计算机学院_操作系统_课程实验报告

实验题目： 实验 1 进程控制		学号：202300130183
日期：2025/3/3	班级： 23 级智能班	姓名：宋浩宇
Email：202300130183@mail.sdu.edu.cn		

实验方法介绍：
使用 Oracle Virtual Box 运行 Ubuntu24.04 虚拟环境来编写编译相应的代码。

实验过程描述：
关于示例实验，我们先将实验指导书提供的代码复刻一遍。
以下为复刻代码（使用 VSCode 渲染）：
pctl.c 文件：

```c
#include "pctl.h"

int main(int argc, char *argv[])
{
    int i;
    int pid;
    int status;
    char* args[] = { "/bin/ls","-a",NULL };
    signal(SIGINT, ((__sighandler_t)sigcat));
    perror("SIGINT");
    pid = fork();
    if (pid < 0)
    {
        printf("Create Process fail!\n");
        exit(EXIT_FAILURE);
    }
    if (pid == 0)
    {
        printf("I am Child process %d\n My father is %d\n", getpid(),
getppid());
        pause();
        printf("%d child will Running:\n", getpid());
        if (argv[1] != NULL)
        {
            for (i = 0;argv[i] != NULL;i++)
            {
                printf("%s ", argv[i]);
                printf("\n");
```

```
            }
            status = execve(argv[1],&argv[1],NULL);
        }
        else
        {
            for (i = 0;args[i] != NULL;i++)
            {
                printf("%s ", args[i]);
                printf("\n");
            }
            status = execve(args[0], args, NULL);
        }
    }
    else
    {
        printf("\nI am Parent process %d\n", getpid());
        if (argv[1] != NULL)
        {
            printf("%d Waiting for child done.\n\n", getpid());
            waitpid(pid, &status, 0);
            printf("\nMy child exit! status=%d\n\n", status);
        }
        else
        {
            sleep(5);
            if (kill(pid, SIGINT) >= 0)
            {
                printf("%d Wakeup %d child.\n", getpid(), pid);
            }
            printf("%d don`t Wait for child done.\n\n", getpid());
        }
    }
    return EXIT_SUCCESS;
}
```

pctl.h 文件：

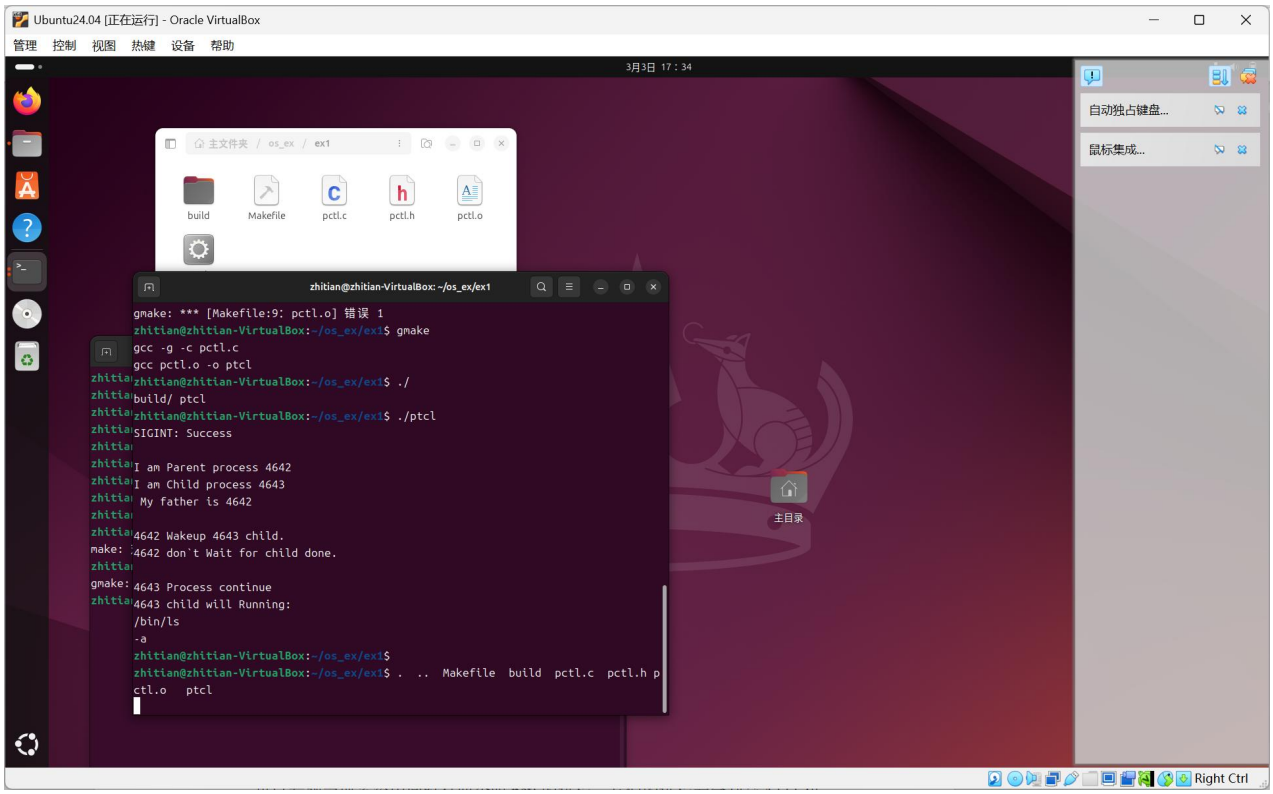
```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
void sigcat()
{
    printf("%d Process continue\n", getpid());
```

```
}
```

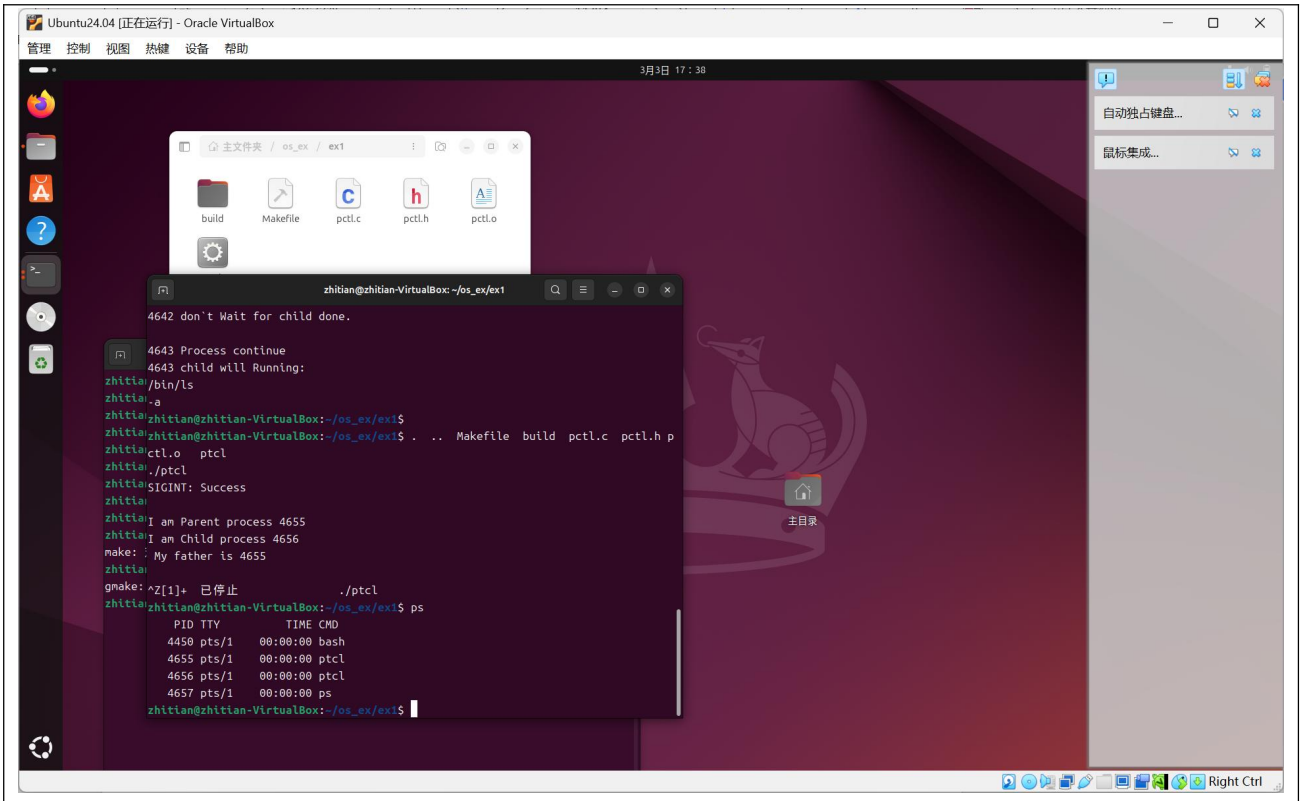
创建 Makefile，内容为：

```
srcs = sfex.c
objs = sfex.o
opts = -g -c
all:pctl
pctl:$(objs)
    gcc $(objs) -o sfex
pctl.o:$(srcs)
    gcc $(opts) $(srcs)
clean:
    rm sfex *.o
```

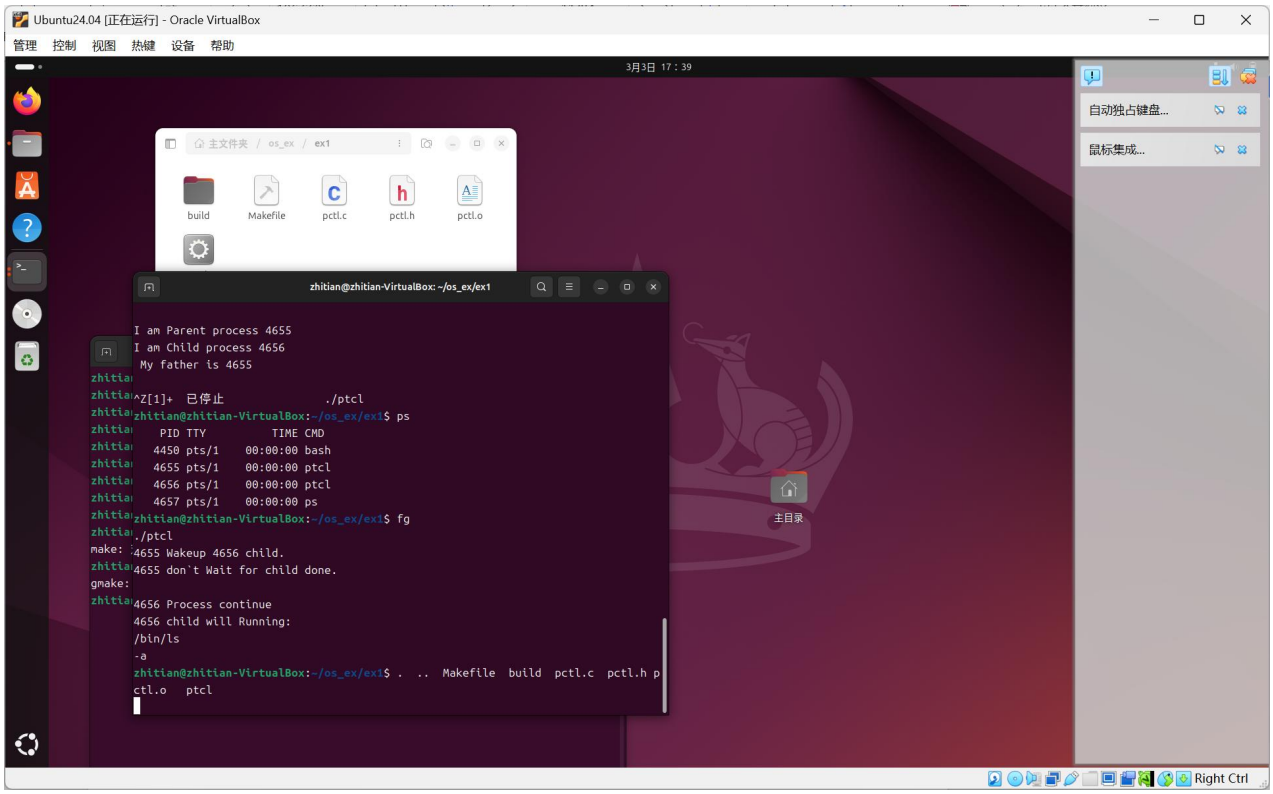
编译并运行程序，运算结果如下：



键入 Ctrl+Z 暂停进程，并查看当前工作目录运行的所有进程，结果如下：



让挂起的进程继续执行：



以上为示例实验的结果。

关于独立实验，我们首先需要写出实现题目要求功能的代码：

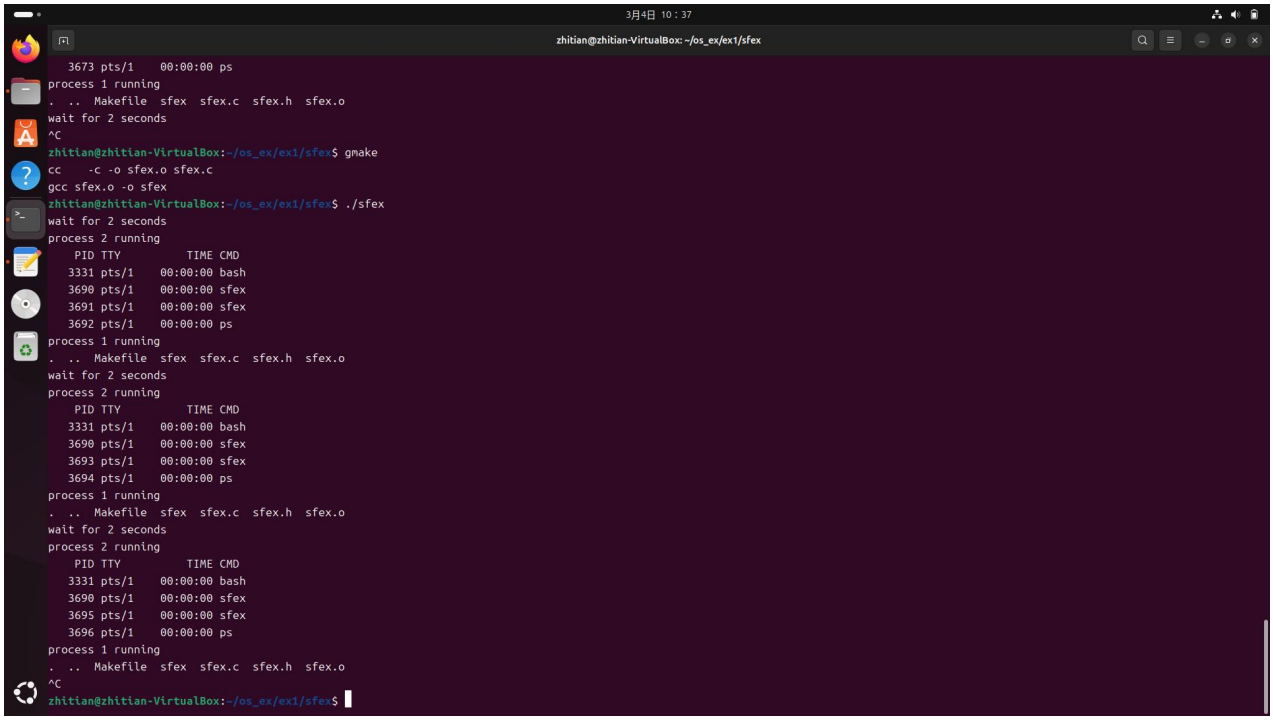
```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
void sigusr1_handler(int sig)
{
    // 空处理函数，仅用于唤醒 pause()
}
int main(int argc, char* argv[])
{
    char* args1[] = { "/bin/ls","-a",NULL };
    char* args2[] = { "/bin/ps",NULL,NULL };
    int pid1, pid2;
    int status1, status2;

    struct sigaction sa;
    sa.sa_handler = sigusr1_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGUSR1, &sa, NULL);


    while (1)
    {
        printf("wait for 2 seconds\n");
        sleep(2);
        pid1 = fork();
        if (pid1 == 0)
        {
            pause();
            printf("process 1 running \n");
            status1 = execve(args1[0], args1, NULL);
            fflush(stdout);
            exit(status1);
        }
        if (pid1 > 0)
        {
            pid2 = fork();
            if (pid2 == 0)
            {
                // printf("wait_for %d\n",getppid());
                // fflush(stdout);
                // pause();
                printf("process 2 running \n");
                status2 = execve(args2[0], args2, NULL);
```

```
                fflush(stdout);
                exit(status2);
            }
            else
            {
                // sleep(1);
                // kill(pid2, SIGUSR1);
                // printf("send_signal %d\n",pid2);
                // fflush(stdout);
                waitpid(pid2, &status2, 0);
                kill(pid1, SIGUSR1);
                waitpid(pid1, &status1, 0);
            }
        }
    }
}
```

简单讲解一下，首先父进程会等待两秒，这是为了让这个循环的执行速度慢一些，方便我们观察结果。建立子进程一，进程一会立刻被挂起，然后父进程会建立子进程二，并进入等待子进程二执行结束的状态，子进程二会直接执行 ps 命令，然后子进程二结束，父进程继续执行，父进程发出信号触发子进程里的空函数，使进程继续执行，父进程发送完信号后进入等待子进程一执行结束的状态，子进程一被激活继续执行，执行 ls 命令后结束，父进程继续循环执行。

执行结果如下：



结论分析：

1. 根据实验中观察和记录的信息结合示例实验和独立实验程序，说明它们反映出操作系统教材中进程及处理机管理一节讲解的进程的哪些特征和功能?

答：体现了进程的动态性、并发性、独立性、异步性特征；资源分配和管理、进程控制、进程调度、进程通信、上下文切换的功能。

2. 在真实的操作系统中它是怎样实现和反映出教材中讲解的进程的生命期、进程的实体和进程状态控制的。

答：关于进程的生命期问题，在 linux 系统中，进程的创建是通过 fork()函数实现的，进程的终止是通过调用 exit()函数或者 main()函数 return 来触发，又或者使通过信号触发，比如 SIGINT、SIGSEGV 等，更进一步地查询资料还可以知道，linux 系统中还有僵尸进程，表示的是完成执行之后资源还保存着的进程，通过父进程回收资源，或者祖先进程 init 进程回收。进程还有进程树的概念，可以通过进程树来查看进程的父子关系。而关于进程的实体，在 linux 系统中被定义为 task_struct，其中包括进程的标识信息、调度状态、占用在内存中的位置的指针、文件的读写状态、信号的处理函数等等。在 linux 系统的进程的状态控制中，进程有运行、阻塞、就绪、终止、僵尸这几个状态，当 IO 完成后，进程进入就绪状态，加入运行的队列，在完成时间片的运行之后，会在被转化成就绪态，再进入运行队列，并通过调度器来根据一定的策略调度。

3. 你对于进程概念和并发概念有哪些新的理解和认识?

答：了解到了进程是计算机中以操作系统为载体运行程序的最小单位，即实体化的程序，并且除了程序的代码中程序员所编写的内容以外,其中还包含了大量的操作系统做的虚拟内存映射、内存分段、进程状态储存等工作。而关于并发，对于一个只有一个核心的 CPU，可以理解并发并不是同时执行，而是 CPU 交替着为每一个进程服务，使得对于人的感知来说就像是同时发生的一样。

4. 子进程是如何创建和执行新程序的?

答：子进程是通过 fork()函数建立的，对于操作系统来说是赋值了一份进程的实体，执行新程序是通过 exec()一类的函数通过替换原进程的上下文实现的。

5. 信号的机理是什么?怎样利用信号实现进程控制?

答：信号本质上是一种内核直接管理的软中断，由内核或进程发送给目标进程，触发其预定义行为，实现了一种异步通信的功能。而想要利用信号实现进程控制，可以使用一些系统预定义好的信号，比如 SIGINT，也可以自定义一个信号处理函数，并调用 signal()函数注册处理函数。

结论：
进程实现了操作系统对于每一个程序实体的动态创建、终止、资源分配和回收、地址空间隔离，以及多个进程之间的通信。操作系统通过调度器动态地为每一个进程分配时间片，以保证对人来说的进程的并行执行。并且操作系统暴露给程序员等其他使用者一些相应的接口，让使用者可以充分利用操作系统的进程的特性。进程通过动态性、独立性、并发性等特征，配合资源分配、调度和通信等功能，使操作系统能够高效管理复杂任务，是现代计算系统实现多任务和资源共享的核心机制。