

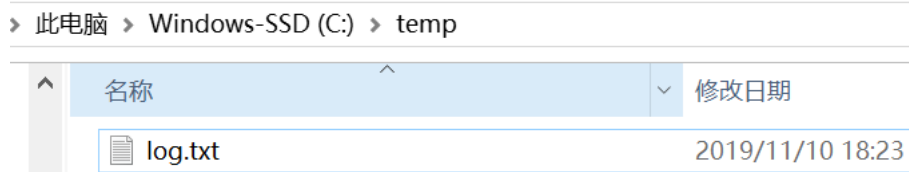
# 实验报告

实验名称	实验七：多线程程序练习																						
<div>1. 攻击面分析：<div>1) 函数 <code>initLog</code>:<ul style="list-style-type: none"><li>➤ 获得的参数 <code>path</code> 中 <code>log</code> 文件路径可能不一定存在，导致创建日志文件失败；</li><li>➤ <code>path</code> 为空，创建也会失败；</li><li>➤ 未关闭文件导致资源泄漏。</li></ul></div><div>2) 函数 <code>logStr</code> :<ul style="list-style-type: none"><li>➤ 若全局变量 <code>PATH</code> 为空（若未初始化），则无法打开日志文件；</li><li>➤ 线程之间数据竞争，导致异常；</li><li>➤ 退出时未对互斥量解锁；</li><li>➤ 未关闭文件导致资源泄漏。</li></ul></div></div>																							
<div>2. 设计思路：<div>1) <code>initLog</code> 流程:<table><tr><th>顺序</th><th>步骤</th></tr><tr><td>①</td><td>先判断 <code>path</code> 是否为空，若为空，返回-1</td></tr><tr><td>②</td><td>获取 <code>path</code> 的文件路径： 将 <code>path</code> 传给 <code>string</code> 类型的变量 <code>spath</code>，调用函数 <code>spath.find_last_of("log.txt")</code>找到"log.txt"所在位置 <code>nPos</code>； 若 <code>nPos</code> 为-1，表示未找到，返回-1； 若不为-1，则调用 <code>substr(0, nPos - 7)</code>截取前面的字符串，即文件路径；</td></tr><tr><td>③</td><td>调用 <code>access</code> 函数判断文件路径是否存在： <code>access(folderPath.c_str(), 0)</code>； 若存在转下一步；若不存在则创建该文件路径：利用 <code>system</code> 函数和 <code>"mkdir -p "</code>命令创建文件夹。</td></tr><tr><td>④</td><td>将 <code>path</code> 赋给全局变量 <code>PATH</code>；利用 <code>ofstream</code> 创建日志文件，<code>is_open()</code> 函数判断是否成功，成功则关闭文件，返回 0，失败返回-1。</td></tr></table></div><div>2) <code>logStr</code> 流程:<table><tr><th>顺序</th><th>步骤</th></tr><tr><td>①</td><td>先判断 <code>PATH</code> 是否为空，若为空，返回-1；</td></tr><tr><td>②</td><td>使用 <code>mutex</code> 库，用互斥锁卫士 <code>lock_guard</code> 看管互斥量；</td></tr><tr><td>③</td><td>使用 <code>time</code> 函数获得时间；设置 <code>tm</code> 指针变量 <code>p</code>，利用 <code>p-&gt;tm_hour</code>、<code>p-&gt;tm_min</code>、<code>p-&gt;tm_sec</code> 分别获得时分秒；</td></tr><tr><td>⑤</td><td>利用 <code>get_id()</code>获得线程 <code>id</code>；</td></tr><tr><td>⑥</td><td>利用 <code>ofstream</code> 打开文件按格式写入数据；</td></tr></table></div></div>		顺序	步骤	①	先判断 <code>path</code> 是否为空，若为空，返回-1	②	获取 <code>path</code> 的文件路径： 将 <code>path</code> 传给 <code>string</code> 类型的变量 <code>spath</code> ，调用函数 <code>spath.find_last_of("log.txt")</code> 找到"log.txt"所在位置 <code>nPos</code> ； 若 <code>nPos</code> 为-1，表示未找到，返回-1； 若不为-1，则调用 <code>substr(0, nPos - 7)</code> 截取前面的字符串，即文件路径；	③	调用 <code>access</code> 函数判断文件路径是否存在： <code>access(folderPath.c_str(), 0)</code> ； 若存在转下一步；若不存在则创建该文件路径：利用 <code>system</code> 函数和 <code>"mkdir -p "</code> 命令创建文件夹。	④	将 <code>path</code> 赋给全局变量 <code>PATH</code> ；利用 <code>ofstream</code> 创建日志文件， <code>is_open()</code> 函数判断是否成功，成功则关闭文件，返回 0，失败返回-1。	顺序	步骤	①	先判断 <code>PATH</code> 是否为空，若为空，返回-1；	②	使用 <code>mutex</code> 库，用互斥锁卫士 <code>lock_guard</code> 看管互斥量；	③	使用 <code>time</code> 函数获得时间；设置 <code>tm</code> 指针变量 <code>p</code> ，利用 <code>p-&gt;tm_hour</code> 、 <code>p-&gt;tm_min</code> 、 <code>p-&gt;tm_sec</code> 分别获得时分秒；	⑤	利用 <code>get_id()</code> 获得线程 <code>id</code> ；	⑥	利用 <code>ofstream</code> 打开文件按格式写入数据；
顺序	步骤																						
①	先判断 <code>path</code> 是否为空，若为空，返回-1																						
②	获取 <code>path</code> 的文件路径： 将 <code>path</code> 传给 <code>string</code> 类型的变量 <code>spath</code> ，调用函数 <code>spath.find_last_of("log.txt")</code> 找到"log.txt"所在位置 <code>nPos</code> ； 若 <code>nPos</code> 为-1，表示未找到，返回-1； 若不为-1，则调用 <code>substr(0, nPos - 7)</code> 截取前面的字符串，即文件路径；																						
③	调用 <code>access</code> 函数判断文件路径是否存在： <code>access(folderPath.c_str(), 0)</code> ； 若存在转下一步；若不存在则创建该文件路径：利用 <code>system</code> 函数和 <code>"mkdir -p "</code> 命令创建文件夹。																						
④	将 <code>path</code> 赋给全局变量 <code>PATH</code> ；利用 <code>ofstream</code> 创建日志文件， <code>is_open()</code> 函数判断是否成功，成功则关闭文件，返回 0，失败返回-1。																						
顺序	步骤																						
①	先判断 <code>PATH</code> 是否为空，若为空，返回-1；																						
②	使用 <code>mutex</code> 库，用互斥锁卫士 <code>lock_guard</code> 看管互斥量；																						
③	使用 <code>time</code> 函数获得时间；设置 <code>tm</code> 指针变量 <code>p</code> ，利用 <code>p-&gt;tm_hour</code> 、 <code>p-&gt;tm_min</code> 、 <code>p-&gt;tm_sec</code> 分别获得时分秒；																						
⑤	利用 <code>get_id()</code> 获得线程 <code>id</code> ；																						
⑥	利用 <code>ofstream</code> 打开文件按格式写入数据；																						

⑦	判断文件是否大于 1M，大于则利用 CopyFile 函数执行备份操作。 关闭文件。 返回 0。
---	--

### 3) 测试用例及结果:

#### ➤ int initLog(char\* path)测试结果:

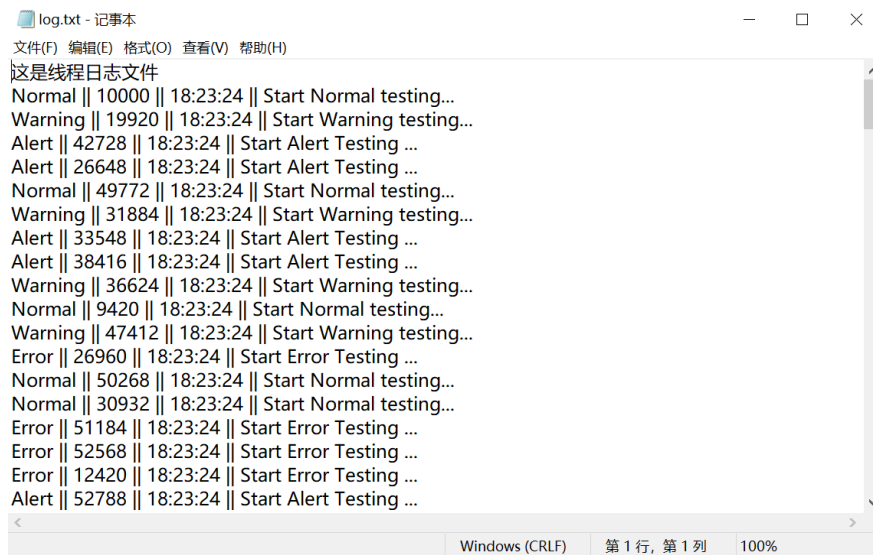


文件创建成功。

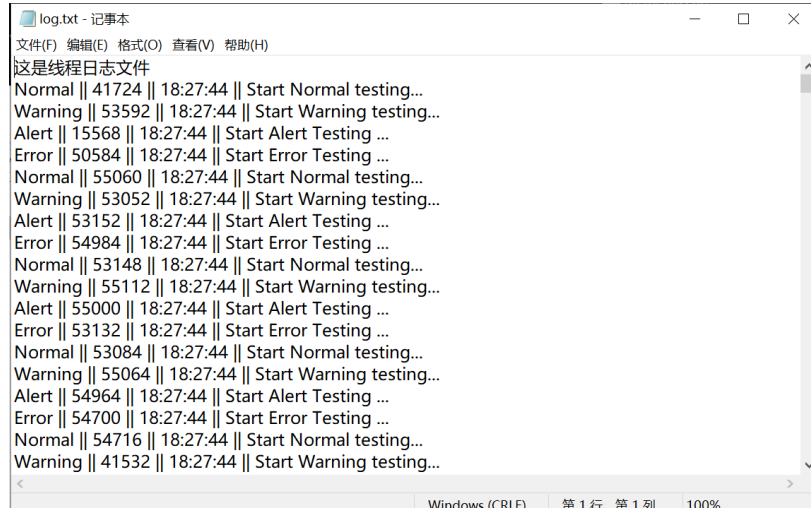
#### ➤ int logStr(char\* level, char\* str)测试用例:

① 创建 40 个进程，调用 detach()函数使子线程独立于主线程并发执行

```
for (int i = 0; i < 10; i++) {
    char c1[] = "Normal", c2[] = "Start Normal testing...";
    thread th1(logStr, c1, c2); th1.join();//th1.detach();
    char c3[] = "Warning", c4[] = "Start Warning testing...";
    thread th2(logStr, c3, c4); th2.join();//th2.detach();
    char c5[] = "Alert", c6[] = "Start Alert Testing ... ";
    thread th3(logStr, c5, c6); th3.join();//th3.detach();
    char c7[] = "Error", c8[] = "Start Error Testing ... ";
    thread th4(logStr, c7, c8); th4.join();//th4.detach();
}
```



② 当线程均调用 join()函数，串行运行时，结果为:



```
log.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
这是线程日志文件
Normal || 41724 || 18:27:44 || Start Normal testing...
Warning || 53592 || 18:27:44 || Start Warning testing...
Alert || 15568 || 18:27:44 || Start Alert Testing ...
Error || 50584 || 18:27:44 || Start Error Testing ...
Normal || 55060 || 18:27:44 || Start Normal testing...
Warning || 53052 || 18:27:44 || Start Warning testing...
Alert || 53152 || 18:27:44 || Start Alert Testing ...
Error || 54984 || 18:27:44 || Start Error Testing ...
Normal || 53148 || 18:27:44 || Start Normal testing...
Warning || 55112 || 18:27:44 || Start Warning testing...
Alert || 55000 || 18:27:44 || Start Alert Testing ...
Error || 53132 || 18:27:44 || Start Error Testing ...
Normal || 53084 || 18:27:44 || Start Normal testing...
Warning || 55064 || 18:27:44 || Start Warning testing...
Alert || 54964 || 18:27:44 || Start Alert Testing ...
Error || 54700 || 18:27:44 || Start Error Testing ...
Normal || 54716 || 18:27:44 || Start Normal testing...
Warning || 41532 || 18:27:44 || Start Warning testing...
Windows (CRLF) 第 1 行, 第 1 列 100%
```

### 3. 心得体会

本次实验，我了解到了 C\C++ 中的多线程操作，了解了并能够简单使用 `thread`、`mutex` 库以及更加熟悉了文件 IO 操作，还了解如何获取时间，也让我更加明白线程并发的机制。