四川大学计算机学院、软件学院

实验报告

学号：2022141460155 姓名：林诺晗 专业：计算机金融 班级：计金班 第8周

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 课程名称 | 操作系统实验 | | 实验课时 | 2 |
| 实验项目 | 进程的创建 | | 实验时间 | 2024-4-18 |
| 实验目的 | （1）加深对进程概念的理解，进一步认识并发执行的实质。  （2）掌握Linux 操作系统中进程的创建和终止操作。  （3）掌握在Linux 操作系统中创建子进程并加载新映像的操作。 | | | |
| 实验环境 | VMware Workstation 17 Pro  Ubuntu 22.04.3 LTS | | | |
| 实验内容（算法、程序、步骤和方法） | **一．编写一个C 程序，并使用系统调用fork()创建一个子进程。**  ①在子进程中分别输出当前进程为子进程的提示、当前进程的PID 和父进程的PID、根据用户输入确定当前进程的返回值、退出提示等信息。  ②在父进程中分别输出当前进程为父进程的提示、当前进程的PID 和子进程的PID、等待子进程退出后获得的返回值、退出提示等信息。  **相关原理：Linux进程**  （1）交互进程：由一个shell启动的进程。交互进程既可在前台运行，也可以在后台运行，前者称为前台进程，后者称为后台进程。  （2）批处理进程：这种进程和终端没有联系，是一个进程系列，由多个进程按照指定的方式执行。  （3）守护进程（Daemon）：运行在后台的一种特殊进程，它在系统启动时启动，并在后台运行。  **Linux进程状态**  运行态（TASK\_RUNNING）：进程准备运行，或正在运行  可中断等待态（TASK\_INTERRUPTIBLE）：进程等待特定事件  可中断等待态（TASK\_UNINTERRUPTIBLE）：进程处于等待状态，但是此刻进程是不可中断的  僵尸态（TASK\_ZOMBIE）：进程已经停止运行，但在内存仍有结构(task\_struct)  停止态（TASK\_STOPPED/ TASK\_TRACED ）：进程暂停状态  **进程标识符管理**  int getpid(); //取得当前进程的标识符（进程ID）。  int getppid(); //取得当前进程的父进程ID。  int getpgrp(); //取得当前进程的进程组标识符。  int getpgid(int pid); //将当前进程的进程组标识符改为当前进程的进程ID，使其成为进程组首进程，并返回这一新的进程组标识符。  **源代码：**  #include <unistd.h>  #include <sys/types.h>  #include <errno.h>  #include <sys/wait.h>  #include <stdlib.h>  #include <stdio.h>  int main()  {  pid\_t childpid;  int retval;  int status;  childpid=fork();  if(childpid>=0)  {  if(childpid==0)  {  printf("CHILD: I am the child process! \n");  printf("CHILD: Here's my PID: %d\n", getpid());//获取当前PID  printf("CHILD: My parent's PID is: %d\n", getppid());//获取父进程PID  printf("CHILD: The value of fork return is: %d\n", childpid);  printf("CHILD: Sleep for 1 second...\n");  sleep(1);  printf("CHILD: Enter an exit value (0~255):");  scanf("%d",&retval);  printf("CHILD: Google! \n");  exit(retval);  }  else  {  printf("Parent, I am the parent process! \n");  printf("Parent: Here's my PID: %d\n", getpid()); //获取当前PID  printf("Parent: The value of my child's PID is: %d\n", childpid);  printf("Parent: I will now wait for my child to exit.\n");  wait(&status); //等待并收回子进程  printf("Parent: Child's exit code is: %d\n", WEXITSTATUS(status));  printf("Parent: Google! \n");  exit(0);  }  }  else  {  perror("fork error!");  exit(0);  }  }  IMG_256  ****编译程序****：使用编译器（如gcc）将.c文件编译成可执行文件。  ****运行程序****：编译成功后，会生成一个名为parson的可执行文件，程序运行如下：  IMG_256   1. **编写另一个C 程序，使用系统调用fork()以创建一个子进程，并使用这个子进程调用exec 函数族以执行系统命令ls。**   **相关原理：**  **进程创建——fork()**  fork ()函数通过系统调用创建一个与原来进程几乎完全相同的进程  两个进程可以做完全相同的事，  根据初始参数或者传入的变量不同，两个进程也可以做不同的事。  一个进程调用fork()函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。  fork()的一个奇妙之处就是它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：  1）（>0）在父进程中，fork返回新创建子进程的进程ID；  2）（=0）在子进程中，fork返回0；  3）（<0）如果出现错误，fork返回一个负值；  fork ()出错可能有两种原因  当前的进程数已经达到了系统规定的上限，这时errno的值被设置为EAGAIN。  系统内存不足，这时errno的值被设置为ENOMEM。  **加载新的进程映像——exec函数族**  创建的进程往往希望它能执行新的程序，在Linux中，进程创建和加载新进程映像是分离操作的。  在Linux中，当创建一个进程后，通常使用exec系列函数将子进程替换成新的进程映像。  exec函数族的作用是根据指定的文件名找到可执行文件，并用它来取代调用进程的内容，换句话说，就是在调用进程内部执行一个可执行文件。  这里的可执行文件既可以是二进制文件，也可以是任何Linux下可执行的脚本文件。  与一般情况不同，exec函数族的函数执行成功后不会返回，因为调用进程的实体，包括代码段，数据段和堆栈等都已经被新的内容取代，只留下进程ID等一些表面上的信息仍保持原样。  **wait()/waitpid()函数**  作用：父进程查询子进程状态  #include <sys/types.h>  #include <sys/wait.h> pid\_t wait(int \*status);  pid\_t waitpid(pid\_t pid, int \*status, int options);  进程一旦调用了wait()，就立即阻塞自己，由wait()自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个已经变成僵尸态的子进程，wait ()就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait ()就会一直阻塞在这里，直到有一个出现为止。  wait()要与fork()配套出现，如果在使用fork()之前调用wait()，wait()的返回值则为-1，正常情况下wait()的返回值为子进程的PID  当父进程没有使用wait()函数等待已终止的子进程时,子进程就会进入一种无父进程清理自己尸体的状态，此时的子进程就是僵尸进程，不能在内核中清理尸体的情况  **源代码：**  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  int main() {  pid\_t pid = fork(); // 创建子进程  if (pid == -1) {  // fork失败  perror("fork");  exit(EXIT\_FAILURE);  } else if (pid == 0) {  // 子进程  const char \*args[] = {"ls", "-l", (char \*)NULL};  if (execvp(args[0], (char \*const \*)args) == -1) {  perror("execvp");  exit(EXIT\_FAILURE);  }  } else {  // 父进程  int status;  pid\_t child\_pid = waitpid(pid, &status, 0); // 等待子进程结束  if (child\_pid == -1) {  perror("waitpid");  exit(EXIT\_FAILURE);  }  if (WIFEXITED(status)) {  printf("子进程退出，退出码：%d\n", WEXITSTATUS(status));  }  }  return 0;  }  IMG_256  在这个程序中，fork()用于创建一个子进程。execvp()用于替换子进程中的进程映像为ls -l命令。它接受命令名称和参数数组。waitpid()用于让父进程等待子进程结束。它接受子进程的PID、一个指向int的指针（用于接收子进程的状态）、以及一些选项。  ****编译程序****：使用编译器（如gcc）将.c文件编译成可执行文件。  ****运行程序****：编译成功后，会生成一个可执行文件，程序运行如下：  IMG_256  该子进程将执行ls -l命令，列出当前目录下的文件和目录，以长格式显示。父进程将等待子进程完成，然后退出。 | | | |
| 结论  （结果） | 实验成功，熟练掌握相关命令的使用。编写出了一个C 程序，并使用系统调用fork()创建一个子进程编写另一个C 程序，使用系统调用fork()以创建一个子进程，并使用这个子进程调用exec 函数族以执行系统命令ls。 | | | |
| 小结 | （1）加深对进程概念的理解，进一步认识并发执行的实质。  （2）掌握Linux 操作系统中进程的创建和终止操作。  （3）掌握在Linux 操作系统中创建子进程并加载新映像的操作。 | | | |
| 指导老师  评议 |  | | | |
| 成绩评定： | 指导教师签名： | | |