2020-2021第2学期

操作系统实验报告

班 级： 人工智能2019-01班

学 号： 2019113035

姓 名： 龚傲林

专 业： 人工智能

任课教师： 胡晓鹏

二零二一年六月

目 录

[实验1 Linux环境及简单C语言程序调试 1](#_Toc75106585)

[1.1实验目的 1](#_Toc75106586)

[1.2实验内容 1](#_Toc75106587)

[1.2.1常见命令使用 1](#_Toc75106588)

[1.2.2 编写并运行简单的C语言程序 3](#_Toc75106589)

[1.3实验总结 3](#_Toc75106590)

[实验2 Linux进程创建及多进程并发 4](#_Toc75106591)

[2.1实验目的 4](#_Toc75106592)

[2.2实验内容 4](#_Toc75106593)

[2.2.1使用C语言，在父进程中创建一个子进程 4](#_Toc75106594)

[2.2.2在父子进程中分别调用getpid和getppid 4](#_Toc75106595)

[2.3实验总结 6](#_Toc75106596)

[实验3 Linux线程创建及多线程并发 7](#_Toc75106597)

[3.1实验目的 7](#_Toc75106598)

[3.2实验内容 7](#_Toc75106599)

[3.2.1创建并使用VScode工具编写代码 7](#_Toc75106600)

[3.2.2程序执行结果 8](#_Toc75106601)

[3.3实验总结 8](#_Toc75106602)

[实验4 Linux软中断通信 9](#_Toc75106603)

[4.1实验目的 9](#_Toc75106604)

[4.2实验内容 9](#_Toc75106605)

[4.2.1代码编写与分析 9](#_Toc75106606)

[4.2.2运行结果与分析 9](#_Toc75106607)

[4.3实验总结 10](#_Toc75106608)

[实验5 基于共享内存的进程通信 11](#_Toc75106609)

[5.1实验目的 11](#_Toc75106610)

[5.2实验内容 11](#_Toc75106611)

[5.2.1代码编写与分析 11](#_Toc75106612)

[5.2.2运行结果与分析 12](#_Toc75106613)

[5.3实验总结 12](#_Toc75106614)

[实验6 时间片轮转调度算法 14](#_Toc75106615)

[6.1实验目的 14](#_Toc75106616)

[6.2实验内容 14](#_Toc75106617)

[6.2.1代码编写与分析 14](#_Toc75106618)

[6.2.2运行结果与分析 15](#_Toc75106619)

[6.3实验总结 15](#_Toc75106620)

实验1 Linux环境及简单C语言程序调试

1.1实验目的

了解Linux系统的组织和行为，包括Linux的用户界面、目录结构组织、运行程序的基本方式。

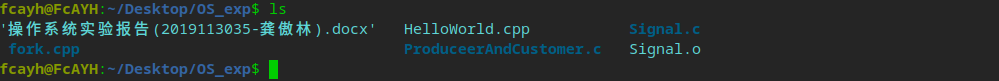
1.2实验内容

* 通过使用终端（terminal）熟悉Linux Shell、常见命令（cd、ls、ps、chmod等和redirect、pipe机制
* 编写并调试一个简单的C语言程序
  + 使用vi或者gedit等编辑器编辑一个C语言源程序，并使用gcc对源程序进行编译，运行生成的可执行文件并熟悉调试方法

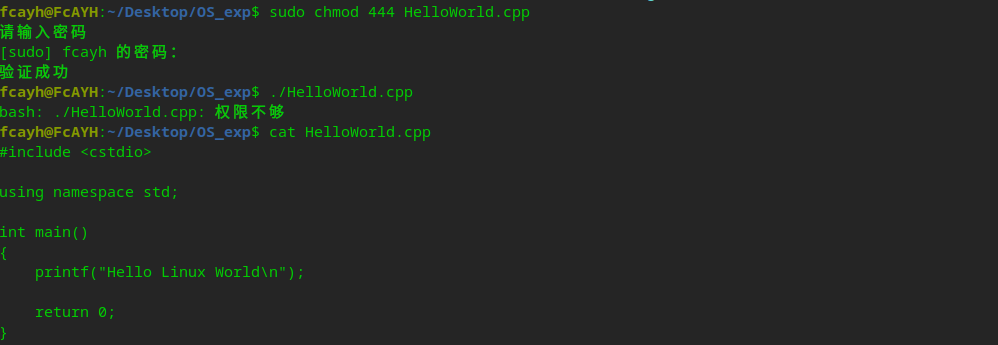
1.2.1常见命令使用

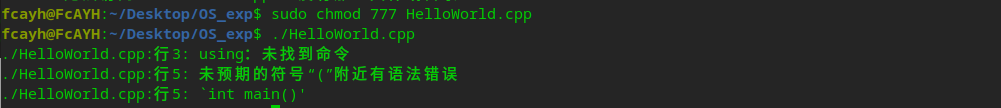
cd:

ls:

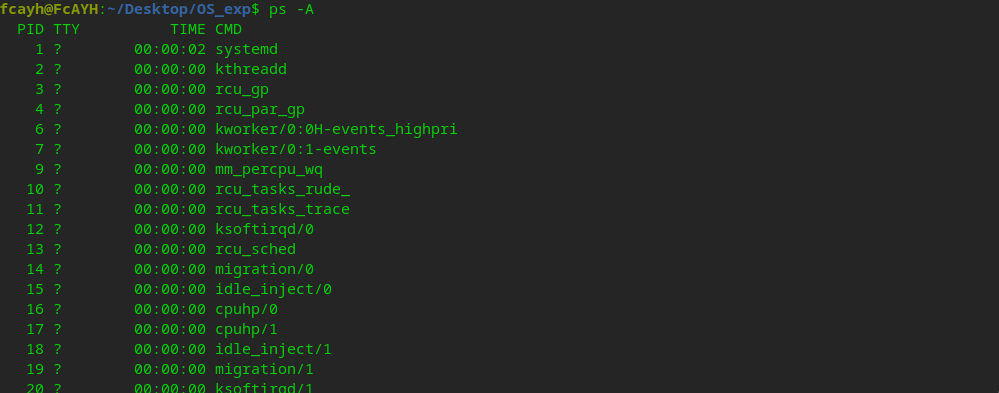


chmod:



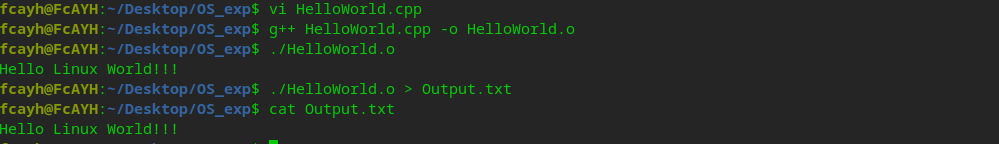


ps:



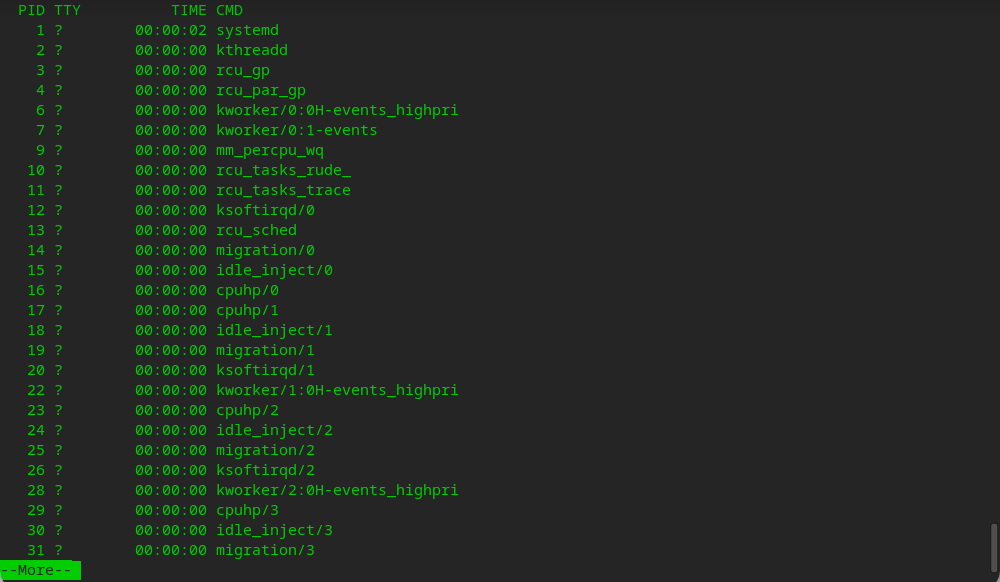
redirect:





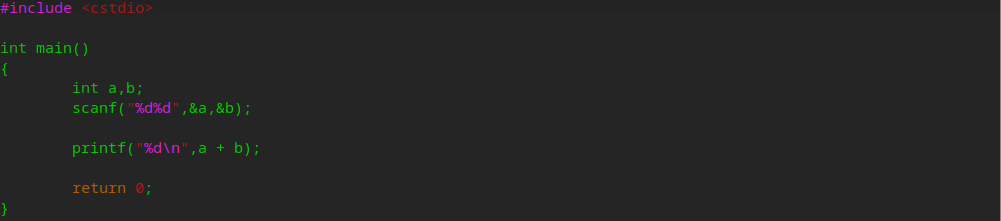
pipe:



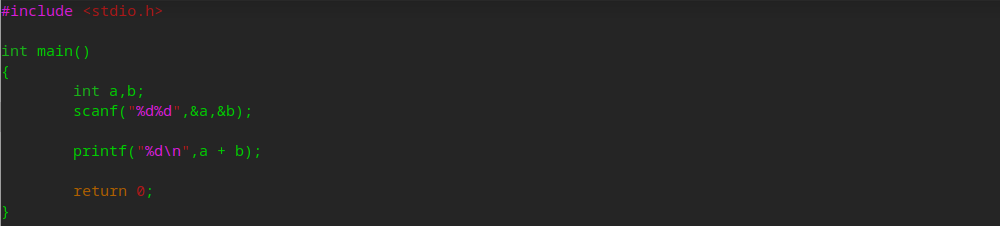
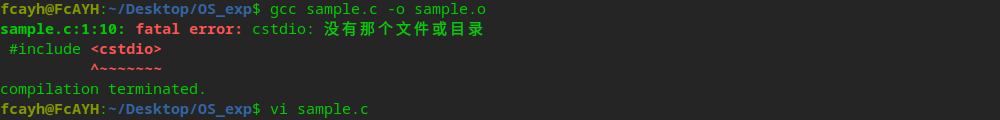


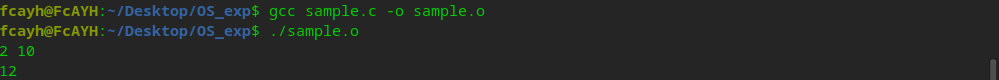
1.2.2 编写并运行简单的C语言程序

截图_选择区域_20210523164115



编译报错与修复：





1.3实验总结

在这次实验中，我学习了Linux操作系统的基本操作方式，掌握了诸如cd、ls、cat、chmod、ps等基本指令，尝试了使用redirect和pipe机制阅读和写入文件。并且在最后使用vim工具和C语言编写了简单的a+b程序，并通过gcc指令编译，修正语法错误后正确运行。

实验2 Linux进程创建及多进程并发

2.1实验目的

了解Linux系统调用fork的工作原理

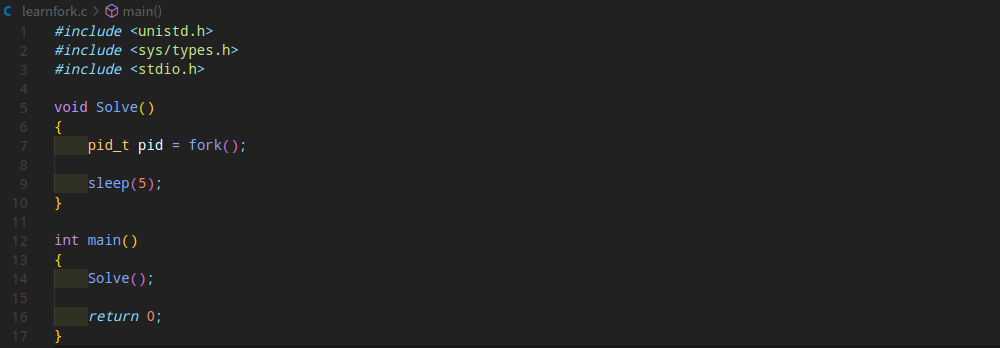
2.2实验内容

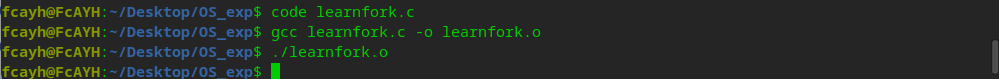
* 在Linux上编写C语言程序，实现在父进程中调用fork创建一个子进程
* 在父子进程中分别调用getpid、getppid返回父子进程的进程号、并检查返回的父子进程号所代表的父子关系是否一致/对应。如果不一致则分析其中的问题，并对源程序进行修改，使得返回的进程号与父子进程关系一致，从而更好的理解并发。

2.2.1使用C语言，在父进程中创建一个子进程



使用vscode工具编写代码：



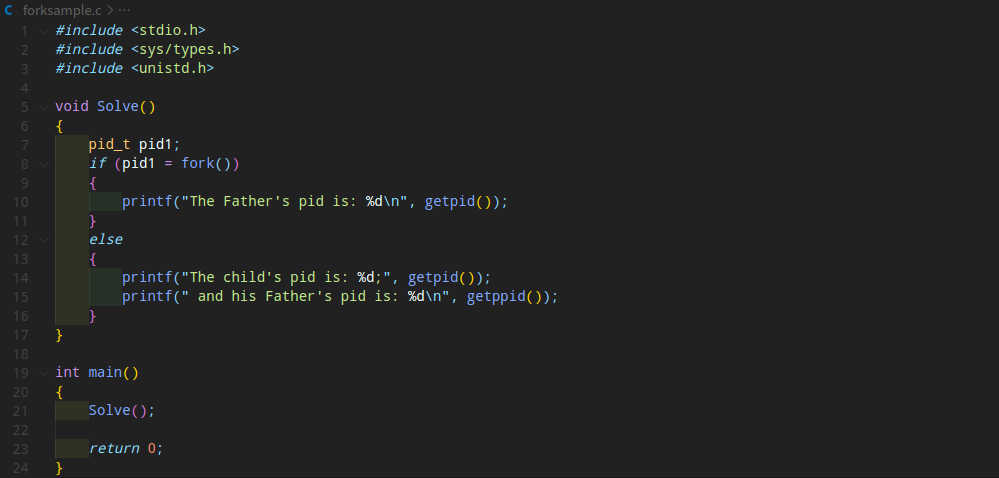


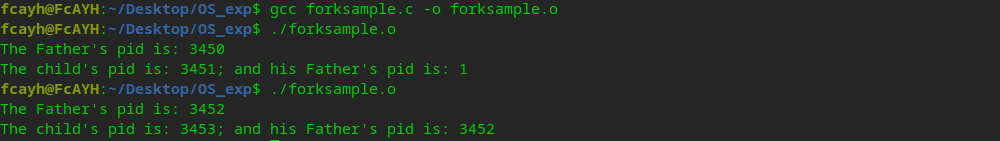


2.2.2在父子进程中分别调用getpid和getppid

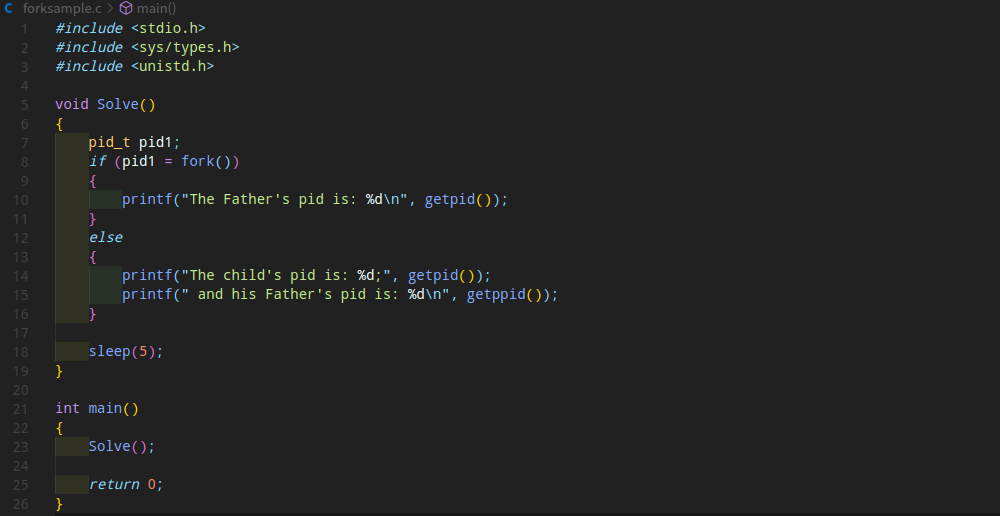


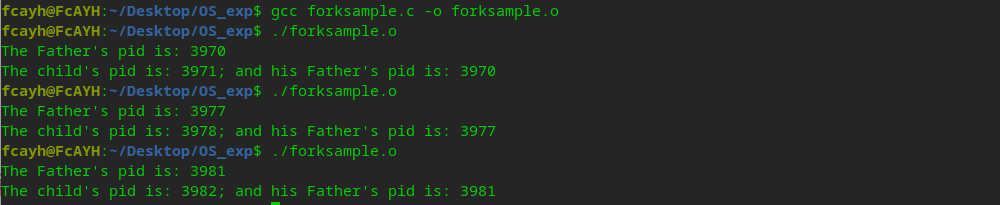
使用vscode工具编写代码：





此时我们发现，重复运行程序，有时在子进程中输出的父亲的进程号与在父进程中输出自身的进程号相同，有时则不同且为1。经过学习、思考与反复调试程序，得出结论：是由于父进程和子进程同时运行，导致有时子进程较快运行结束，自此当子进程调用getppid时，父进程仍在运行，故而可以输出正确的父进程进程号。而有时父进程运行较快，当子进程调用getppid时，父进程已经运行结束了，故而此时子进程的父进程变成系统的systemd进程，其进程号为1。解决方案：在代码中加入sleep(5)使程序延时5s，确保子进程调用getppid时，父进程没有被杀死。





2.3实验总结

通过此次实验，我了解了Linux下fork()函数的原理，在 pid = fork() 之前，仅仅有一个进程在运行，但在这条语句之后。就变成两个进程在运行了，这两个进程共享代码段。将要运行的下一条语句就是 pid = fork() 之后的语句。两个进程中，原来就存在的那个进程被称作为”父进程”,新出现的那个进程被作为”子进程”,父子进程的差别在于进程标识符(PID)不同。子进程的数据空间、堆栈空间都会从父进程得到一个拷贝，而不是共享。并且子进程与父进程的运行也是独立的。除此之外，我还掌握了使用getpid()函数来获取当前进程的pid，使用getppid()函数来获取其父亲的pid。

实验3 Linux线程创建及多线程并发

3.1实验目的

了解Linux多线程的工作原理。

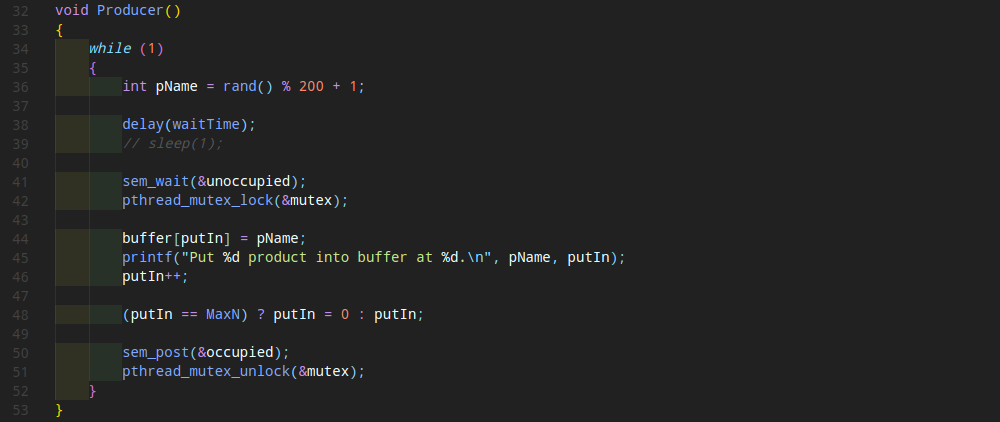
3.2实验内容

* 在Linux上使用C语言编写一个多线程程序，并通过信号量机制来模拟解决一个有界缓冲区的生产者——消费者问题。其中一个线程代表生产者，另外一个线程代表消费者，使得生产者和消费者线程可以周而复始的工作而不出现错误。

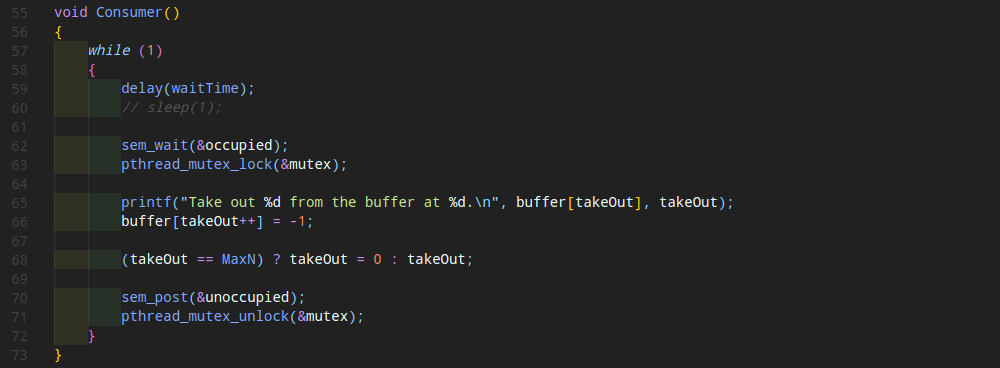
3.2.1创建并使用VScode工具编写代码



生产者(producer):



消费者(consumer):



具体操作过程为：

建立两个线程，分别执行Producer函数和Consumer函数。在Producer函数中，生产产品，并放到储物架（即缓冲区）的后一位，若已满，则再第一位开始放。此处利用信号量机制，保证互斥，即将货物放到货架上的过程中消费者不能取物件。在Consumer函数中，消费者从货架的当前位置依次向后取货物，如果取到货架的最后一位，则再从第一位开始取。此处同样利用了信号量机制，保证互斥，即消费者取物件的时候，生产者无法将物件置于货架。

3.2.2程序执行结果

编译并运行程序，将结果输出到.txt文件中：



运行结果：



3.3实验总结

通过此次实验，首先我学习了在Linux下使用pthread\_create创建新的线程并为其分配一个函数去执行，其次掌握了利用信号量机制模拟解决一个有界缓冲区访问冲突问题。不仅加深了对多线程工作的理解，还同时掌握了诸多库函数的使用方式。

实验4 Linux软中断通信

4.1实验目的

* 了解Linux软中断（Signal）的工作原理;

4.2实验内容

* 在Linux上使用C语言编写一个包含信号处理机制的程序，在该程序中可以安装自己的信号处理程序；在此基础上发送一个对应的信号，验证自己安装的信号处理程序是否正常执行，从而加深对信号处理机制的理解。

4.2.1代码编写与分析

使用VScode工具编写代码：



通过fork()函数，在父进程中创建两个子进程。通过signal机制，实现当接受到SIGINT信号后，分别给两个子进程发送SIGUSR1/2信号使其结束，并随后将自己结束。

父进程设置signal：



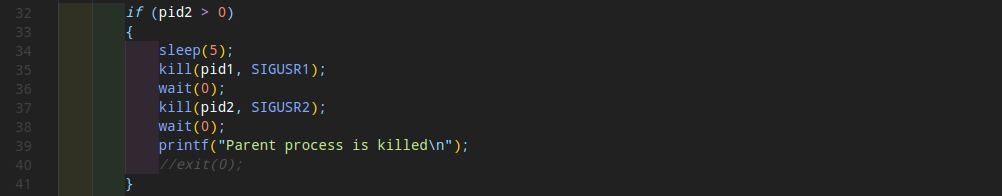
子进程1设置signal：



子进程2设置signal：

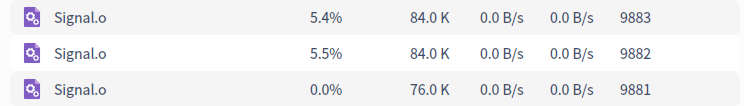


父进程发送signal关闭子进程：



4.2.2运行结果与分析

父进程(9881)创建了两个子进程(9882)和(9883)。



按下ctrl + c后，第一行的”Interruption”为父进程接收到SIGINT信号后，执行stop1函数的结果。第二行的”Child Interuption”为第一个子进程接收到父进程发出的SIGUSR1信号后，执行stop2函数的结果，随后子进程1被结束掉。同理，父进程发送SIGUSR2信号后，将子进程2结束掉。随后父进程被结束掉。



4.3实验总结

通过此次实验，掌握了软中断的基本原理。首先复习了利用fork()创建新进程的具体操作与原理，其次了解了Signal函数中所提供的常用信号。起初误以为Signal函数必须在执行时收到信号才起作用，故而犯了很多错误，通过学习了解到signal的功能是将某个软中断信号**与某个可执行的处理函数进行关联**，当信号发出并被指定的进程接收后，系统就中断接收该软件中断信号进程的执行，转而执行与信号相关联的函数，该函数执行完毕后再返回被中断的进程继续执行。同时，除了用户定义信号 SIGUSR1 和 SIGUSR2 外，其他软中断信号都已经由操作系统预置了相应的处理函数，用户进程如果对这些软中断信号进行了预置，则使该信号与新的函数进行关联，当该软中断信号被接收时，转而执行的不再是操作系统预置的处理函数，而是用户对该软中断信号重新预置的处理函数。而对于同一个软中断信号，可以通过多个 signal() 系统调用分别与不同的处理函数进行关联。系统在响应该软中断信号时，**执行的是当前预置的处理函数**(最近预置的),从而实现同一软中断信号在不同的情况下转向不同的处理函数去执行。

实验5 基于共享内存的进程通信

5.1实验目的

* 了解Linux共享内存（Shared Memory）进程进程间通信的基本原理。

5.2实验内容

* 在Linux上使用C语言编写两个程序，其中一个程序（进程）创建一个共享内存区域，另一个程序（进程）从该共享区域读出内容，验证写入的内容与读出的内容是否一致，从而加深对共享内存工作原理的理解。

5.2.1代码编写与分析

创建shmwrite.c文件用于实现向内存中写入数据的功能。

创建shmread.c文件用于实现从内存中读取数据的功能。



shmwrite.c具体实现过程：

1. 创建共享内存；



1. 将共享内存连接到当前进程的地址空间；



1. 当数据未处于读取状态时，向其中写入数据。



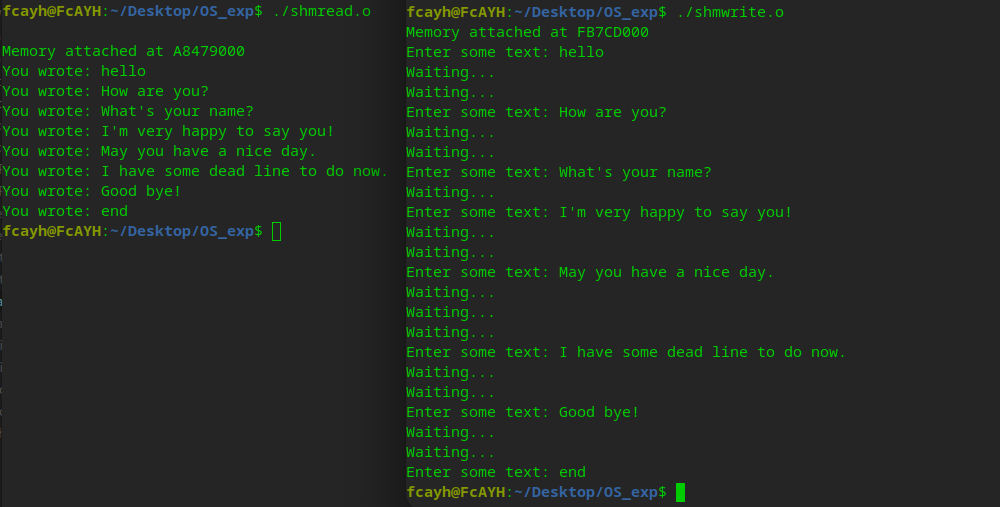
shmread.c具体实现过程：

1. 创建共享内存；
2. 将共享内存连接到当前进程的地址空间；
3. 在共享内存中被写入数据后，读取其中的内容。



5.2.2运行结果与分析

同时打开两个终端，分别编译运行shmwrite.c和shmread.c。如下图所示，每当在shmwrite的窗口中输入一段字符串，都可以立即在shmread的窗口中获得该数据。



5.3实验总结

通过本次实验，我了解了什么是共享内存，并且上手实践了如何编写代码创建并使用共享内存，从而实现进程间通信。

其中我学习了与共享内存有关的基本函数：

1. shmget ( )：创建共享内存

int shmget(key\_t key, size\_t size, int shmflg);

key：由ftok生成的key标识，标识系统的唯一IPC资源。

size：需要申请共享内存的大小。

shmflg：如果要创建新的共享内存，需要使用IPC\_CREAT，IPC\_EXCL，如果是已经存在的，可以使用IPC\_CREAT或直接传0。

返回值：成功时返回一个新建或已经存在的的共享内存标识符，取决于shmflg的参数。失败返回-1并设置错误码。

1. shmat ( )：挂接共享内存

void \*shmat(int shmid, const void \*shmaddr, int shmflg);

shmid：共享存储段的标识符。

\*shmaddr： shmaddr = 0，则存储段连接到由内核选择的第一个可以地址上。

shmflg：若指定了SHM\_RDONLY，则以只读方式连接此段，否则以读写方式连接此段。

返回值：成功返回共享存储段的指针（虚拟地址），并且内核将使其与该共享存储段相关的shmid\_ds结构中的shm\_nattch计数器加1（类似于引用计数）；出错返回-1。

1. shmdt ( )：去关联共享内存

当一个进程不需要共享内存的时候，就需要去关联。该函数并不删除所指定的共享内存区，而是将之前用shmat函数连接好的共享内存区脱离目前的进程。

int shmdt(const void \*shmaddr);

\*shmaddr：连接以后返回的地址。

返回值：成功返回0，并将shmid\_ds结构体中的 shm\_nattch计数器减1；出错返回-1。

1. shmctl ( )：销毁共享内存

int shmctl(int shmid, int cmd, struct shmid\_ds \*buf);

shmid：共享存储段标识符。

参数cmd：指定的执行操作，设置为IPC\_RMID时表示可以删除共享内存。

\*buf：设置为NULL即可。

返回值：成功返回0，失败返回-1。

实验6 时间片轮转调度算法

6.1实验目的

* 了解时间片轮转调度算法的工作原理。

6.2实验内容

* 在Linux上编写C语言，实现从键盘输入时间片长度，任务个数，每一个任务的到达时间及服务时间;
* 构造相应的进程并按时间片轮转调度算法对所有进程进行管理，进程运行情况可以输出到终端，从而深入理解时间片轮转调度算法的原理。

6.2.1代码编写与分析

创建文件RoundRobin.c

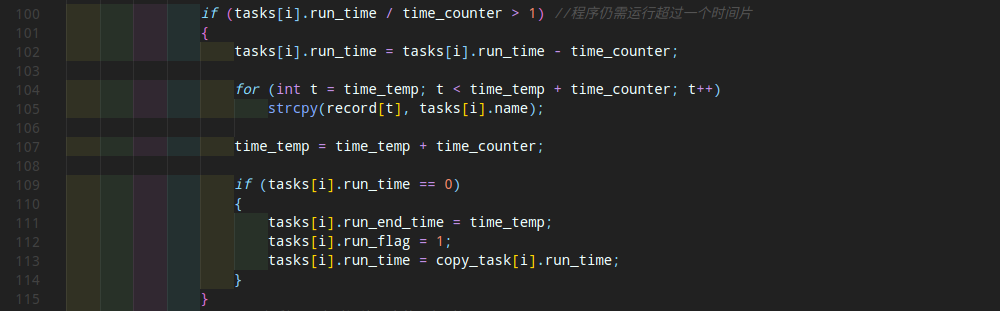


一.轮转法的基本原理：

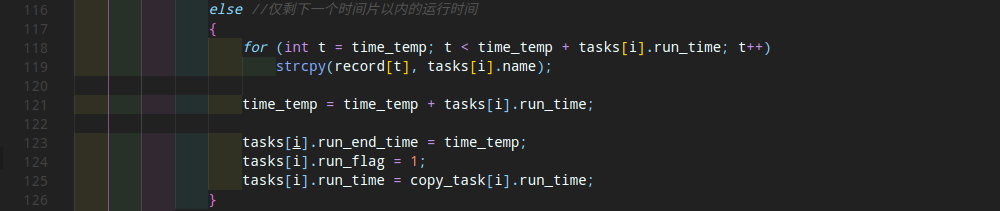
根据先来先服务的原则，将需要执行的所有进程按照到达时间的大小排成一个升序的序列，每次都给一个进程同样大小的时间片，在这个时间片内如果进程执行结束了，那么把进程从进程队列中删去，如果进程没有结束，那么把该进程停止然后改为等待状态，放到进程队列的尾部，直到所有的进程都已执行完毕

二.进程的切换

1. 程序仍需运行超过一个时间片：意思是在该时间片内，进程只能完成它的一部分任务，在时间片用完之后，将进程的状态改为等待状态，将进程放到进程队列的尾部，等待cpu的调用。

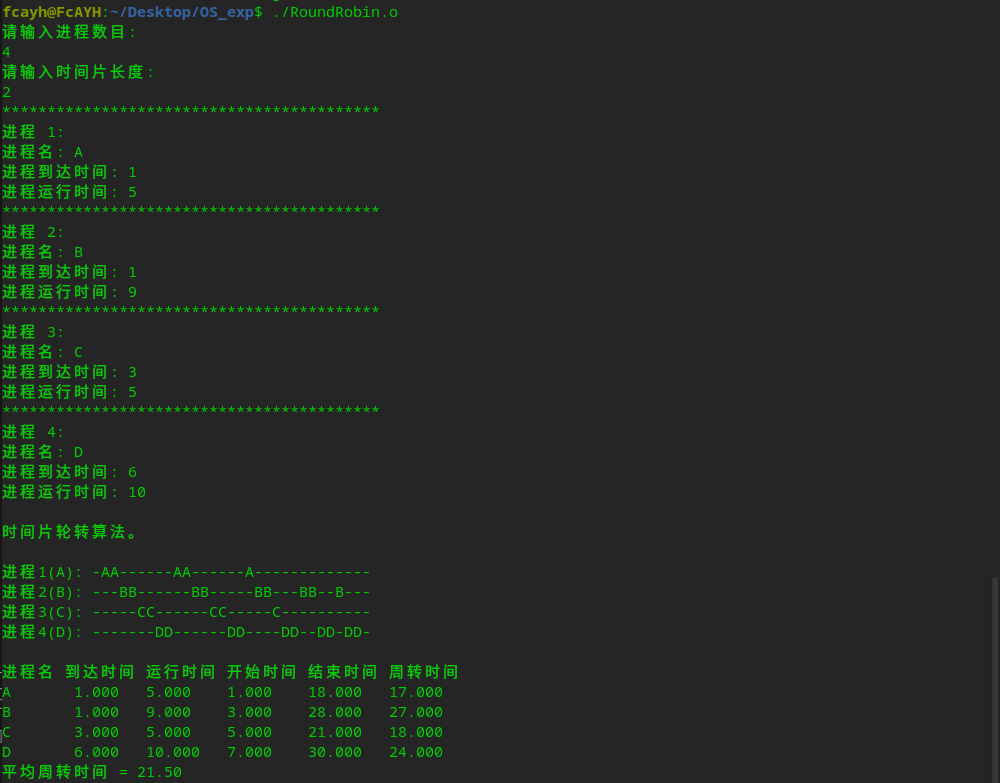


1. 程序在一个时间片内结束：意思就是在该时间片内，进程可以运行至结束，进程运行结束之后，将进程从进程队列中删除，然后启动新的时间片。



6.2.2运行结果与分析

将基本信息(进程数目，时间片长度，各进程详细信息)输入后，程序现实以可视化的形式输出了每一个单位时间运行的进程的名字，并随后输出了每个进程周转运行的详细信息。



6.3实验总结

通过此次实验，完成了编写C语言代码模拟时间片轮转调度算法，加深了对RuondRobin算法的理解。