操作系统上机

**实 验 报 告**

|  |  |  |
| --- | --- | --- |
| **评语：**   |  |  | | --- | --- | | **成**  **绩** |  |     **教 师： 李晓**  **2024 年 5 月 26 日** |

**班 级： 2203013 班**

**学 号： 22009200439**

**姓 名： 赵宇阳**

# 操作系统实验报告

**1. 实验题目一 创建进程**

1.1 实验内容

创建两个进程,让子进程读取一个文件,父进程等待子进程读取完文件后 继续执行,实现进程协同工作。

进程协同工作就是协调好两个进程,使之安排好先后次序并以此执行,可 以用等待函数来实现这一点。当需要等待子进程运行结束时,可在父进程中调 用等待函数。

1.2 实验软件环境

**OS:** Manjaro Linux x86\_64

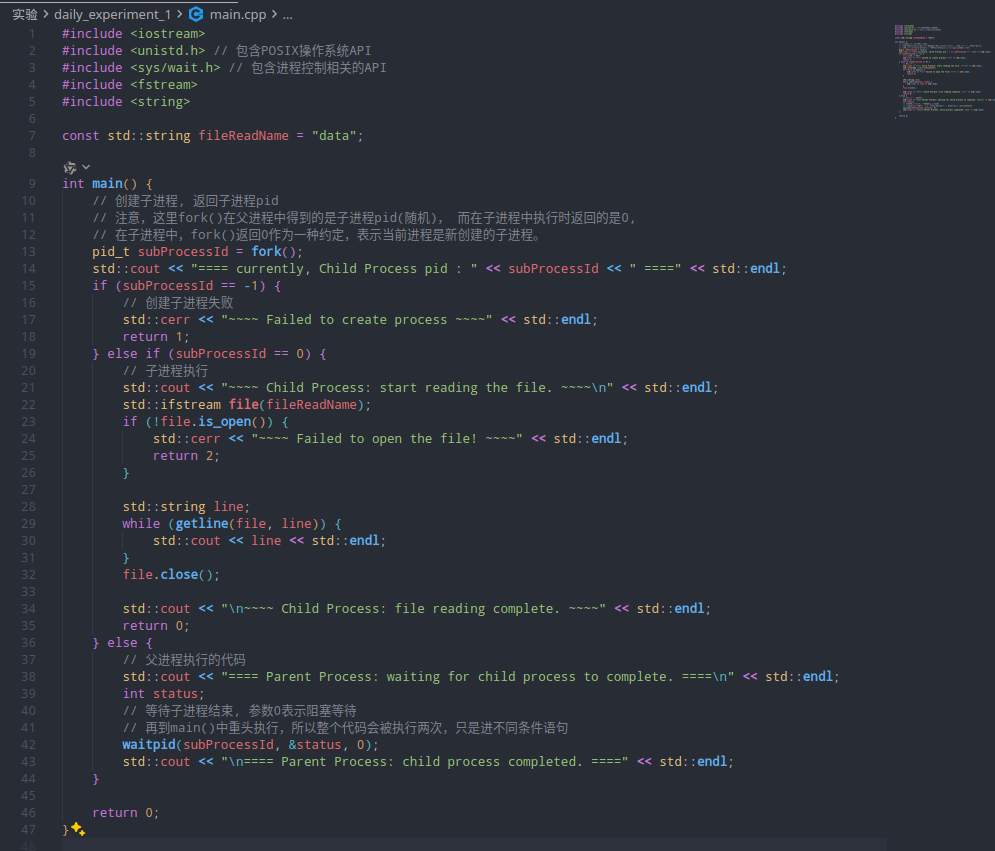
**编译器:** gcc (GCC) 13.2.1 20240417

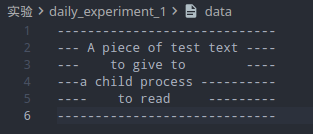
代码编辑: VSCode 1.89.1

1.3 实验方法

**利用fork() 函数创建一个子进程，然后使子进程读取初始设置好的某个文件内容，并打印到标准输出，父进程在子进程完成后继续执行并退出**

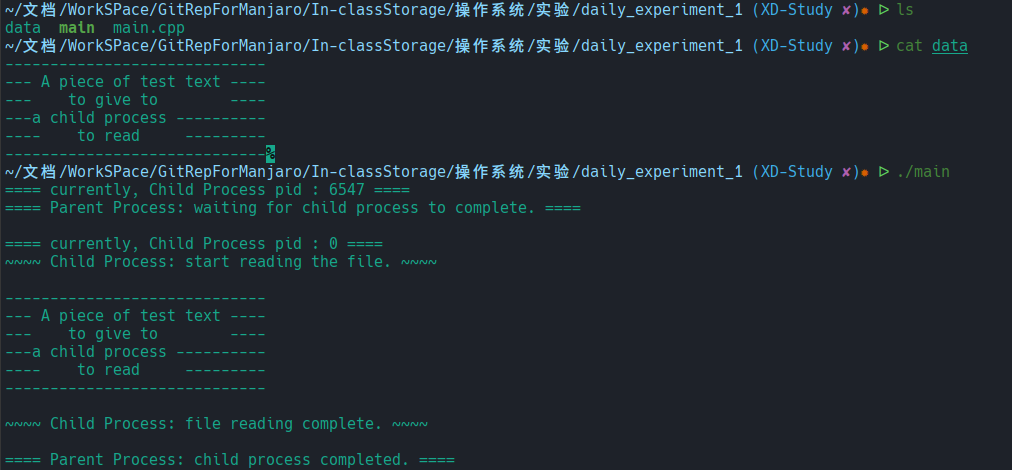
**实验代码：**

**文件内容：**

****

1.4 实验结果展示及分析

实验运行结果：

**分析：**

**创建子进程:**

fork()函数被调用，它会创建一个新的进程。在父进程中，fork()返回新进程的 PID，在子进程中返回0。

如果fork()返回值为-1，表示创建子进程失败，程序通过std::cerr输出错误 信息并返回1。

**子进程逻辑**:

输出提示信息，表明开始读取文件。

打开名为data的文件，如果无法打开，输出错误信息并返回2。

使用getline()逐行读取文件内容，并将每一行输出到标准输出。

完成读取后，关闭文件，输出完成信息，并返回0。

**父进程逻辑**:

输出提示信息，表明正在等待子进程完成。

使用waitpid()函数阻塞等待子进程结束，传入子进程ID和状态变量。

子进程结束后，父进程继续执行，输出子进程完成的信息。

**2. 实验题目二 线程共享进程数据**

2.1 实验内容

在进程中定义全局共享数据,在线程中直接引用该数据进行更改

并输出该数据。

2.2 实验软件环境

**OS:** Manjaro Linux x86\_64

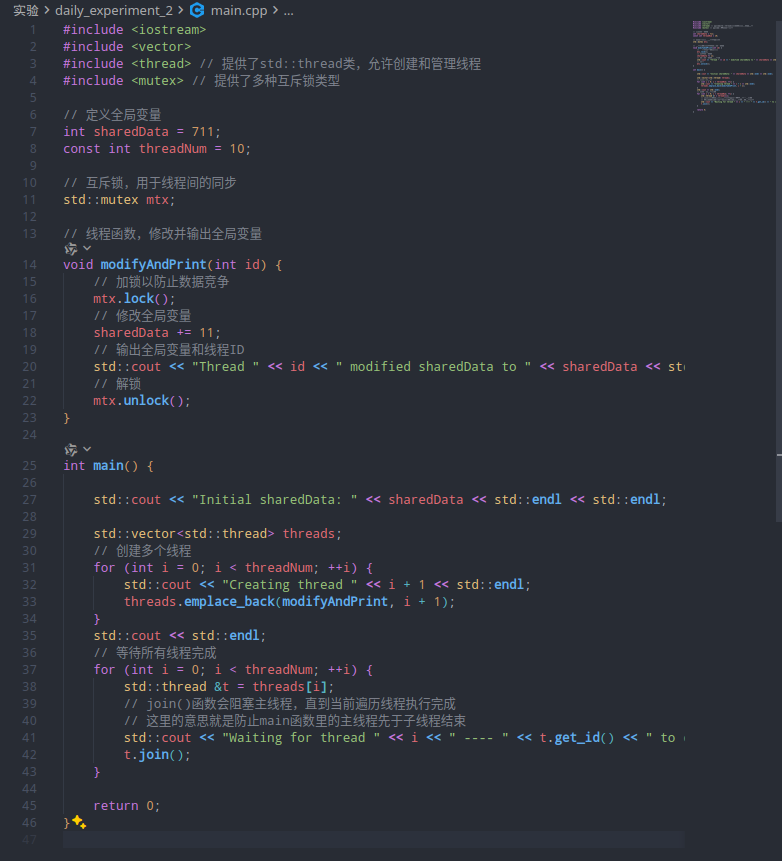
**编译器:** gcc (GCC) 13.2.1 20240417

代码编辑: VSCode 1.89.1

2.3 实验方法：

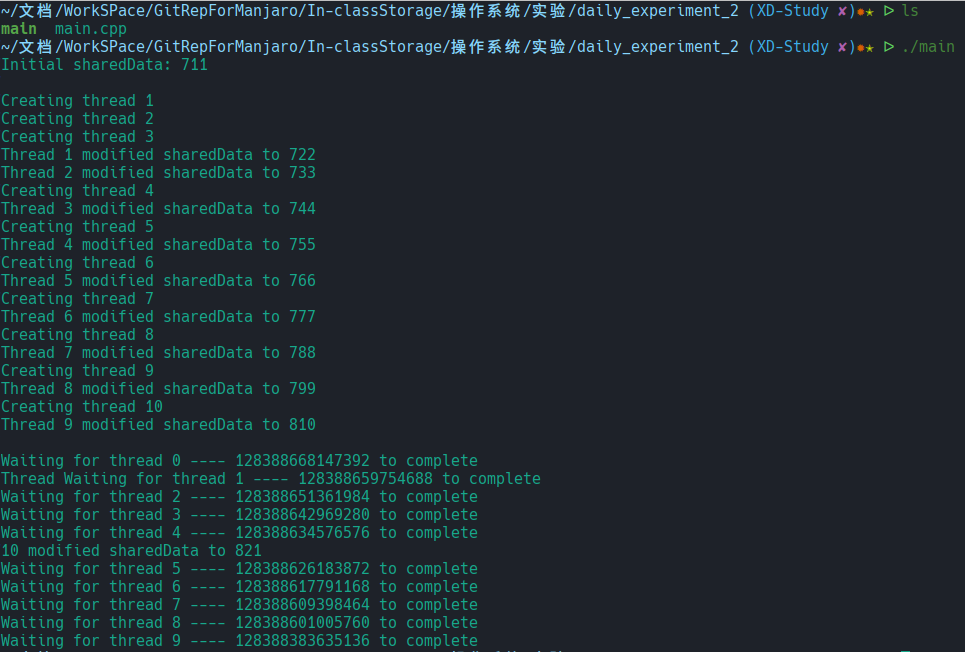
**设置互斥锁用于线程间同步；创建全局共享变量进行数据修改；**

**实验代码：**

****

2.4 实验结果展示及分析

**实验运行结果：**

 **运行分析：**

**初始化**:

定义一个全局整型变量sharedData，初始值为711，用于线程间共享。

设置线程数量threadNum为10

创建一个std::mutex对象mtx，用于线程同步，防止数据竞争。

**线程函数**modifyAndPrint:

接收一个线程ID作为参数。

使用mtx.lock()获取互斥锁，保证对sharedData的独占访问。

将sharedData增加11。

输出线程ID和更新后的sharedData值。

通过mtx.unlock()释放互斥锁，允许其他线程访问sharedData。

**创建线程**:

循环threadNum次，每次创建一个新线程，调用modifyAndPrint函数并将当 前的线程ID（从1开始）作为参数。

使用std::thread::emplace\_back将线程对象添加到threads向量中。

**线程同步**:

使用另一个循环，调用std::thread::join等待每个线程执行完成，确保所有 线程在主线程继续之前结束。

**3. 实验题目三 信号通信**

3.1 实验内容

父进程创建一个有名事件,由子进程发送事件信号,父进程获取

事件信号后进行相应的处理。

3.2 实验软件环境

**OS:** Manjaro Linux x86\_64

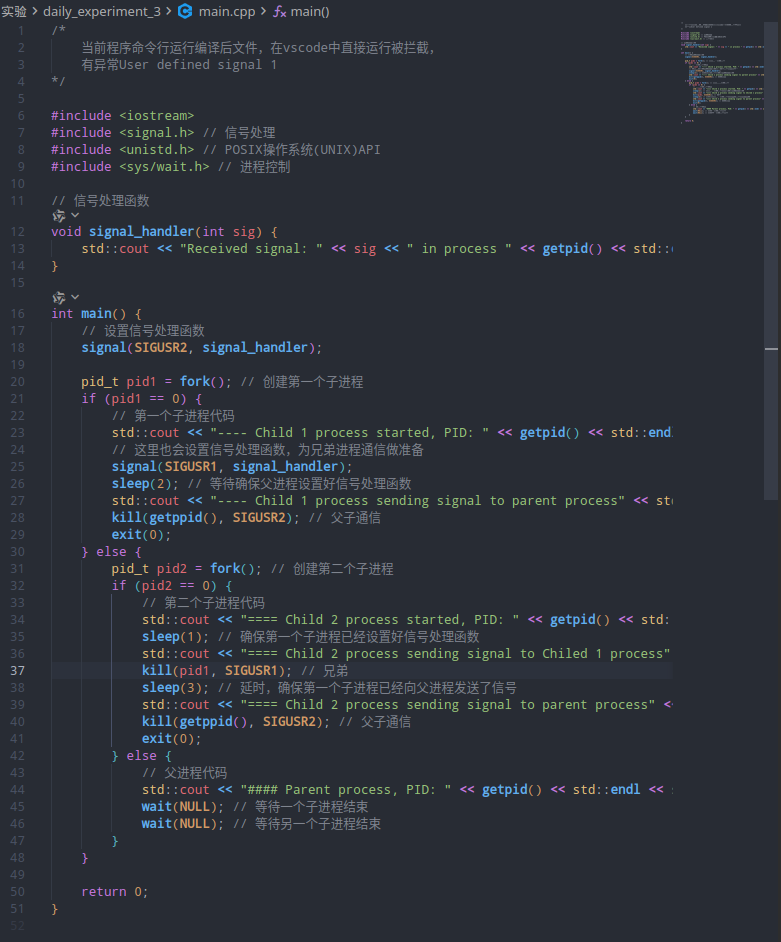
**编译器:** gcc (GCC) 13.2.1 20240417

代码编辑: VSCode 1.89.1

3.3 实验方法

**使用fork()创建子进程；signal()函数设置信号处理函数；kill()函数进行通信**

**实验代码：**

****

3.4 实验结果展示及分析

实验运行结果：

 分析：

**信号处理**:

包含<signal.h>以使用信号处理函数。

定义了signal\_handler函数，当进程接收到信号时，打印信号类型和进程ID

**进程创建**:

包含<unistd.h>和<sys/wait.h>以使用fork()和wait()函数。

使用fork()创建两个子进程（pid1 和 pid2），通过if条件判断确定当前进程是父 进程还是子进程。

**进程间通信**:

子进程1在等待2秒后向父进程发送SIGUSR2信号。

子进程2在等待1秒后向子进程1发送SIGUSR1，并在3秒后向父进程发送 SIGUSR2信号。

**进程同步**:

父进程使用wait(NULL)两次，分别等待两个子进程结束，确保子进程执行完毕。

**4. 实验题目四 匿名管道通信**

4.1 实验内容

分别建立名为 Parent 的单文档应用程序和 Child 的单文档应用程

序作为父子进程,由父进程创建一个匿名管道,实现父子进程向匿

名管道写入和读取数据。

4.2 实验软件环境

**OS:** Manjaro Linux x86\_64

**编译器:** gcc (GCC) 13.2.1 20240417

代码编辑: VSCode 1.89.1

4.3 实验方法

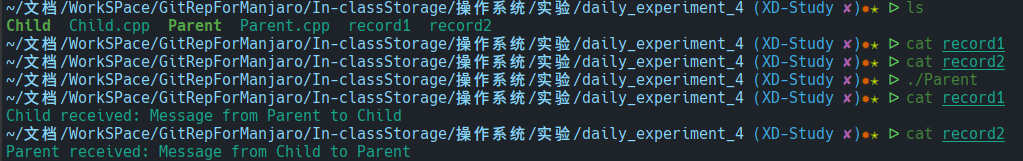
**使用pipe()函数创建管道；使用fork()创建子进程；dup2()重定向进程标 准输入；使用execl()函数执行子程序**

**实验代码：**

****

4.4 实验结果展示及分析

实验运行结果：

 分析：

**管道创建**:

定义两个整型数组pipefd1和pipefd2，分别用于创建两个管道，一个用于父进 程写入子进程读取，另一个用于子进程写入父进程读取。

**子进程操作**:

关闭不需要的管道端（子进程不需要向父进程写入，也不需要从子进程读取）。

使用dup2重定向子进程的标准输入（STDIN\_FILENO）为父进程的管道读端，标 准输出（STDOUT\_FILENO）为子进程的管道写端。

调用execl执行名为Child的程序，替换当前进程。如果执行失败，输出错误信 息并退出子进程。

**父进程操作**:

关闭不需要的管道端（父进程不需要从子进程读取，也不需要向子进程写入）。

创建一个ofstream对象record，用于记录接收的数据。

向子进程的管道写端写入数据（write函数）。

从子进程的管道读端读取数据（read函数）并记录到文件。

关闭管道和文件，等待子进程结束（wait(NULL)）。

**5. 实验题目五 命名匿名管道通信**

5.1 实验内容

建立父子进程,由父进程创建一个命名匿名管道,由子进程向

命名管道写入数据,由父进程从命名管道读取数据。

5.2 实验软件环境

**OS:** Manjaro Linux x86\_64

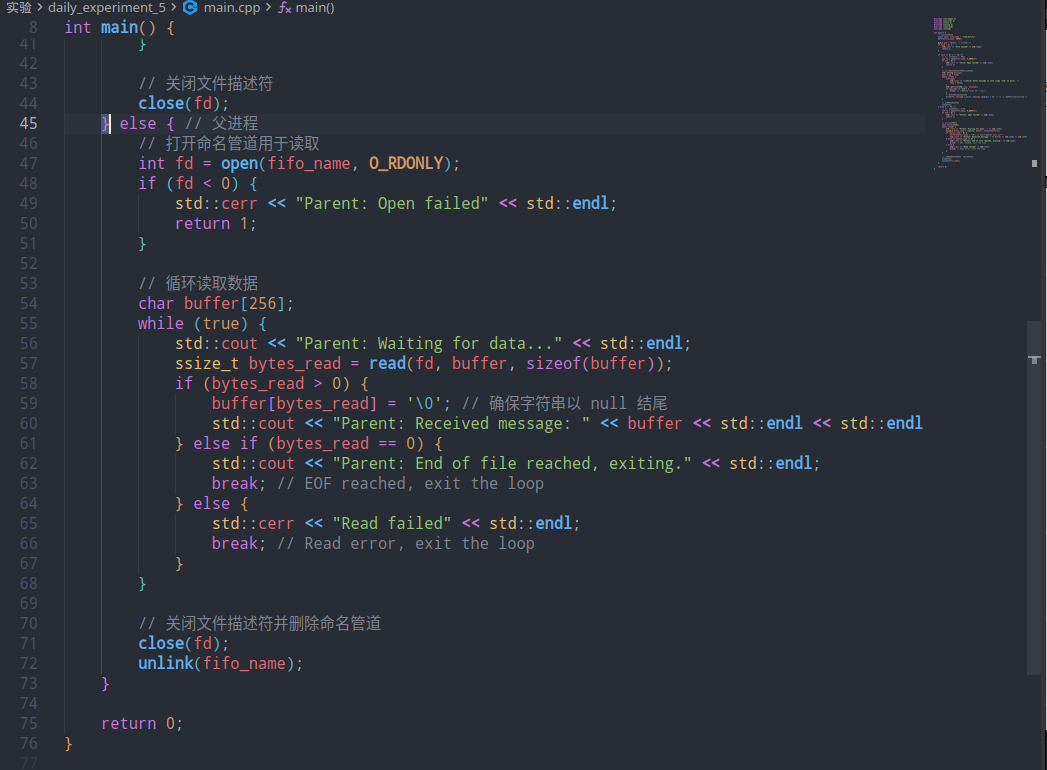
**编译器:** gcc (GCC) 13.2.1 20240417

代码编辑: VSCode 1.89.1

5.3 实验方法

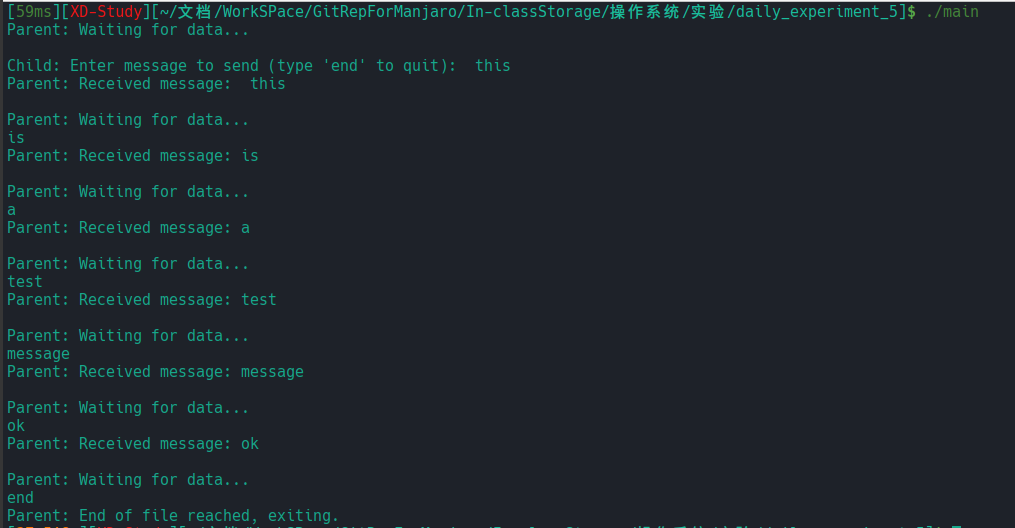
使用mkfifo()函数创建命名管道；fork()创建子进程；子进程打开命名管道进行写操作；父进程打开命名管道进行读操作；

实验代码：



**5.4 实验结果展示及分析**

**实验结果：**

**分析：**

**命名管道创建:**

定义了一个常量fifo\_name，表示命名管道的路径。

使用mkfifo函数创建一个名为/tmp/myfifo的命名管道，权限设为0666。

**创建子进程**:

使用fork()创建一个子进程。如果fork()失败，输出错误信息并退出。

**子进程操作**:

打开命名管道以进行写入操作。

从用户获取输入，直到用户输入"end"为止，将输入的文本写入命名管道。

写入完成后，关闭文件描述符。

**父进程操作**:

打开命名管道以进行读取操作。

循环读取管道中的数据，直到遇到EOF（文件结束）或读取错误。

将读取到的数据打印到标准输出。

读取完成后，关闭文件描述符并删除命名管道。

**6. 实验题目三 信号量实现进程同步**

**3.1 实验内容**

生产者进程生产产品,消费者进程消费产品。

• 当生产者进程生产产品时,如果没有空缓冲区可用,那么生产

者进程必须等待消费者进程释放出一个缓冲区。

• 当消费者进程消费产品时,如果缓冲区中没有产品,那么消费

者进程将被阻塞,直到新的产品被生产出来。

**3.2 实验软件环境**

**OS:** Manjaro Linux x86\_64

**编译器:** gcc (GCC) 13.2.1 20240417

代码编辑: VSCode 1.89.1

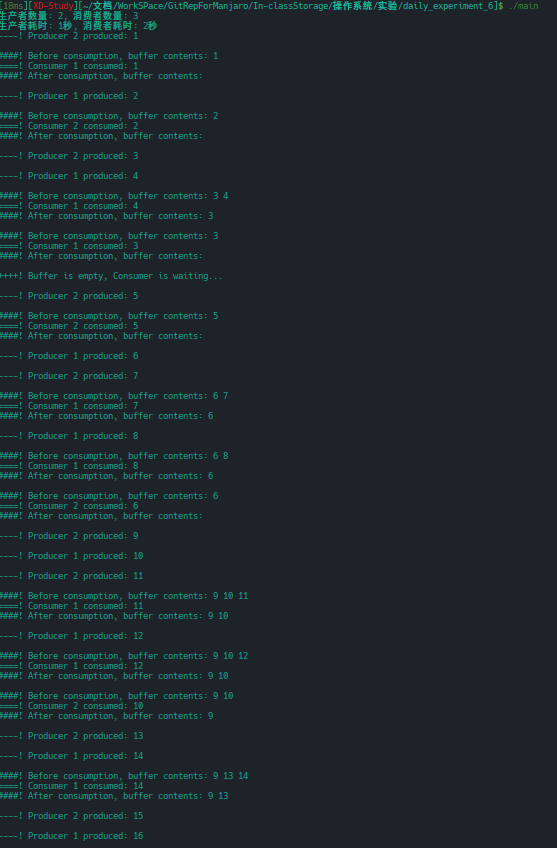
**3.3 实验方法**

**实验代码：**

****

****

**3.4 实验结果展示及分析**

 **实验运行结果：**

**分析：**

**全局变量**:

BUFFER\_SIZE：缓冲区大小。

MAX\_PRODUCTS：产品总数上限。

productsProduced 和 productsConsumed：原子变量，分别记录已生产的产 品数和已消费的产品数。

buffer：存储产品的缓冲区，使用std::vector实现。

product，empty 和 mutex：三个信号量，product用于表示产品数量，empty 表示空闲空间数量，mutex用于保护共享资源（缓冲区）的访问。

stopRequested：原子布尔变量，用于控制程序的停止。

**函数**:

producer：生产者线程函数，生成产品并将其放入缓冲区，直到达到最大产品数 量或接收到停止请求。

consumer：消费者线程函数，从缓冲区取出并消费产品，直到达到最大产品数量 或接收到停止请求。

stopListener：监听线程函数，等待用户输入，当用户按下回车键时设置 stopRequested为true，停止生产者和消费者线程。

**主函数**:

初始化信号量和线程。

创建指定数量的生产者和消费者线程，以及一个监听线程。

当所有生产者和消费者线程都结束后，销毁信号量并退出。

**执行流程**:

生产者线程生成产品，检查缓冲区是否有空间，如果有则将产品放入缓冲区并更新 相关计数。

消费者线程检查缓冲区是否有产品，如果有则取出并消费，同时更新相关计数。

监听线程等待用户输入，当用户按下回车键时，设置停止标志，通知生产者和消费 者线程停止工作。

**7. 实验题目七 共享主存实现进程通信**

**3.1 实验内容**

**为基于共享主存解决读者-写着问题,需要由写进程首先创建一**

**个共享主存,并将该共享主存区映射到虚拟地址空间,随后读进程**

**打开共享主存,并将该共享主存区映射到自己的虚拟地址空间,从**

**中获取数据,并进行处理,以此实现进程通信。**

**3.2 实验软件环境**

**OS: Manjaro Linux x86\_64**

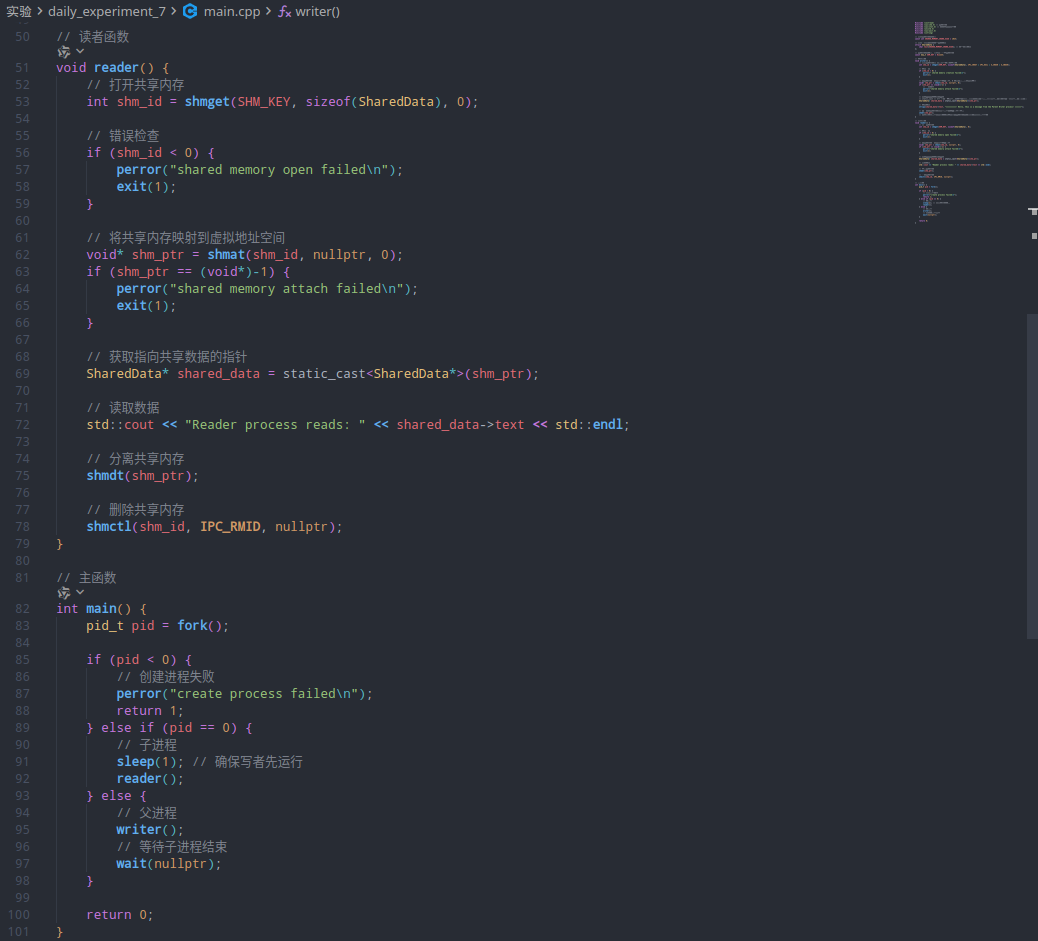
**编译器: gcc (GCC) 13.2.1 20240417**

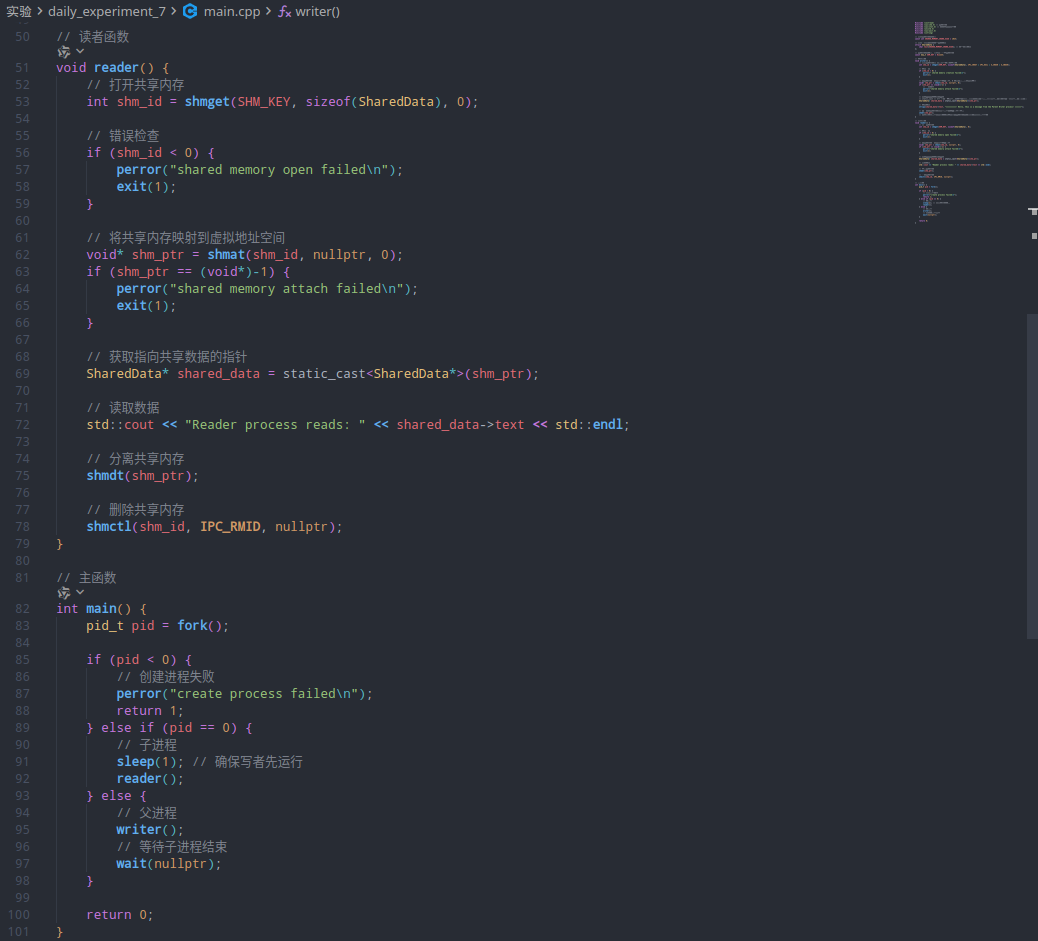
**代码编辑: VSCode 1.89.1**

**3.3 实验方法**

**创建共享内存并映射到虚拟地址空间**

**实验代码：**

****

****

**3.4 实验结果展示及分析**

**实验运行结果：**

**分析：**

**常量和结构体**:

SHARED\_MEMORY\_CHUNK\_SIZE：定义共享内存块的大小。

SharedData：定义一个结构体，包含一个char数组用于存储文本数

SHM\_KEY：定义共享内存的键，用于唯一标识共享内存。

**写者函数（writer）**:

使用shmget创建共享内存，设置权限（只读/只写）并检查是否成功。

使用shmat将共享内存映射到当前进程的地址空间。

将数据写入共享内存的text字段。

使用shmdt从进程地址空间中分离共享内存。

**读者函数（reader）**:

使用shmget打开已存在的共享内存，检查是否成功。

使用shmat将共享内存映射到当前进程的地址空间。

从共享内存中读取数据并打印。

使用shmdt从进程地址空间中分离共享内存。

使用shmctl删除共享内存。

**主函数（main）**:

使用fork创建子进程。

父进程（writer）运行writer函数，写入数据到共享内存。

子进程（reader）睡眠1秒，等待父进程写入数据，然后运行reader函数，读取并打 印数据。

父进程等待子进程结束。