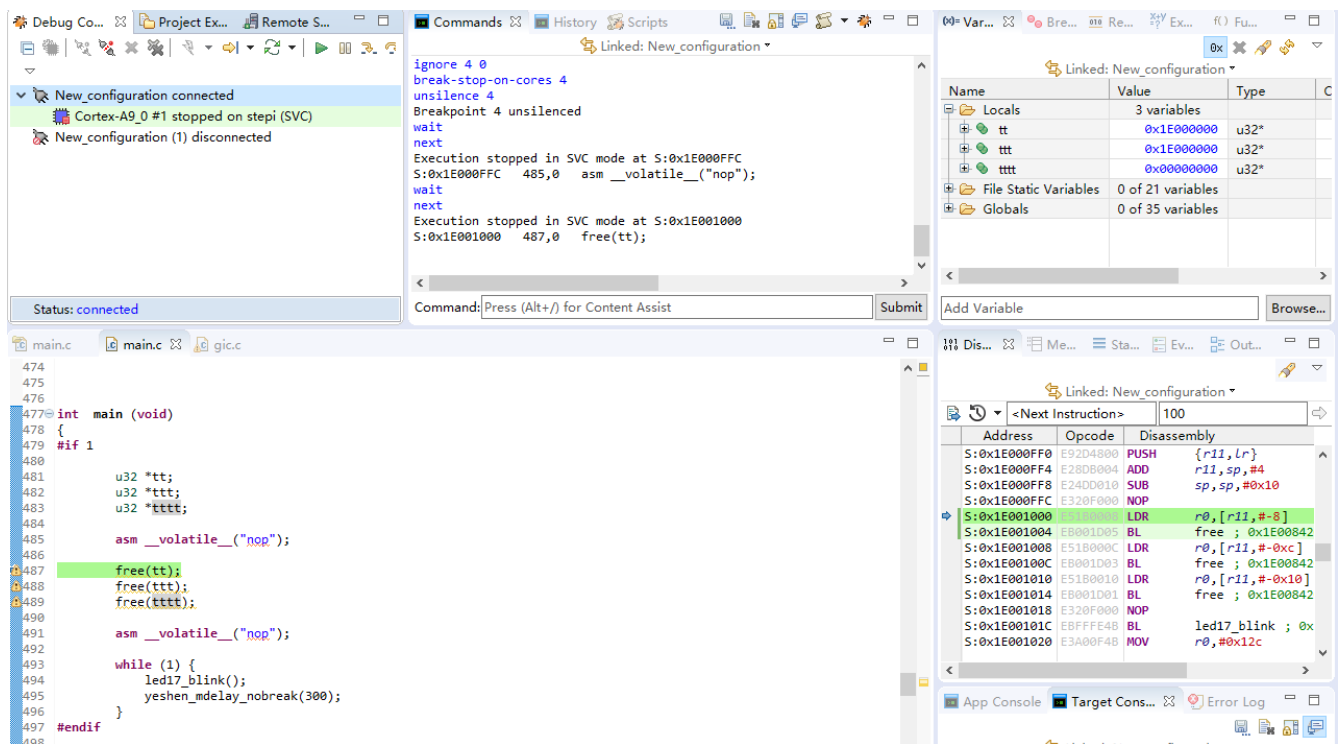
原版：

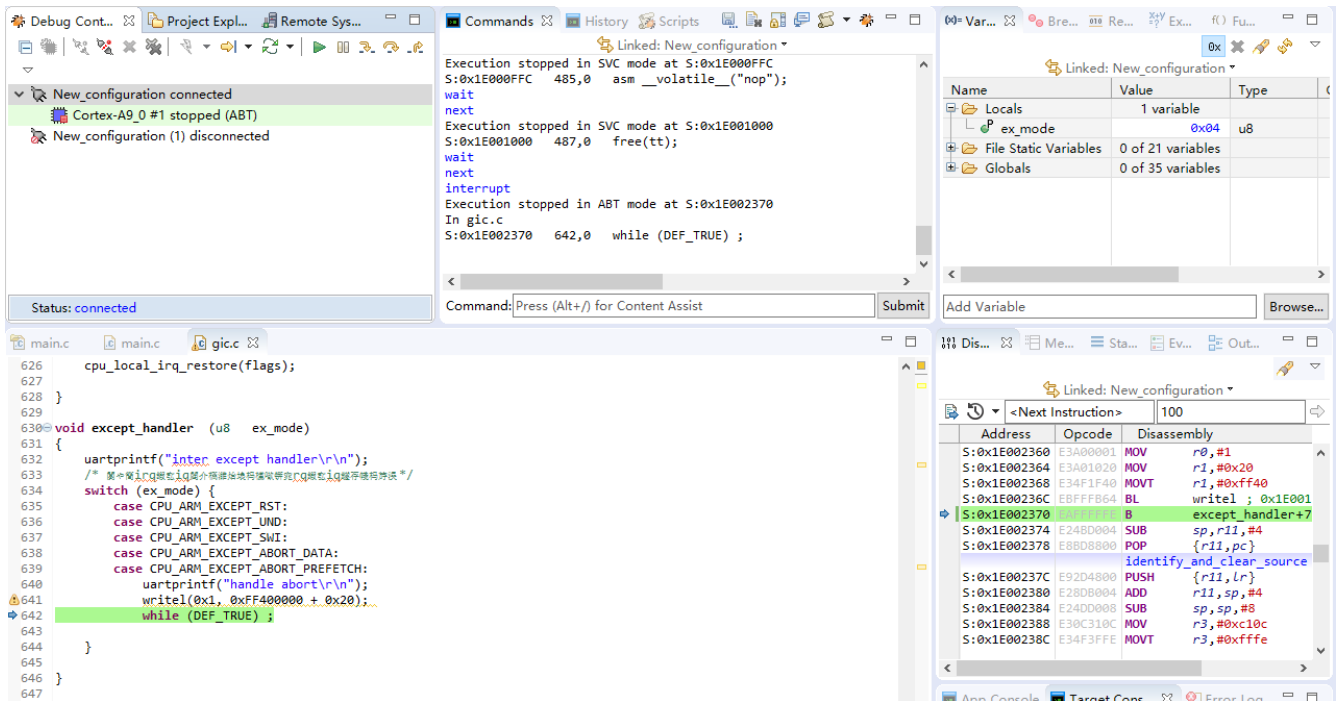




上图释放未初始化的三个指针tt,ttt,tttt。释放前两个没有跑飞，因为tt和ttt刚好是0,就是空指针，而释放tttt的时候跑飞了，因为这不是0。

移植newlib版:



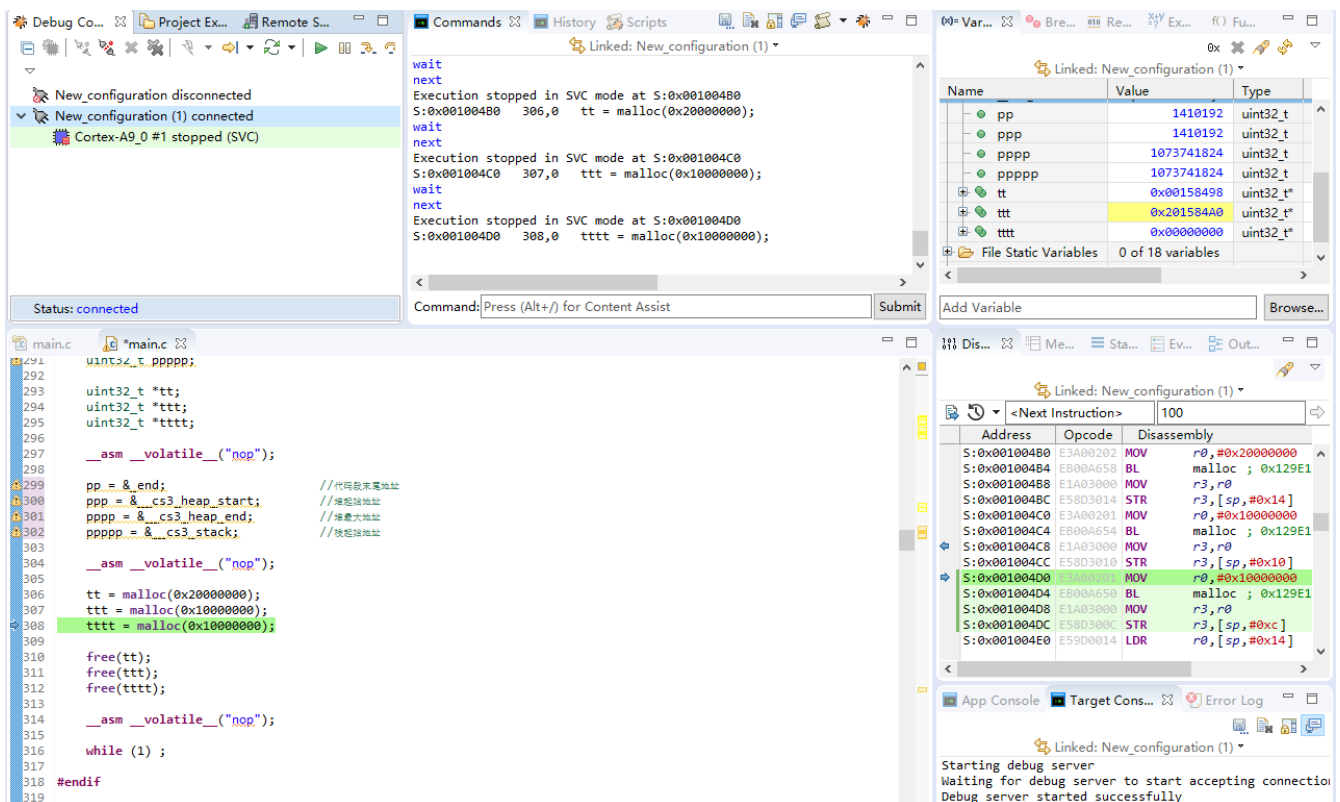


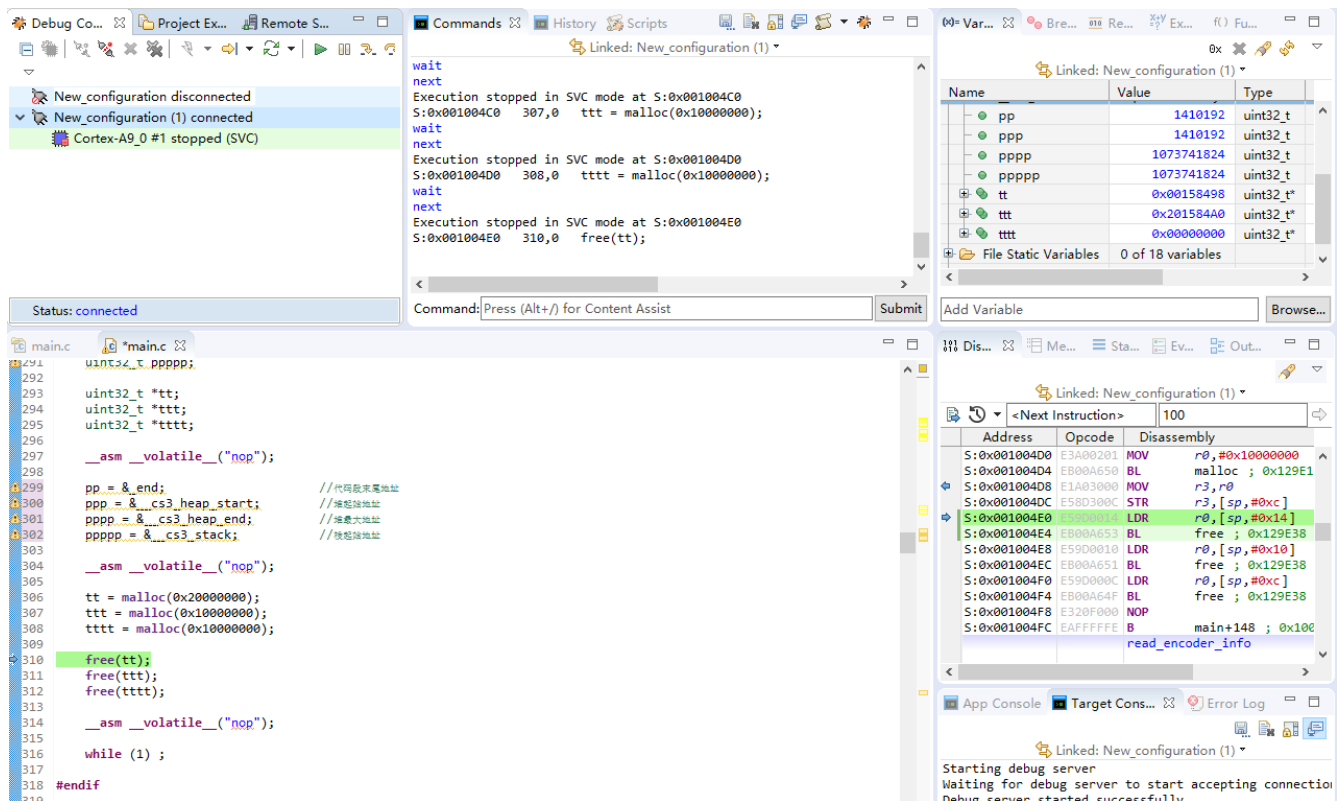
上图也是释放未初始化的三个指针tt,ttt,tttt。第一次释放就跑飞，因为第一个指针就不是0。

结论：不能释放未初始化指针，如果未初始化指针的值不巧好为0而直接释放就会跑飞，因为这不是一个合法的申请得到的指针，两个版本释放未初始化指针行为一致。

三、申请超限

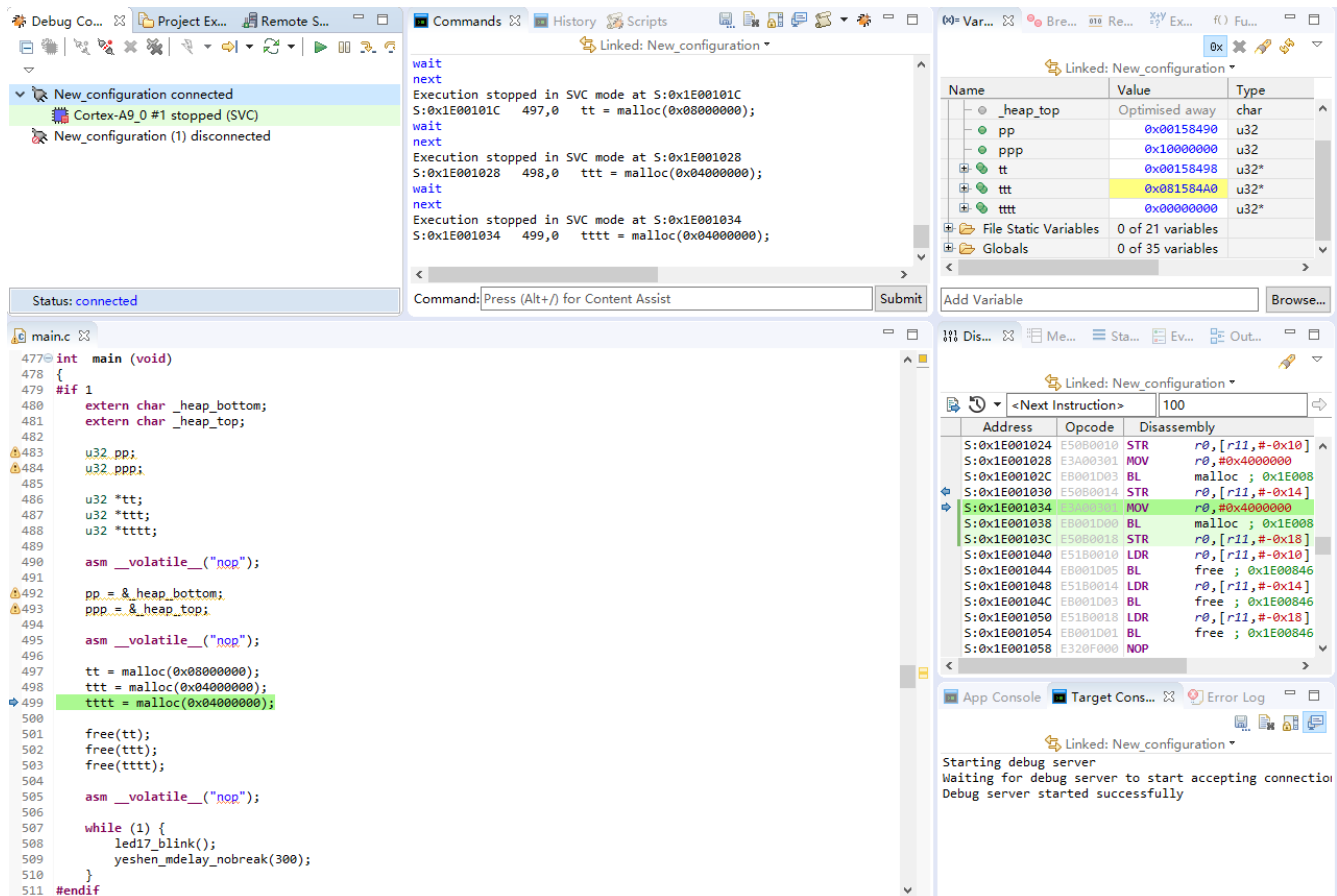
原版：

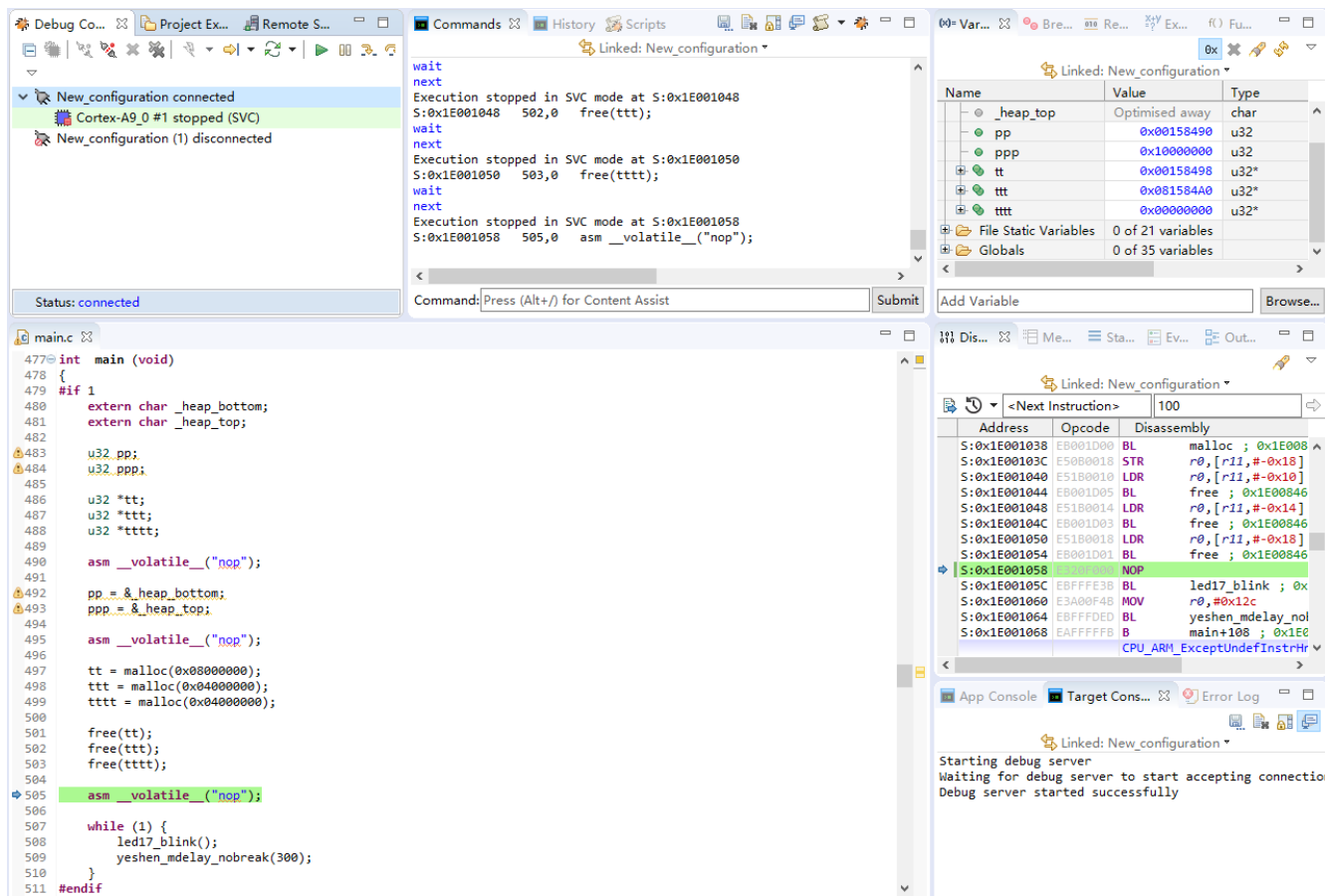




上图连续申请三块内存，申请前两块时没有超过堆的最大界限，到申请第三块时超限，可从图中看到，tttt为0，即申请超限后返回空指针，然后往后正常运行。

移植newlib版:





上图的现象和未移植newlib版一致。

结论：申请超限后返回空指针，不跑飞。两个版本行为一致。

四、成功申请内存后，越界写然后释放

原版：

The image displays a debugger interface with two panels showing the execution of a program on a Cortex-A9 processor.

Top Panel (SVC mode):

- Commands:** Shows the execution flow: `wait`, `next`, `Execution stopped in SVC mode at S:0x0010049C`, `S:0x0010049C 293,0 tttt = tt - 1;`, `wait`, `next`, `Execution stopped in SVC mode at S:0x001004A8`, `S:0x001004A8 294,0 *tttt = 0xbade7a;`, `wait`, `next`, `Execution stopped in SVC mode at S:0x001004B8`, `S:0x001004B8 296,0 free(ttt);`.
- Disassembly:** Shows the current instruction: `S:0x001004B8 E800A651 LDR r0, [sp, #0x14]`.
- Variables:** Shows the state of variables: `argc` (5 variables), `argv` (0x00000000), `tt` (0x00158498), `tttt` (0x001584A8), `File Static Variables` (0 of 18 variables), `Globals` (0 of 368 variables).

Bottom Panel (UND mode):

- Commands:** Shows the execution flow: `next`, `Execution stopped in SVC mode at S:0x001004A8`, `S:0x001004A8 294,0 *tttt = 0xbade7a;`, `wait`, `next`, `Execution stopped in SVC mode at S:0x001004B8`, `S:0x001004B8 296,0 free(ttt);`, `interrupt`, `Execution stopped in UND mode at S:0xFFFF0004`, `S:0xFFFF0004 LDR pc, [pc, #20] ; [0xFFFF0020] = 0xFFFF0020`.
- Disassembly:** Shows the current instruction: `S:0xFFFF0004 EA00001A B cs3_region_init_oc_ram`.
- Variables:** Shows the state of variables: `argc` (0 variables), `argv` (0 of 18 variables), `File Static Variables` (0 of 368 variables), `Globals` (0 of 368 variables).

上图展示的是成功申请到一块内存后然后越界向前写一个字，然后跑飞。

The first screenshot shows a GDB session where the program successfully allocates memory and then frees it. The status bar indicates 'Status: connected'. The command window shows the execution flow: 'wait', 'next', 'Execution stopped in SVC mode at S:0x00100488', 'wait', 'next', 'Execution stopped in SVC mode at S:0x001004C4', 'wait', 'next', 'Execution stopped in SVC mode at S:0x001004D4'. The disassembly window shows the instructions for the 'main' function, including 'LDR r0, [sp, #0x14]' and 'BL free ; 0x129E20'.

```

278 // main app
279 /*****
280 int main(int argc, char** argv)
281 {
282     #if 1
283
284         uint32_t *tt;
285         uint32_t *ttt;
286         uint32_t *tttt;
287
288         __asm __volatile__("nop");
289
290         tt = malloc(8);
291         ttt = malloc(16);
292
293         tttt = tt + 2;
294         *tttt = 0xbadeda7a;
295         tttt = tttt + 1;
296         *tttt = 0xbadeda7a;
297
298         free(tt);
299         free(ttt);
300
301         __asm __volatile__("nop");
302
303         while (1);
304
305     #endif
306 }

```

The second screenshot shows the program crashing due to a buffer overflow. The status bar indicates 'Status: connected'. The command window shows the execution flow: 'next', 'Execution stopped in SVC mode at S:0x001004C4', 'wait', 'next', 'Execution stopped in SVC mode at S:0x001004D4', 'wait', 'next', 'interrupt', 'Execution stopped in UND mode at S:0xFFFF0004'. The disassembly window shows the instructions for the 'main' function, including 'LDR pc, [pc, #20]' and 'DCI 0xffff0020 ; ? U'. The error log shows 'Starting debug server' and 'Debug server started successfully'.

```

278 // main app
279 /*****
280 int main(int argc, char** argv)
281 {
282     #if 1
283
284         uint32_t *tt;
285         uint32_t *ttt;
286         uint32_t *tttt;
287
288         __asm __volatile__("nop");
289
290         tt = malloc(8);
291         ttt = malloc(16);
292
293         tttt = tt + 2;
294         *tttt = 0xbadeda7a;
295         tttt = tttt + 1;
296         *tttt = 0xbadeda7a;
297
298         free(tt);
299         free(ttt);
300
301         __asm __volatile__("nop");
302
303         while (1);
304
305     #endif
306 }

```

上图展示的是成功申请一块内存然后越界往后写内存，最后跑飞。

移植newlib版：

The screenshot displays a debugger interface with the following components:

- Debug Control:** Shows the status of the target system as 'connected'.
- main.c:** The source code for the program, showing a loop that writes to memory.
- Commands:** The command window showing the execution of the program and the state of the target system.
- Disassembly:** The disassembly window showing the assembly instructions being executed.

The 'main.c' code is as follows:

```
477 int main (void)
478 {
479     #if 1
480
481     u32 *tt;
482     u32 *ttt;
483     u32 *tttt;
484
485     asm __volatile__ ("nop");
486
487     tt = malloc(8);
488     ttt = malloc(16);
489
490     tttt = tt - 1;
491     *tttt = 0xbadeda7a;
492
493     free(tt);
494     free(ttt);
495
496     asm __volatile__ ("nop");
497
498     while (1) {
499         led17_blink();
500         yeshen_mdelay_nobreak(300);
501     }
502 }
503 #endif
```

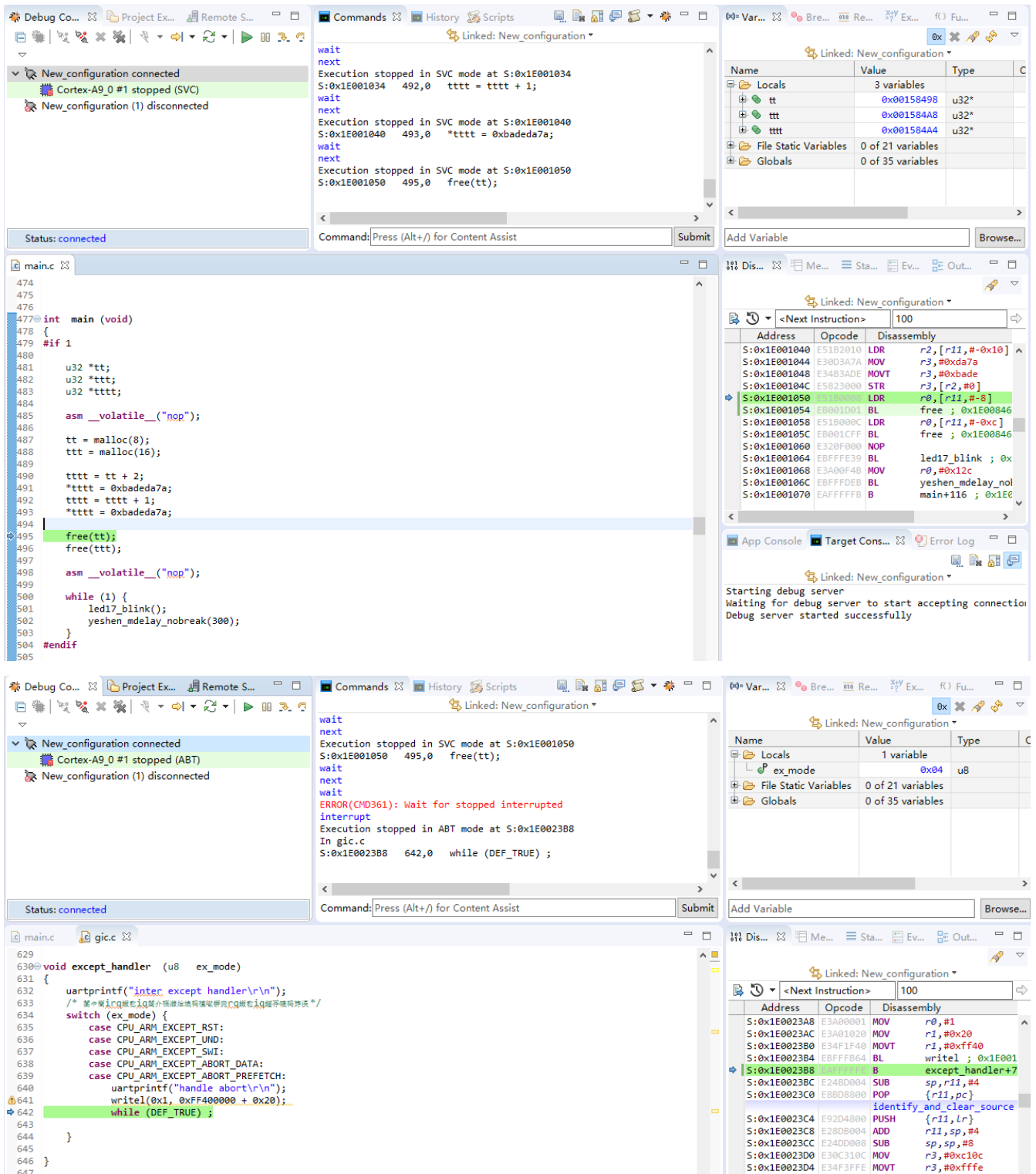
The 'Commands' window shows the execution of the program and the state of the target system:

```
wait
next
Execution stopped in SVC mode at S:0x1E001018
S:0x1E001018 490,0 tttt = tt - 1;
wait
next
Execution stopped in SVC mode at S:0x1E001024
S:0x1E001024 491,0 *tttt = 0xbadeda7a;
wait
next
Execution stopped in SVC mode at S:0x1E001034
S:0x1E001034 493,0 free(tt);
```

The 'Disassembly' window shows the assembly instructions being executed:

Address	Opcode	Disassembly
S:0x1E001024	E5182010	LDR r2, [r1, #-0x10]
S:0x1E001028	E3003A7A	MOV r3, #0xda7a
S:0x1E00102C	E3483ADE	MOV r3, #0xbade
S:0x1E001030	E5823000	STR r3, [r2, #0]
S:0x1E001034	E8001D00	LDR r0, [r1, #-8]
S:0x1E001038	E8001D00	BL free ; 0x1E00844
S:0x1E00103C	E518000C	LDR r0, [r1, #-0xc]
S:0x1E001040	E8001CFE	BL free ; 0x1E00844
S:0x1E001044	E32F0000	NOP
S:0x1E001048	EBFFFE40	BL led17_blink ; 0x
S:0x1E00104C	E3A00F48	MOV r0, #0x12c
S:0x1E001050	EBFFFD2	BL yeshen_mdelay_nob
S:0x1E001054	EAFFFFB	B main+88 ; 0x1E00

上图展示的是成功申请一块内存然后越界向前写内存，最后跑飞。



上图展示的是成功申请一块内存然后越界往后写内存，最后跑飞。

结论：成功申请内存后不能越界写，否则会跑飞。两个版本行为一致。