目录

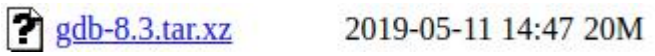
一、安装

命令行安装

```
sudo apt-get install gdb
```

但是我命令行默认安装的是 8.1.1，在后面执行一些命令时有问题，因此自己手动源码编译了 gdb 8.3 版本

先从官网下载对应压缩包 https://ftp.gnu.org/gnu/gdb/



然后解压在 gdb 目录下运行

```
./configure
```

编译安装

```
sudo make install
```

二、调试前准备

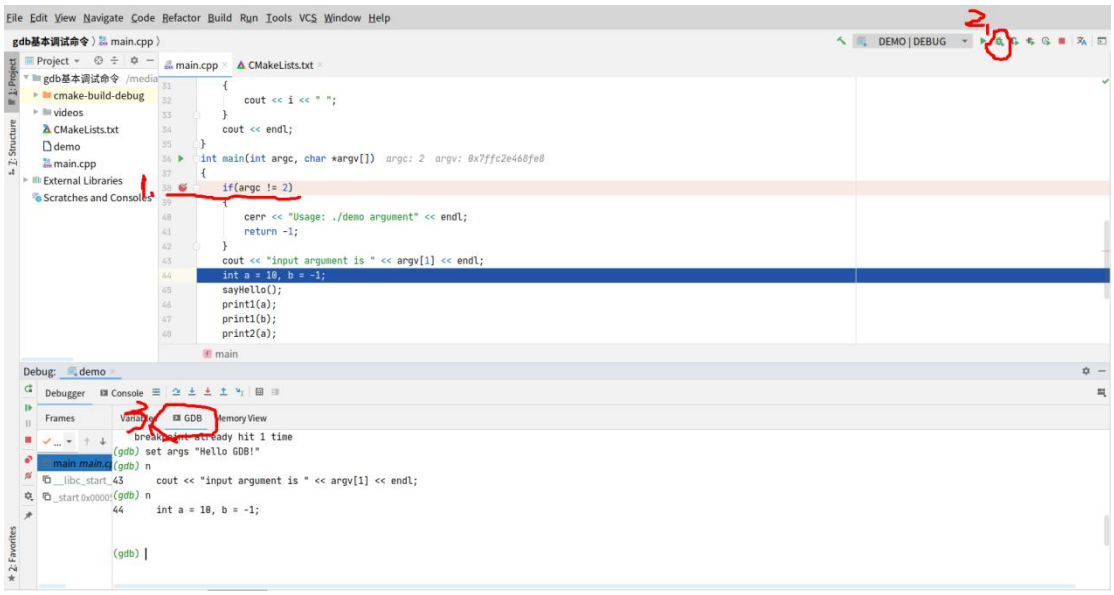
```
gcc -g  demo.c -o demo
```

注意如果是 c++程序则使用 g++进行编译

如果是多进程或者多线程程序，可能需要加上-lpthread 参数

```
gcc -g  demo.c -o demo -lpthread
```

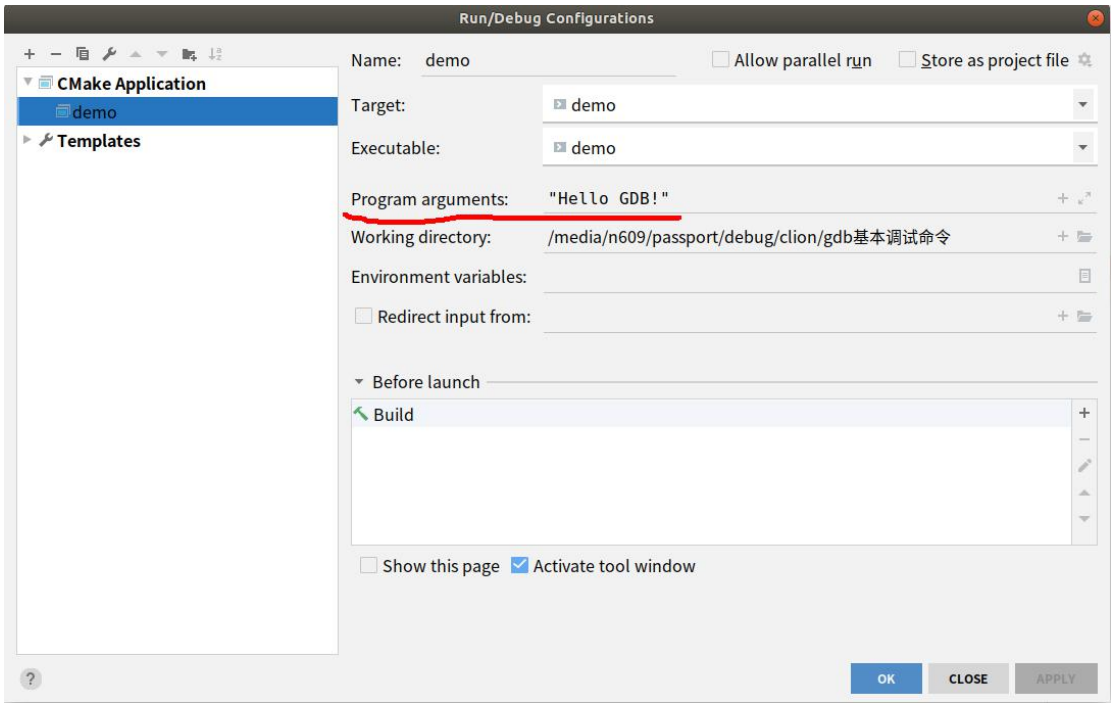
以上是命令行上使用 GDB 的准备，如果是在 CLion 上使用 GDB，先在程序上设置断点，再按 debug 按钮，之后切换到 GDB 调试窗口即可，如下图：



三、基本调试命令

设置程序的参数

原来执行带参程序命令
./demo 参数 1 参数 2
gdb 调试带参程序
gdb demo (gdb) set args 参数 1 参数 2
打印参数
p/print argv[1]
如果参数带有特殊字符，比如空格时，将参数用""包含即可
见 vedio1(CLion 无法在 debug 前用 GDB 设置参数，因此使用命令行演示，先是观看程序，然后先不设置参数执行，然后设置参数执行)
如果要直接用 CLion 设置参数，操作如下：



设置断点

break/b 行号或者函数名或者条件
break/b lineNum break/b functionName
break/b test.c:23 if b == 0 #注意变量需要在生命周期内

开始运行程序

从头开始运行程序，直到遇到断点，否则程序会一直运行下去
run/r

执行当前语句

next/n

打印变量的值

print/p 变量名
print 后面还可以接一个函数表达式，表达式会被执行，表达式不用加；

继续运行程序

继续运行程序直到遇到断点

continue/c

见 vedio2(先是打印参数 argv[1], 然后是设置断点，通过行号设置断点；通过函数名设置断点；运行进入函数内部后通过条件设置断点，单步执行)

Demo 程序

```
#include <iostream>
using namespace std;

void sayHello()
{
    cout << "Hello, World!" << endl;
}

void print1(int n)
{
    if(n <= 0)
    {
        cerr << "Please assert n > 0!" << endl;
        return;
    }
    while(n>0)
    {
        cout << n-- << " ";
    }
    cout << endl;
}

void print2(int n)
{
    if(n <= 0)
    {
        cerr << "Please assert n > 0!" << endl;
        return;
    }
    for(int i=1; i<=n; ++i)
    {
        cout << i << " ";
    }
    cout << endl;
}
int main(int argc, char *argv[])
```

```
{
    if(argc != 2)
    {
        cerr << "Usage: ./demo argument" << endl;
        return -1;
    }
    cout << "input argument is " << argv[1] << endl;
    int a = 10, b = -1;
    sayHello();
    print1(a);
    print1(b);
    print2(a);
    print2(b);
    return 0;
}
```

设置变量的值

```
set var  变量名=新值
```

gdb 支持 tab 补全，支持上下键寻找之前执行的命令

退出

```
quit/q
```

进出函数内部

```
step/s
```

注意如果工程中包含了函数的源代码就可以进去，如果只是包含库但没有源代码则是进不去的

从函数内部退出

```
finish
```

显示源代码

```
list/l
```

见 vedio3，先是打印代码，单步调试，进入函数内部，退出函数内部，单步调试，设置变量值，观察打印的值，结束调试

四、调试多进程程序

程序

```
#include <stdio.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    printf("begin\n");

    if(fork() != 0)
    {
        printf("我是父进程：  pid=%d, ppid=%d\n", getpid(), getppid());

        int ii;
        for(ii=0; ii<10; ii++)
        {
            printf("父进程: ii=%d\n", ii);
            sleep(1);
        }

        exit(0);
    }
    else
    {
        printf("我是子进程: pid=%d, ppid=%d\n", getpid(), getppid());

        int jj;
        for(jj=0; jj<10; jj++)
        {
            printf("子进程: jj=%d\n", jj);
            sleep(1);
        }

        exit(0);
    }
    return 0;
}
```

fork 函数介绍参考 https://blog.csdn.net/jason314/article/details/5640969，简单总结就是：

1、创建一个与原来进程几乎完全相同的进程

2、调用一次，返回两次

(1) 在父进程中，fork 返回创建子进程的进程 ID

(2) 在子进程中，fork 返回 0

(3) 如果出现错误，fork 返回一个负值

选择调试的进程

调试父进程，默认情况下
set follow-fork-mode parent
调试子进程
set follow-fork-mode child

见 vedio4，Clion 有些问题，使用终端运行。先编译程序，开始 gdb 调试，先是默认情况下（设置断点，单步执行，将进入父进程，子进程正常运行），然后更改调试的进程为子进程，运行（设置断点，单步执行，调试的是子进程，父进程正常运行）

设置调试模式

默认是 on，表示调试当前进程时，其它进程会继续运行；
如果设置为 off，则表示调试当前进程时其它进程会被挂起
set detach-on-fork [on|off]

查看和切换调试的进程

查看进程
info inferiors

切换当前调试的进程
inferior 进程编号

终端上调试多进程程序时出现 bug:
Cannot insert breakpoint 1. Cannot access memory at address 0x803

在 CLion 中可以正常使用 set detach-on-fork [on|off]，没有报出上面的 bug，CLion 上 gdb 版本是 8.3

解决办法：安装 gdb 8.3 版本，之前命令行使用的是 8.1.1 版本有问题

见 vedio5，先是设置子进程模式为 off，运行（调试的是父进程，但是此时子进程不会运行），然后是切换到子进程，继续调试，及时打印进程的状态信息

五、调试多线程

程序

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int x = 0, y = 0; // x 用于子线程 1，y 用于子线程 2
```

```
pthread_t pthid1, pthid2;

// 第一个线程的主函数
void *pth1_main(void *arg);

// 第二个线程从主函数
void *pth2_main(void *arg);

int main()
{
    // 创建线程 1
    if(pthread_create(&pthid1, NULL, pth1_main, (void*)0) != 0)
    {
        printf("pthread_create pthid1 failed. \n"); return -1;
    }

    // 创建线程 2
    if(pthread_create(&pthid2, NULL, pth2_main, (void*)0) != 0)
    {
        printf("pthread_create pthid2 failed.\n"); return -1;
    }

    printf("111\n");
    pthread_join(pthid1, NULL); // 等待子线程 1 运行结束

    printf("222\n");
    pthread_join(pthid2, NULL); // 等待子线程 2 运行结束

    printf("333\n");

    return 0;
}

// 第一个线程的主函数
void *pth1_main(void *arg)
{
    for(x=0; x<100; x++)
    {
        printf("子线程 1: x = %d\n", x);
        sleep(1);
    }
    pthread_exit(NULL);
}
```

```
// 第二个线程的主函数
void *pth2_main(void *arg)
{
    for(y=0; y<100; y++)
    {
        printf("子线程 2: y = %d\n", y);
        sleep(1);
    }
    pthread_exit(NULL);
}
```

编译程序

```
gcc -g demo.c -o demo -lpthread
```

Linux 终端查看 demo 相关进程的命令

```
ps aux | grep demo
```

查看线程的命令

```
ps -aL | grep demo
```

查看主线程和子线程之间的关系

```
pstree -p 主线程 id
```

开始调试

分别在主线程、子线程 1、子线程 2 的入口处设置断点

查看当前线程

```
info threads
```

见 vedio6(展示程序，编译程序，调试程序-g -lpthread，查看进程，查看线程，设置断点，同步查看线程情况)

切换线程

```
thread 线程编号
```

注意默认情况下子线程 1 执行的时候，子线程 2 也会在执行

只运行当前线程

其他线程被挂起
set scheduler-locking on

对应的，默认情况下就是运行全部线程，也可以手动设置
set scheduler-locking off

指定某线程执行命令

thread apply 线程编号 gdb 命令

相应的可以让全部线程执行同个 gdb 命令
thread apply all gdb 命令

见 vedio7

六、其他参考学习资料

官方使用教程：https://www.gnu.org/software/gdb/documentation/
中文翻译（陈皓翻译）：链接:https://pan.baidu.com/s/1Cx90LQFfpRwW36nuH-r3wA 提取码:g587