2.2 进程操作和进程通信

一、进程操作

计算机系统启动时没有进程。操作系统创建第1个进程，是所有进程的祖先。祖先进程创建新的进程，就是它的子进程。父进程创建子进程，如此重复，形成了进程组织的树型结构。

系统中的进程能够并发执行，它们必须要动态的创建和终止（撤销），操作系统必须要提供进程创建和终止的机制。属于进程控制方面的操作主要用于对进程的创建、进程的撤消和进程状态间转换控制。

（1）进程创建

进程在运行期间通过创建进程系统调用可以创建多个新进程，新进程是它的子进程，每个新进程可以再创建其他进程，从而形成了进程树。创建一个进程主要是为新进程创建一个进程控制块（PCB）。创建进程系统调用首先从系统的 PCB 表中索取一个空白的 PCB 表目，并获得其内部标识，然后将调用进程提供的参数，如外部名、正文段、数据段的首址、大小、所需资源、优先级等填入这张空白 PCB 表目中。并设置新进程状态为就绪态，并把该 PCB 插入到就绪队列中，就可进入系统并发执行。

当进程创建新进程时，有两种执行可能:

①父进程与子进程并发执行。

②父进程等待，直到某个或全部子进程执行完。

新进程的地址空间也有两种可能:

①子进程是父进程的复制品(具有与父进程相同的程序和数据)。

②子进程装入另一 4 个新程序。

为了说明这些不同实现，现在来看一下 UNIX 类操作系统。在 UNIX 中，每个进程都用一个唯一的整数形式的进程标识符来标识。通过 fork()系统调用，可创建新进程。新进程通过复制原来进程的地址空间而成。这种机制允许父进程与子进程方便地进行通信。两个进程(父进程和子进程)都继续执行位于系统调用 fork()之后的指令。但是，有一点不同:对于新(子)进程，系统调用 fork()的返回值为 0; 而对于父进程，返回值为子进程的进程标识符(非零)。

通常，在系统调用 fork()之后，一个进程会使用系统调用 exec()，以用新程序来取代进程的内存空间。系统调用 exec()将二进制文件装入内存，并开始执行。采用这种方式，两个进程能相互通信，并能按各自的方法执行。父进程能创建更多的子进程，或者如果在子进程运行时没有什么可做，那么它采用系统调用 wait()把自己移出就绪队列来等待子进程的终止。

下面所示的 C 程序说明了上述 UNIX 系统调用。现在有两个不同的进程运行同一程序。子进程的 pid 值为 0 ，而父进程的 pid 值大于 0。子进程通过系统调用 execlp()(execlp()是系统调用 exec()的一种版本)，用 UNIX 命令/bin/ls (用来列出文件和目录清单)来覆盖其地址空间。父进程通过系统调用 wait()来等待子进程的完成。当子进程完成时(通过显式或隐式调用 exit()) ，父进程会从 wait()调用的下一指令继续执行，并调用系统调用 exit()以表示结束，如图 2.5。

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main(){
    pid_t pid;
```

```
        /* fork a child process */
        pid =fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                return 1;
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls","ls",NULL);
        }
        else { /* parent process */
        }
        /* parent will wait for the child to complete */
        wait (NULL) ;
        printf("Child Complete");
        return 0;
}
```

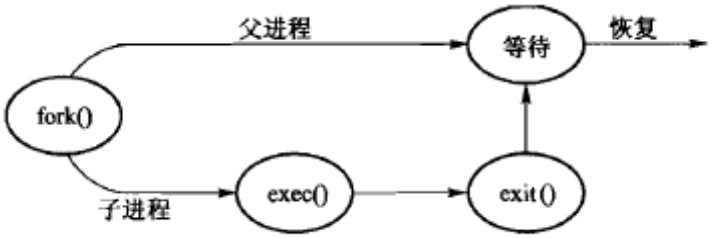

图 2.5 fork 进程创建

再如，考虑在 Windows 中的进程创建。Win32 API 通过采用 CreateProcess()函数(它与 fork()中的父进程生成子进程类似)创建进程。然而，fork()中子进程继承了父进程的地址空间，而 CreateProcess()生成函数时，需要将一个特殊程序装入子进程的地址空间。进一步说，与 fork()不需要传递参数不同，CreateProcess()至少需要传递 10 个参数。

下面所示的 C 程序是一个 CreateProcess()函数，它生成一个装载 mspaint.exe 应用程序的子进程。选择 10 个参数中的默认值传递给 CreateProcess()函数。需要了解 Win32 API
中进程创建和管理细节，可以查阅 Microsoft MSDN 有关文档。

传递给 CreateProcess()的两个参数是 STARTUPINFO 和 PROCESS_INFORMATION 结构的实例。STARTUPINFO 指明新进程的许多特性，如窗口大小、标准输入及输出文件的句柄。PROCESS_INFORMATION 结构包含一个句柄以及新的生成进程和线程的标识。在调用 CeateProcess()之前，调用 ZeroMemoryO 函数来为其中每个结构清空内存。

首先传递给 CeateProcess()函数的两个参数是应用名和命令行参数。如果应用名为 NULL(此时它就是 NULL)，命令行参数指明了要装入的应用。在这个例子中，装入的是微软 Windows 中的 mspaint.exe 应用程序。除这两个初始参数之外，使用系统默认参数来继承进程和线程句柄和指定不创建标志，还使用父进程的已有环境块和启动目录。最后，提供了两个指向程序刚开始生成的 STARTUPINFO 和 PROCESS_INFORMATION 结构的指针。在 Win32 中用 WaitForSingleObject()等待子进程结束，它的参数指定一个子进程的句柄

(pi.hP rocess) 即等待进程结束。一旦子进程结束，控制将从 WaitForSingleObject()函数返回到父进程。

```
#include <windows.h>
#include <stdio.h>
void main()
{
   STARTUPINFO si;
   PROCESS_INFORMATION pi;
   ZeroMemory( &si, sizeof(si) );
   si.cb = sizeof(si);
   ZeroMemory( &pi, sizeof(pi) );
   // Start the child process.
   if( !CreateProcess( NULL,   // No module name (use command line).
      "E:\\WINDOWS\\system32\\mspaint.exe", // Command line.
      NULL,            // Process handle not inheritable.
      NULL,            // Thread handle not inheritable.
      FALSE,           // Set handle inheritance to FALSE.
      0,              // No creation flags.
      NULL,            // Use parent's environment block.
      NULL,            // Use parent's starting directory.
      &si,             // Pointer to STARTUPINFO structure.
      &pi )            // Pointer to PROCESS_INFORMATION structure.
   )
   {
      printf( "CreateProcess failed (%d).\n", GetLastError() );
      return -1;
   }
   // Wait until child process exits.
   WaitForSingleObject( pi.hProcess, INFINITE );
   // Close process and thread handles.
   CloseHandle( pi.hProcess );
   CloseHandle( pi.hThread );
}
```

（2）进程终止

对于树型层次结构的进程系统终止系统调用采用的策略是由父进程发出，终止它的一个子进程及该子进程所有的子孙进程，被终止进程的所有资源（主存、外存、PCB 表目）全部释放出来归还系统，并将它们从所有的队列中移去。如终止的进程正在运行，则要调用进程调度程序将处理器分给其它进程。

当进程完成执行最后的语句并使用系统调用 exit()请求操作系统删除自身时，进程终止。这时，进程可以返回状态值(通常为整数)到父进程(通过系统调用 wait())。

进程通过适当的系统调用(如 Win32 中的 TenninatePorcess())能终止另一个进程。通常，只有被终止进程的父进程才能执行这一系统调用。否则，用户可以任意地终止彼此的作业。记住，父进程需要知道其子进程的标识符。因此，当一个进程创建新进程时，新创建进

程的标识符要传递给父进程。

父进程终止其子进程的原因有很多，如:

- 子进程使用了超过它所分配到的一些资源。(为判定是否发生这种情况，要求父进程有一个检查其子进程状态的机制。)
- 分配给子进程的任务已不再需要。
- 父进程退出，如果父进程终止，那么操作系统不允许子进程继续。

（3）进程的阻塞

当前进程因请求某事件而不能执行时（例如请求 I/O 而等待 I/O 完成时），该进程将调用阻塞系统调用阻塞自己，暂时放弃处理机。进程阻塞是进程自身的主动行为。阻塞过程首先立即停止原来程序的执行，把 PCB 中的现行状态由运行态改为阻塞态，并将 PCB 插入到等待某事件的阻塞队列中，最后调用进程调度程序进行处理机的重新分配。

（4）进程的唤醒

当被阻塞的进程所期待的事件发生时（例如 I/O 完成时），则有关进程（例如 I/O 设备处理程序或释放资源的进程等）调用唤醒系统调用，将阻塞的进程唤醒，将等待该事件的进程从阻塞队列移出，插入到就绪队列中，将该进程的 PCB 中现行状态改为就绪状态。

二、进程间通信

进程间通信(Inter-Process Communication, IPC)要解决的问题是进程间的信息交换这种信息交换的量可大可小。操作系统提供了多种进程间通信机制，可分别适用于多种不同的场合，要特别关注考试大纲的三种进程间通信：共享存储机制、消息传递机制、管道通信机制。

（1）共享存储系统

在这种通信方式中，在内存中划出一块存储区，供多个进程共享，共享进程通过对这一共享存储区中的数据进行读或写来实现通信。共享存储区是进程通信速度最快的一种通信机制，且可以实现大量数据传送。

（2）消息传递系统

在消息传递系统中，进程间的数据交换以消息（message）为单位，在计算机网络中，消息又称为报文。消息传递系统因其实现方式不同，又可分为直接通信方式和间接通信方式两种。

消息传递机制至少需要提供两条原语 send 和 receive，send 向一个给定的目标发送一个消息，receive 则从一个给定的源接受一条消息。

在直接通信方式下，发送或接收消息的每个进程必须指出消息发给谁或从谁那里接收消息，用 send 原语和 receive 原语实现进程之间的通信，这两个原语定义如下:

send（P，消息）：把消息发送给进程 P

receive（Q，消息）：从进程 Q 接收消息

进程 P 和 Q 通过执行这两个操作自动建立了一种连接，并且这一种连接仅仅发生在这 P 和 Q 进程之间。

间接通信是消息传递的另一种方式。在这种情况，消息不直接从发送者发送到接收者，而是发送到暂存消息的共享数据结构组成的队列，这个实体称为信箱（mailbox）。因此二个进程通信情况，一个进程发送一个消息到某个信箱，而另一个进程从信箱中摘取消息。

间接通信的使用好处是增加了使用消息的灵活性。发送者和接收者的关系可能是一对一、多对一，一对多或多对多。一对一的关系允许一个专用通信链路用于二个进程间的交互，进程间交互不受其它进程错误的影响；多对一的关系对客户／服务器交互特别有用，一个进

程对多个其它进程（用户）提供服务，在这种情况信箱经常称作为端口；一对多关系允许一个发送进程和多个接收进程交互，这可用来将消息广播给一组进程。

（3）管道通信

管道（pipe）是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件，又称为管道文件。向管道（共享文件）提供输入的发送进程（即写进程），以字符形式将大量的数据送入管道，而接收管道输出的接收进程（即读进程）可从管道中接收数据。由于发送进程和接收进程是利用管道通信的，故又称管道通信。管道通信是基于文件系统形式的一种通信方式。

管道分为无名管道和有名管道两种类型。无名管道是一个只存在于文件系统中的一个临时文件，从结构上没有文件路径名，不占用文件目录项。有名管道是建立在文件系统中长期存在的、具有路径名的文件，因而其它进程可以知道它的存在，并能利用该路径名来访问该文件。