

**HUBEI UNIVERSITY OF AUTOMOTIVE TECHNOLOGY**



**操作系统原理**

**实 验 报 告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验项目 | 进程控制实验 | | | |
| 学生姓名 | 章崇文 | 学生学号 | 202202296 | |
| 学生班级 | 计算机222 | 完成日期 | 2024.5.1 | |
| 实验成绩 |  | 评阅日期 | |  |
| 评阅教师 |  | | | |

**湖北汽车工业学院实验报告**

班 号 学 号   姓 名

选课班中的序号  完成日期 年 月 日 至 节

## 实验三 进程间通信实验

一、实验目的

1、了解什么是信号。

2、熟悉LINUX系统中进程之间软中断通信的基本原理。

3、理解进程的同步关系。

4、掌握用信号实现进程间的同步操作。

5、了解什么是管道。

6、熟悉UNIX/LINUX支持的管道通信方式。

**二、实验内容**

1、阅读下列程序，执行程序并给出结果截屏，分析程序功能。（2分）

1）#include <unistd.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int k;

void func(int signo)

{

k = 0;

}

int main()

{

int p;

while ((p = fork()) == -1);

if (p == 0)

{

k = 1;

signal(10, func);

while (k != 0)

;

printf("Child process is killed by parent!\n");

}

else

{

sleep(1);

kill(p, 10);

wait(0);

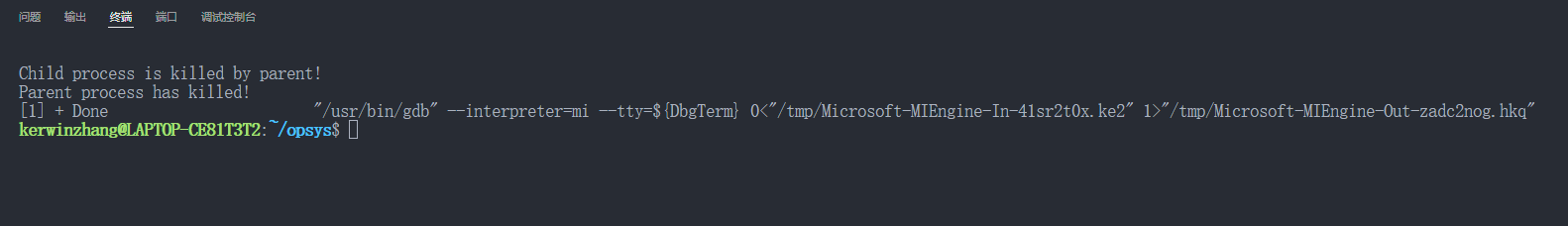
printf("Parent process has killed!\n");

}

return 0;

}

**编译及执行过程和运行结果截屏：**



**分析：**

全局变量 k 用于在父子进程之间传递信号标志。

func() 函数是信号处理函数,用于将 k 设置为 0。

main() 函数中:

首先使用 fork() 创建子进程。

如果是子进程,则将 k 设置为 1,并注册信号处理函数 func()。然后进入一个循环,直到 k 被设置为 0。

如果是父进程,则等待 1 秒钟,然后向子进程发送信号 10 (SIGUSR1)。之后调用 wait() 等待子进程结束。

当子进程收到信号 10 时,会执行信号处理函数 func(),将 k 设置为 0,从而结束子进程的循环。

父进程在等待子进程结束后,会打印出 "Parent process has killed!"。子进程在被信号终止后,会打印出 "Child process is killed by parent!"。

#include <unistd.h>

#include <signal.h>

#include <stdio.h>

int k = 1;

void func(int signo) {

k = 0;

}

int main() {

signal(SIGINT, func);

while (k != 0) {

// do nothing

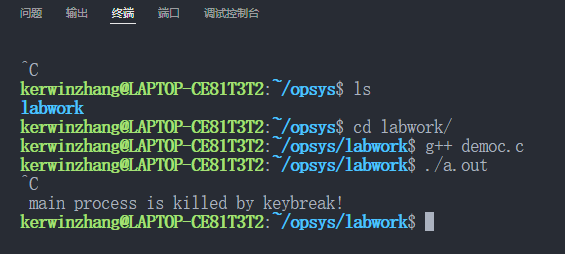
}

printf("\n main process is killed by keybreak!\n");

return 0;

}

**编译及执行过程和运行结果截屏：**



**分析：**

声明了一个全局变量 k，初始值为 1。

定义了一个信号处理函数 func(int signo)，当接收到信号时，将 k 设置为 0。

在 main() 函数中:

注册了 SIGINT 信号的处理函数为 func(int signo)。SIGINT 信号通常是由键盘上的 Ctrl+C 触发的。

进入一个 while 循环，循环条件为 k != 0。

在循环中什么也不做，只是等待信号的到来。

当 k 被设置为 0 时，退出循环，打印 "main process is killed by keybreak!"。

当用户在键盘上按下 Ctrl+C 时，操作系统会向进程发送 SIGINT 信号。

程序收到 SIGINT 信号后，会执行注册的信号处理函数 func(int signo)，将 k 设置为 0。

当 k 被设置为 0 时，while 循环退出，程序打印 "main process is killed by keybreak!"，然后退出。

2、 编写一段程序，使用系统调用fork()创建两个子进程，再用系统调用signal()让父进程捕捉键盘上来的中断信号（即按ctrl+c键），当捕捉到中断信号后，父进程用系统调用kill()向两个子进程发出信号，子进程捕捉到信号后，分别输出下列信息后终止：

Child process 1 is killed by parent!

Child process 2 is killed by parent!

父进程等待两个子进程终止后，输出以下信息后终止：

Parent process is killed!

要求给出编译及执行过程和运行结果截屏。（2分）

#include <iostream>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/wait.h>

void child\_process\_1(int sig) {

std::cout << "Child process 1 is killed by parent!" << std::endl;

exit(0);

}

void child\_process\_2(int sig) {

std::cout << "Child process 2 is killed by parent!" << std::endl;

exit(0);

}

void parent\_process(int sig) {

int status;

kill(0, SIGINT); // 向所有子进程发送信号

waitpid(-1, &status, 0); // 等待第一个子进程终止

waitpid(-1, &status, 0); // 等待第二个子进程终止

std::cout << "Parent process is killed!" << std::endl;

exit(0);

}

int main() {

pid\_t pid1, pid2;

// 创建子进程1

if ((pid1 = fork()) == 0) {

signal(SIGINT, child\_process\_1);

while(1); // 子进程1进入死循环

}

// 创建子进程2

if ((pid2 = fork()) == 0) {

signal(SIGINT, child\_process\_2);

while(1); // 子进程2进入死循环

}

// 父进程捕捉SIGINT信号

signal(SIGINT, parent\_process);

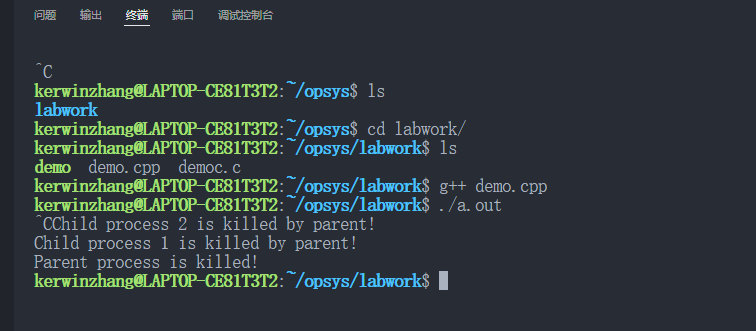
// 父进程等待键盘中断信号

while(1);

return 0;

}

执行过程：



2. 编写程序，利用信号实现司机售票员同步操作问题。要求给出编译及运行过程和结果截（2分）

编程思路：用fork()创建一个子进程代表售票员，司机在父进程中，再用系统调用signal()让父进程（司机）捕捉来自子进程（售票员）发出的信号，让子进程（售票员）捕捉来自（司机）发出的信号，以实现进程间的同步运行。

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sched.h>

int ticketAvailable = 0; // 可用票数

void driverHandler(int signum) {

printf("Driver: Got signal from ticket seller.\n");

ticketAvailable++;

}

void sellerHandler(int signum) {

printf("Ticket Seller: Got signal from driver.\n");

if (ticketAvailable > 0) {

printf("Ticket Seller: Selling a ticket.\n");

ticketAvailable--;

} else {

printf("Ticket Seller: No ticket available.\n");

}

}

int main() {

pid\_t pid = fork();

if (pid < 0) {

fprintf(stderr, "Fork failed.\n");

return 1;

} else if (pid == 0) {

// 子进程，售票员

signal(SIGUSR1, sellerHandler);

while (1) {

printf("Ticket Seller: Waiting for driver's signal.\n");

sleep(1); // 等待司机发出信号

kill(getppid(), SIGUSR1); // 向父进程发送信号

usleep(500000); // 延时0.5秒

}

} else {

// 父进程，司机

signal(SIGUSR1, driverHandler);

while (1) {

printf("Driver: Waiting for ticket seller's signal.\n");

sleep(1); // 等待售票员发出信号

kill(pid, SIGUSR1); // 向子进程发送信号

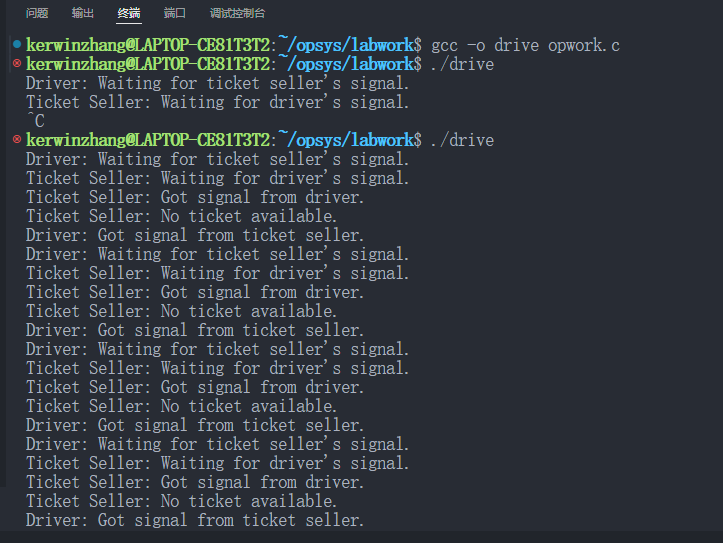
usleep(500000); // 延时0.5秒

}

}

return 0;

}



4、编制一段程序，实现进程的管道通信。使用pipe()建立一条管道线。两个子进程p1和p2分别向管道各写一句话：

Child 1 is sending message!

Child 2 is sending message!

而父进程则从管道中读出来自于两个子进程的信息，显示在屏幕上。（1分）

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

int pipefd[2];

pid\_t p1, p2;

if (pipe(pipefd) == -1) {

fprintf(stderr, "Pipe creation failed.\n");

return 1;

}

p1 = fork();

if (p1 < 0) {

fprintf(stderr, "Fork failed.\n");

return 1;

} else if (p1 == 0) {

// 子进程1，向管道写入消息

close(pipefd[0]); // 关闭读取端

char\* message = "Child 1 is sending message!";

write(pipefd[1], message, strlen(message) + 1); // 写入消息到管道

close(pipefd[1]); // 关闭写入端

exit(0);

}

p2 = fork();

if (p2 < 0) {

fprintf(stderr, "Fork failed.\n");

return 1;

} else if (p2 == 0) {

// 子进程2，向管道写入消息

close(pipefd[0]); // 关闭读取端

char\* message = "Child 2 is sending message!";

write(pipefd[1], message, strlen(message) + 1); // 写入消息到管道

close(pipefd[1]); // 关闭写入端

exit(0);

}

// 父进程，从管道读取消息

close(pipefd[1]); // 关闭写入端

char buffer[256];

int bytesRead;

while ((bytesRead = read(pipefd[0], buffer, sizeof(buffer))) > 0) {

printf("Parent: Received message: %s\n", buffer);

}

close(pipefd[0]); // 关闭读取端

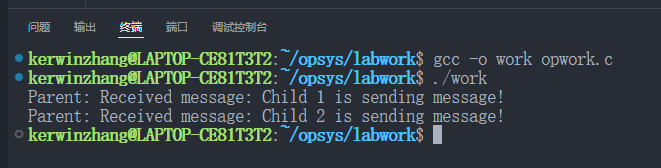
wait(NULL); // 等待子进程结束

wait(NULL);

return 0;

}

编译及执行过程和运行结果截屏。



5、在父进程中用pipe()建立一条管道线，往管道里写信息，两个子进程接收父进程发送的信息。（2分）

#include <unistd.h>

#include <iostream>

#include <cstring>

#include <sys/wait.h>

int main()

{

int fd[2];

pid\_t pid1, pid2;

char message1[] = "Message for Child 1";

char message2[] = "Message for Child 2";

// 创建管道

if (pipe(fd) == -1)

{

perror("pipe");

exit(EXIT\_FAILURE);

}

// 创建第一个子进程

pid1 = fork();

if (pid1 == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid1 == 0)

{ // 子进程1

close(fd[1]); // 关闭写端

// 读取父进程发送的信息

char buffer[100];

ssize\_t bytesRead = read(fd[0], buffer, sizeof(buffer) - 1);

if (bytesRead == -1)

{

perror("read");

exit(EXIT\_FAILURE);

}

buffer[bytesRead] = '\0'; // 确保字符串正确终止

std::cout << "Child 1 received: " << buffer << std::endl;

close(fd[0]); // 关闭读端

exit(EXIT\_SUCCESS);

}

else

{

// 创建第二个子进程

pid2 = fork();

if (pid2 == -1)

{

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid2 == 0)

{ // 子进程2

close(fd[1]); // 关闭写端

// 读取父进程发送的信息

char buffer[100];

ssize\_t bytesRead = read(fd[0], buffer, sizeof(buffer) - 1);

if (bytesRead == -1)

{

perror("read");

exit(EXIT\_FAILURE);

}

buffer[bytesRead] = '\0'; // 确保字符串正确终止

std::cout << "Child 2 received: " << buffer << std::endl;

close(fd[0]); // 关闭读端

exit(EXIT\_SUCCESS);

}

else

{ // 父进程

close(fd[0]); // 关闭读端

// 向子进程发送信息

if (write(fd[1], message1, strlen(message1)) == -1)

{

perror("write to child 1");

}

// 等待子进程1接收完毕

waitpid(pid1, NULL, 0);

// 向子进程发送信息

if (write(fd[1], message2, strlen(message2)) == -1)

{

perror("write to child 2");

}

// 等待子进程2接收完毕

waitpid(pid2, NULL, 0);

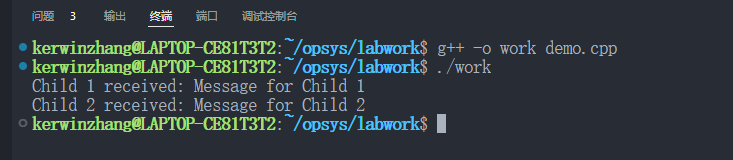
close(fd[1]); // 关闭写端

exit(EXIT\_SUCCESS);

}

}

}



6、编程用管道实现父子进程间的双向通信。给出设计思路和结果截屏。（附加题）

#include <iostream>

#include <unistd.h>

#include <cstring>

#include <sys/wait.h>

int main() {

int pipe1[2], pipe2[2];

pid\_t pid;

char buf[1024];

ssize\_t bytesRead;

// 创建两个管道

if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

// 创建子进程

pid = fork();

if (pid == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) { // 子进程

// 关闭不需要的管道端

close(pipe1[1]);

close(pipe2[0]);

// 从父进程接收数据

bytesRead = read(pipe1[0], buf, sizeof(buf) - 1);

if (bytesRead > 0) {

buf[bytesRead] = '\0';

std::cout << "Child received: " << buf << std::endl;

// 向父进程发送数据

const char \*reply = "Message from child";

write(pipe2[1], reply, strlen(reply));

}

// 关闭管道并退出

close(pipe1[0]);

close(pipe2[1]);

exit(EXIT\_SUCCESS);

} else { // 父进程

// 关闭不需要的管道端

close(pipe1[0]);

close(pipe2[1]);

// 向子进程发送数据

const char \*message = "Message from parent";

write(pipe1[1], message, strlen(message));

// 从子进程接收数据

bytesRead = read(pipe2[0], buf, sizeof(buf) - 1);

if (bytesRead > 0) {

buf[bytesRead] = '\0';

std::cout << "Parent received: " << buf << std::endl;

}

// 等待子进程并清理资源

waitpid(pid, NULL, 0);

// 关闭管道

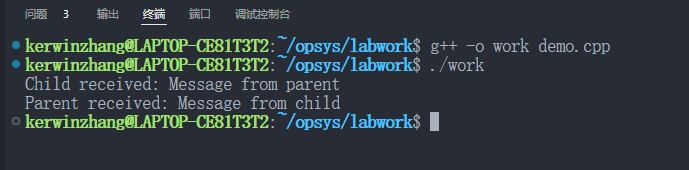
close(pipe1[1]);

close(pipe2[0]);

exit(EXIT\_SUCCESS);

}

}



三、实验总结和体会

本次实验聚焦于信号与管道机制，旨在通过实际操作加深对这些进程间通信与同步技术的理解。通过编写代码，我深入探索了信号作为异步通信工具在Linux系统中的运作原理，并掌握了使用信号实现进程同步的技巧。同时，我也学习了管道的概念及其在UNIX/Linux系统中支持的通信方式。

对于信号，我更加明晰了其作为进程间异步通信手段的重要性。信号能够告知进程关于特定事件（如键盘输入、硬件故障等）的发生，并通过signal函数注册相应的处理函数以作出响应。

在管道的学习过程中，我认识到其作为一种进程间通信手段，可以实现一个进程的输出直接作为另一个进程的输入。通过pipe函数创建管道，我实现了父子进程间的双向通信，加深了对管道工作原理的理解。

实验过程中，我遇到了信号与管道使用上的挑战。例如，在理解信号处理机制时，需要不断实践以加深理解；而在使用管道进行通信时，必须严格把控管道的创建、关闭顺序以及数据的读写操作，以避免潜在的阻塞或数据丢失问题。

通过本次实验，我不仅对进程间通信与同步的概念有了更深刻的认识，还掌握了信号与管道这两种关键技术在多进程环境中的应用。我认识到在多进程系统中，进程间的同步操作对于避免竞态条件和数据不一致至关重要。信号作为一种进程同步机制，通过信号的发送与接收，可以有效地协调与同步进程间的活动。而管道作为一种通信方式，为父子进程间的数据共享与传输提供了便捷的途径。

本次实验不仅提升了我对信号与管道技术的理解，还锻炼了我解决实际问题的能力。我相信这些宝贵的经验将对我未来的系统编程和多进程应用程序开发产生积极影响。