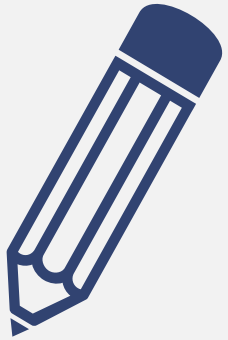


ICS杂谈——VIM与输出调试

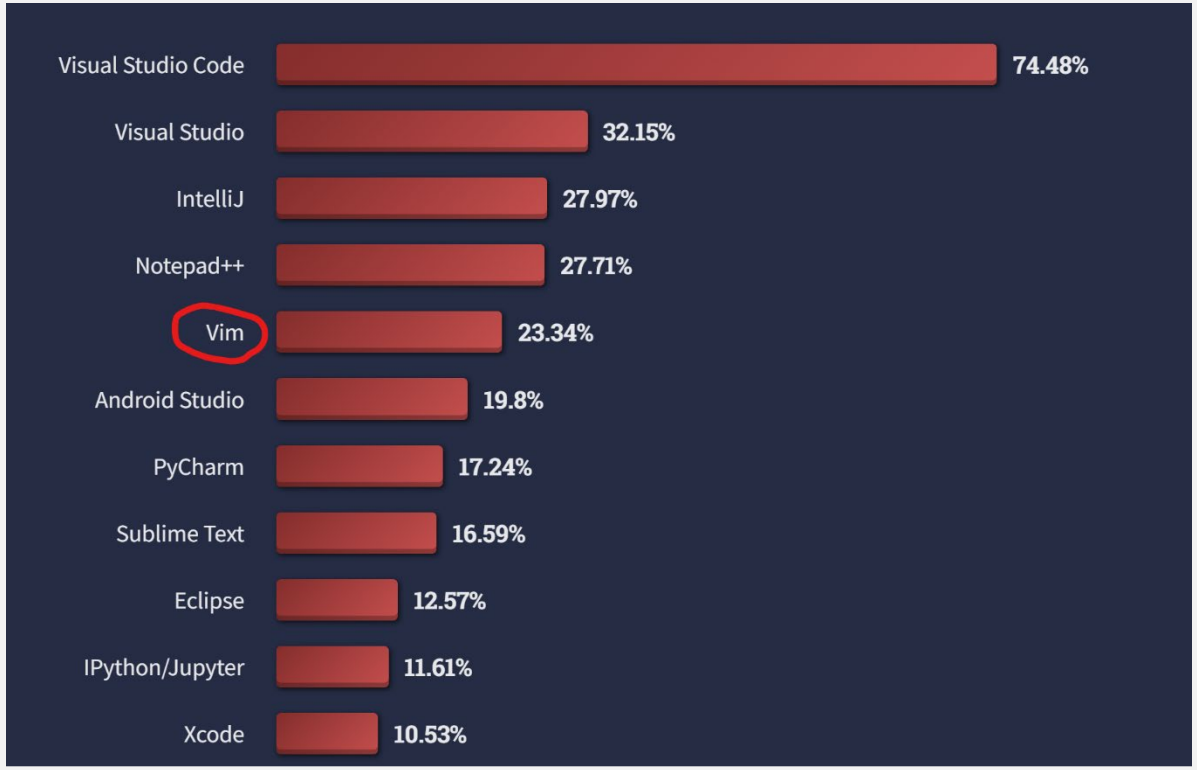
匡亚明学院
潘昕田
xintianpan@smail.nju.edu.cn

VIM

何为vim?

~~**大家应该已经在PA0手册里面认识了**~~

Stack Overflow的调查（Stack Overflow Developer Survey 2022）表明vim是当前最流行的基于命令行的文本编辑器



为什么要用vim?

几个推荐vim的理由

1.可以在终端直接运行，方便调试代码

2.摆脱鼠标，效率更高

3.命令之间相互组合可以实现更强大的功能

~~4.vscode一打开文件就报错+不想配置~~

简单下载并按照PA0手册去配置vim似乎并不足够

不少插件可以帮助我们更便捷地使用vim

我使用过的插件：

nerdtree（强烈推荐）

themes（可以更改vim的颜色主题）

ctags（本质上并不算插件，但也是一个很好用的工具）

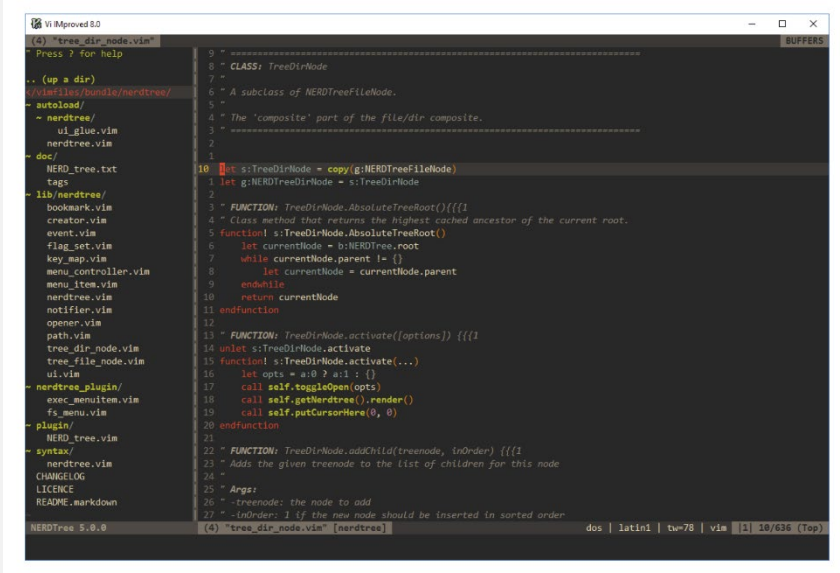
nerdtree —— 文件浏览器

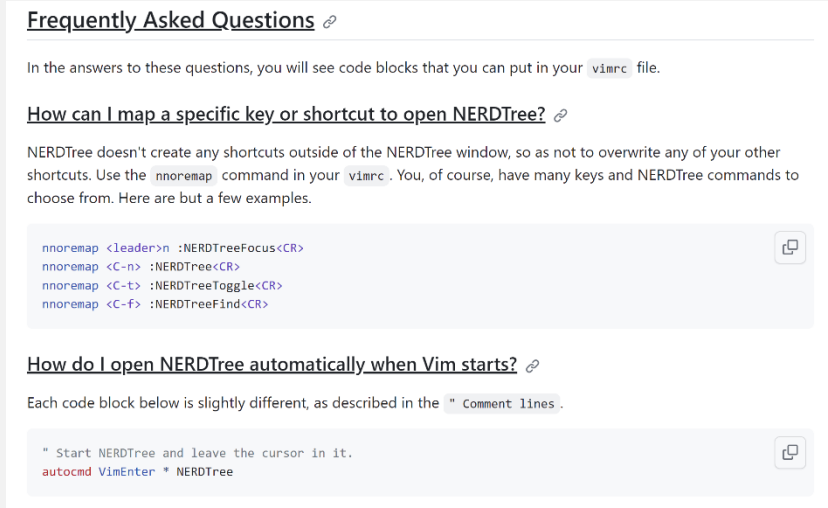
可以在侧面生成一个文件目录便于浏览

文件目录和文本内容是分离的，你可以在vim编辑器内通过指令打开或者关闭它

nerdtree默认是不会直接打开的，但你可以在.vimrc里面配置它

以上指令和配置在nerdtree的Github上都有介绍





Frequently Asked Questions

In the answers to these questions, you will see code blocks that you can put in your `vimrc` file.

How can I map a specific key or shortcut to open NERDTree?

NERDTree doesn't create any shortcuts outside of the NERDTree window, so as not to overwrite any of your other shortcuts. Use the `nnoremap` command in your `vimrc`. You, of course, have many keys and NERDTree commands to choose from. Here are but a few examples.

```
nnoremap <leader>n :NERDTreeFocus<CR>
nnoremap <C-n> :NERDTree<CR>
nnoremap <C-t> :NERDTreeToggle<CR>
nnoremap <C-f> :NERDTreeFind<CR>
```

How do I open NERDTree automatically when Vim starts?

Each code block below is slightly different, as described in the `" Comment lines`.

```
" Start NERDTree and leave the cursor in it.
autocmd VimEnter * NERDTree
```

Theme —— 颜色主题

顾名思义，就是给你的vim配置不同的颜色主题

直接在Google上搜索vim themes即可

ctags —— 代码跳转

ctags实际上是一个生成代码跳转用的.tags文件的工具，你需要通过apt下载它（如果你直接sudo apt install ctags会出现exuberant-ctags和universal-ctags两个选项，选择universal-ctags，exuberant-ctags现在已经停止维护了）

下载好ctags后你需要在.vimrc配置查询tags文件的路径，具体配置方法和跳转操作直接STFW即可

如果坚持使用vscode但不希望到处报错

你需要配置complie_commands.json来完善vscode的函数跳转、代码补全等功能

当然，你并不需要手动配置这一文件，bear这一工具可以帮助你自动通过Makefile生成complie_commands.json（通过bear -- make）来实现

注意到如果你的项目已经make并且build文件存在，你可能需要先make clean

2 输出调试法

printf调试法能不能应用于PA这样的大项目？

可以，但前提必须建立在对于PA各个模块的交互方式较为熟悉（例如am如何和nemu之间交互，PA3中的nanos-lite又是怎么和am交互，PA4增加虚地址转换后交互方式有何变化等都需要了解），否则连在哪输出调试都不知道

同时，不能仅通过printf来调试，必须综合其它工具或内容（例如汇编文档）来

有了difftest和sdb还需要printf调试吗

视个人情况，以下为个人体会

光有difftest并不够，difftest能帮助解决指令实现错误的问题，但是对于非指令实现错误导致的bug则无能为力（例如函数中使用的缓冲区溢出，指令上并无错误，但因为溢出覆盖了其它地址的内容可能导致错误）

sdb由于对每一条指令都要打断进行表达式求值，速度上不及printf，同时如果仅凭PA1中实现的断点功能，很难监测特定值的写入或者读出操作

printf输出调试法例

PA3我曾经在miniSDL库实现绘图api时因为缓冲区申请大小错误导致缓冲区溢出覆盖了其它位置的存储的指针导致后续访存出错，使得运行flappy bird直接闪退。

后来通过printf输出解决这一问题。

printf输出调试法例

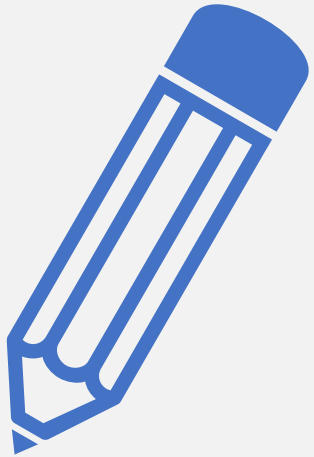
调试流程：

1.首先确定了报错的指令pc值，在汇编文档找到函数的位置，通过printf确定具体的问题

2.通过printf输出出错的指针的值，通过前后比对发现在初始化时值是正确，但是在后续调用被修改为其它值

3.在nemu监测该地址写入操作并比对写入值，printf输出得到写入该值时的pc值，确定其在的函数，最终确认为miniSDL库相关api实现错误，最终解决该bug

只要姿势正确，“printf大法”还是有用武之地的



谢谢！