第3章　进程

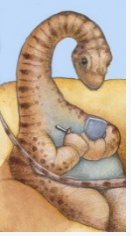
内容

1. 进程概念
2. 进程操作
3. 进程调度
4. 进程间通信

1、进程概念

进程(Process)

- 进程是操作系统执行CPU调度和资源分配的单位
- 操作系统执行各种程序
  - 批处理系统 – 作业(Job)
  - 分时系统 – 用户程序或任务(Task)
- 本书使用的名词作业和进程，基本可互换
  - 作业：被组装成一个整体运行的一组计算步骤
  - 任务：进程或线程
- 进程 – 执行中的程序；进程的执行必须以顺序方式进行（非正式）
- 另一个说法：一个程序在一个数据集上的一次执行



进程例子： Suse Linux

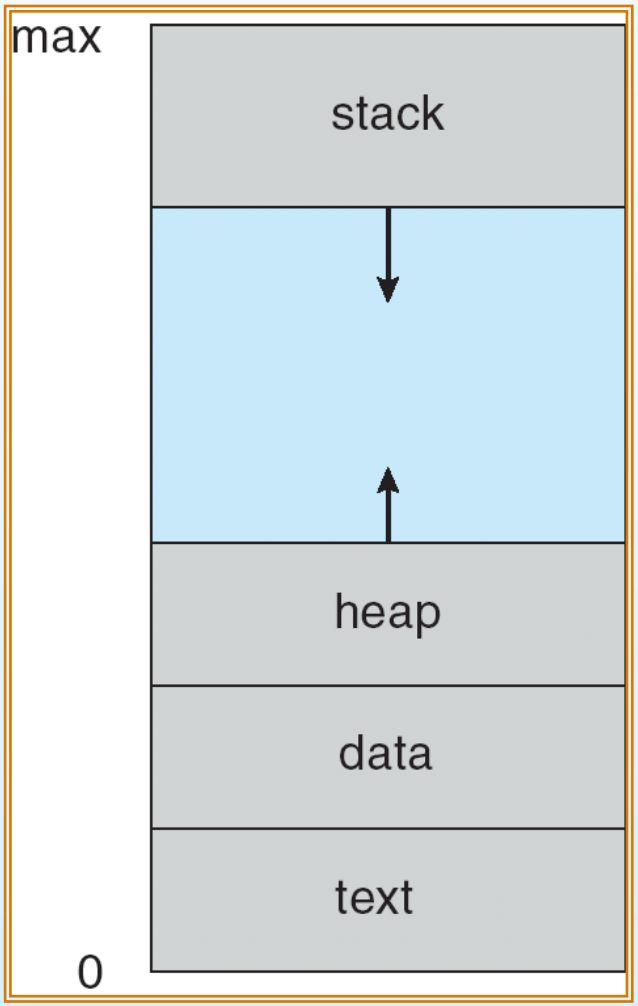
Telnet 202.195.128.17

```
  868 ?        02:23:36 oracle
  870 ?        00:00:35 oracle
  872 ?        00:00:01 oracle
  874 ?        00:18:38 oracle
  876 ?        00:17:29 oracle
  878 ?        00:17:28 oracle
  880 ?        00:17:22 oracle
  882 ?        00:16:25 oracle
  884 ?        00:17:05 oracle
  886 ?        00:17:52 oracle
  888 ?        00:17:19 oracle
  890 ?        00:16:41 oracle
  892 ?        00:17:54 oracle
  913 ?        02:49:25 tnslsnr
  985 ?        00:00:37 master
  999 ?        00:00:00 atd
 1014 ?        00:00:07 cron
 1030 ?        00:01:57 nscd
 1031 ?        00:00:42 nscd
 1032 ?        00:01:55 nscd
 1033 ?        00:01:48 nscd
 1034 ?        00:01:45 nscd
 1035 ?        00:01:48 nscd
 1036 ?        00:01:48 nscd
--More--
```

进程例子： Windows XP

内存中的进程

- 进程包括
  - 代码（**Text**）
  - 当前活动
    - **程序计数器（PC）**-指向当前要执行的指令（地址）
    - **堆栈（Stack）**：存放函数参数、临时变量等临时数据
    - **数据（Data）**：全局变量，处理的文件
    - **堆（Heap）**：动态内存分配

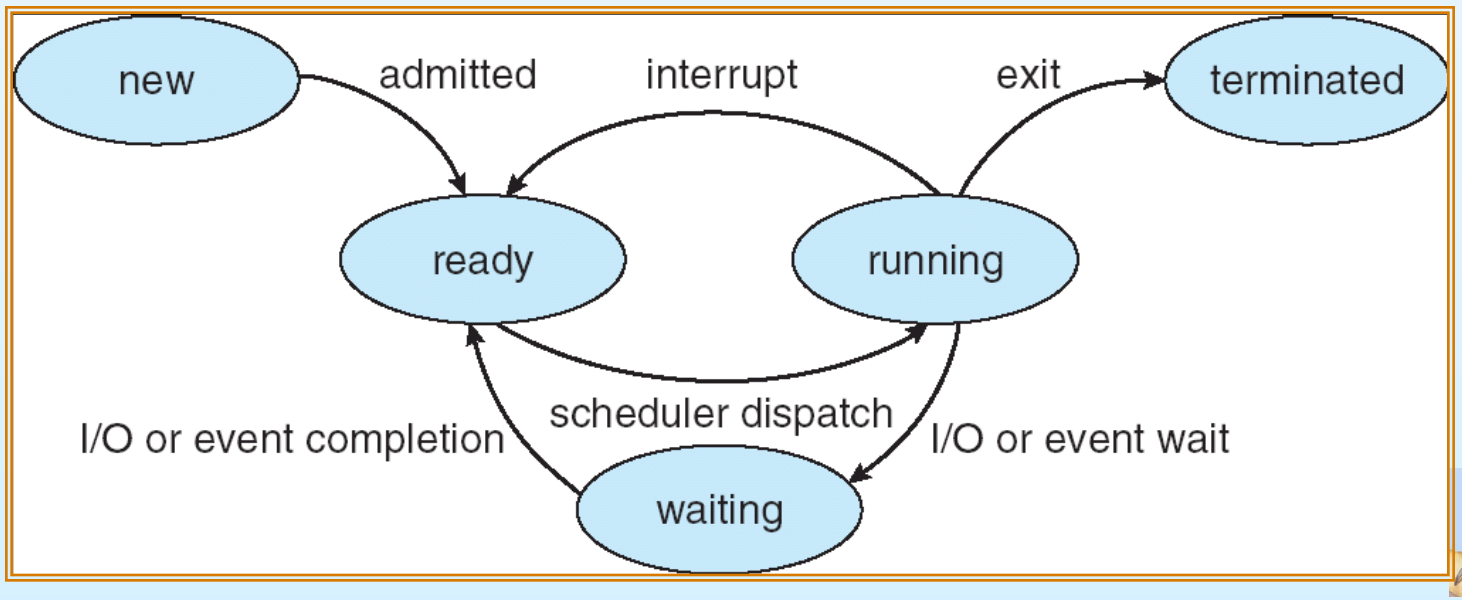


进程和程序

- 进程是程序的一个实例，是程序的一次执行
- 一个程序可对应一个或多个进程，同样一个进程可对应一个或多个程序
- 程序是进程的代码部分
- 进程是活动实体，程序静止（被动）实体
- 进程在内存，程序在外存

进程状态

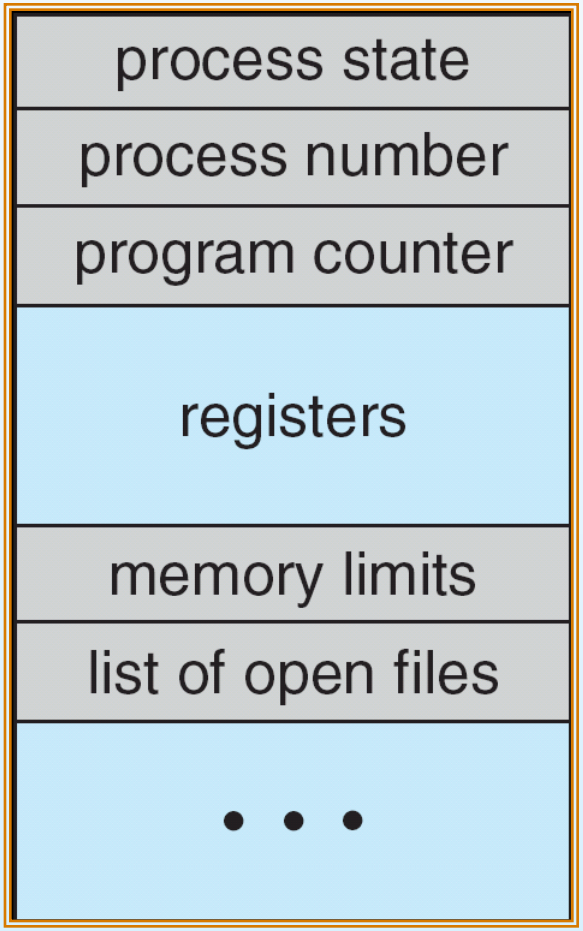
- 进程执行时，改变状态
  - 新建（new）：在创建进程
  - 就绪（ready）：进程等待分配处理器
  - 运行（running）：指令在执行
  - 等待、阻塞（waiting）：进程等待某些事件发生
  - 终止（terminated）：进程执行完毕



进程控制块(PCB)

PCB包含同进程有关的信息，包括：

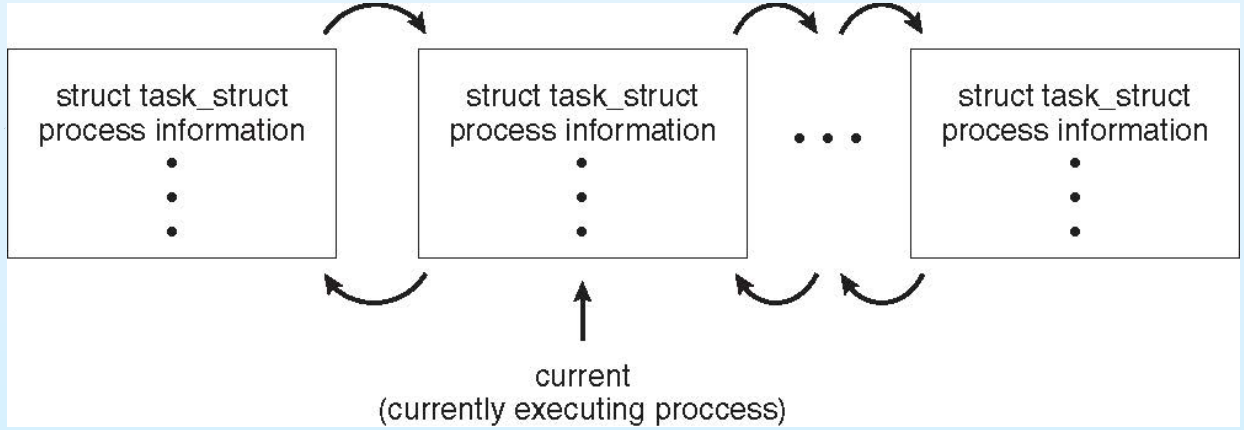
- 进程状态
- 进程号
- 程序计数器
- CPU寄存器
- CPU调度信息
- 内存管理信息
- 计账信息
- I/O状态信息



Linux PCB

C结构：task_struct

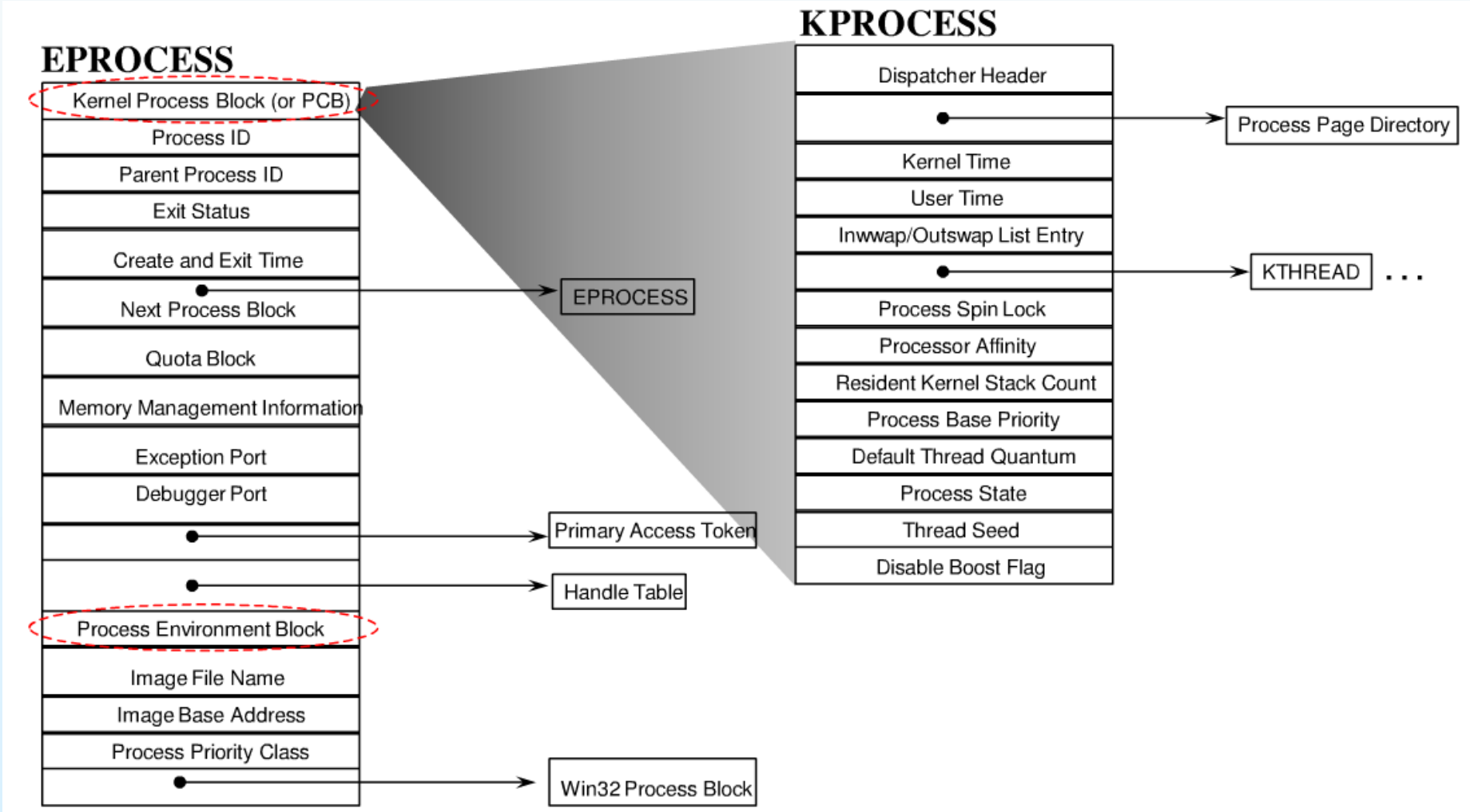
```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process
 */
```



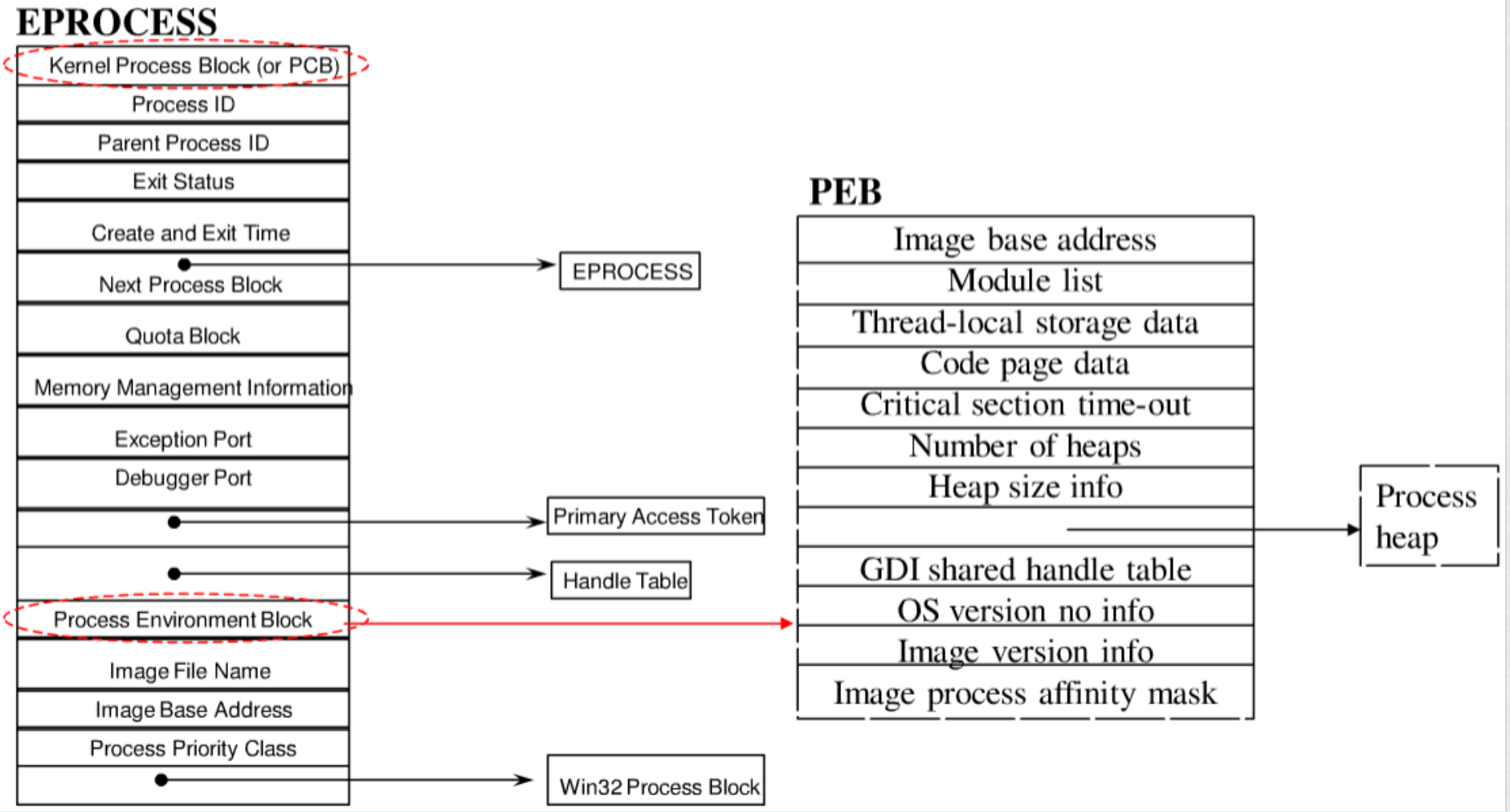
Windows PCB

- 每个Win32进程都由一个执行体进程块（executive process block）EPROCESS：PID, PCB, Access Token, Base Priority, 句柄表，指向进程环境块PEB指针，默认和处理器集合等
- Windows的PCB称为内核进程对象KPROCESS：
- 执行体进程对象EPOCESS和KPROCESS位于内核空间
- 进程环境块PEB （ Process Environment Block ），PEB位于用户空间

EPROCESS/KPROCESS

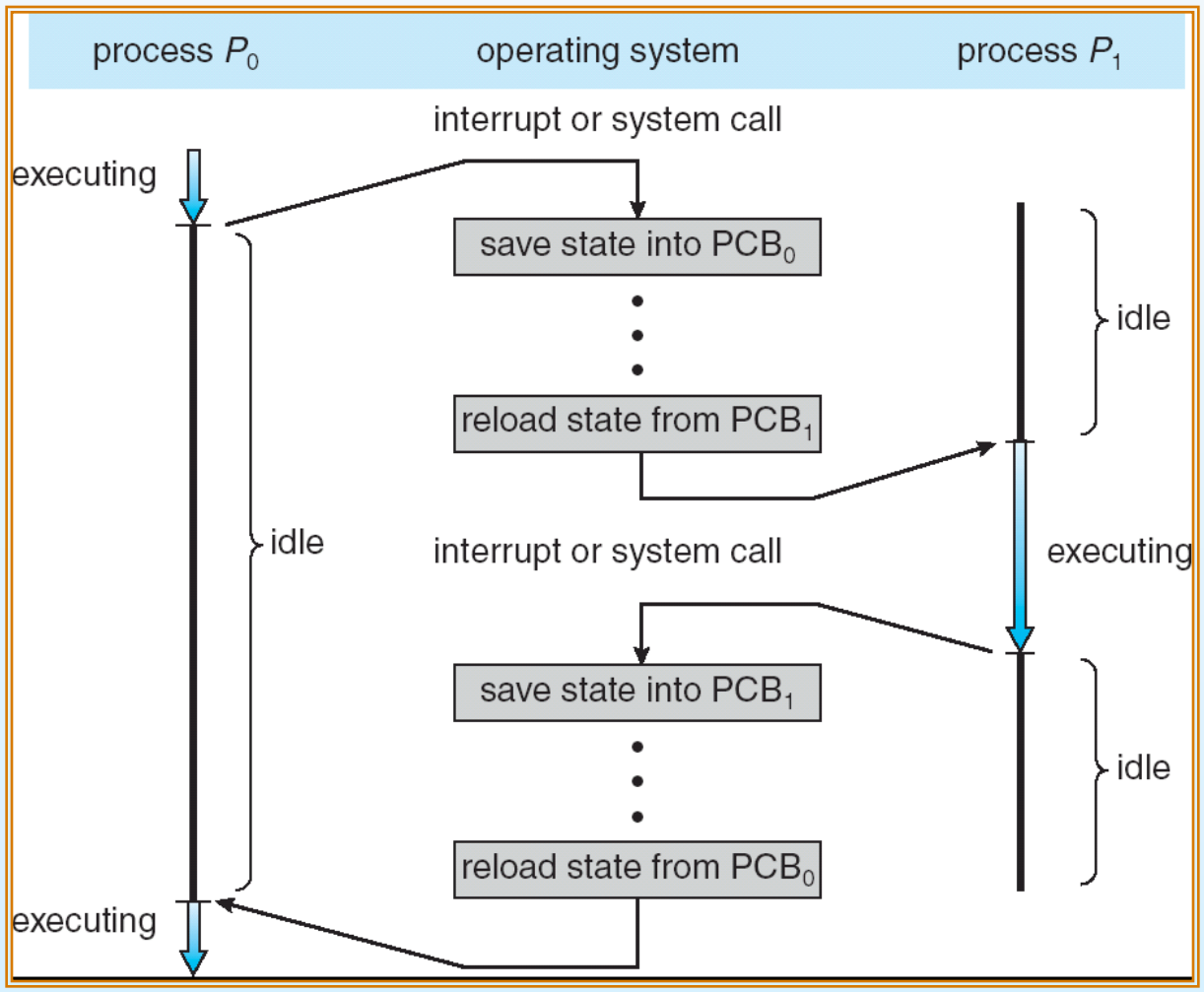


EPROCESS/PEB



CPU 在进程间切换

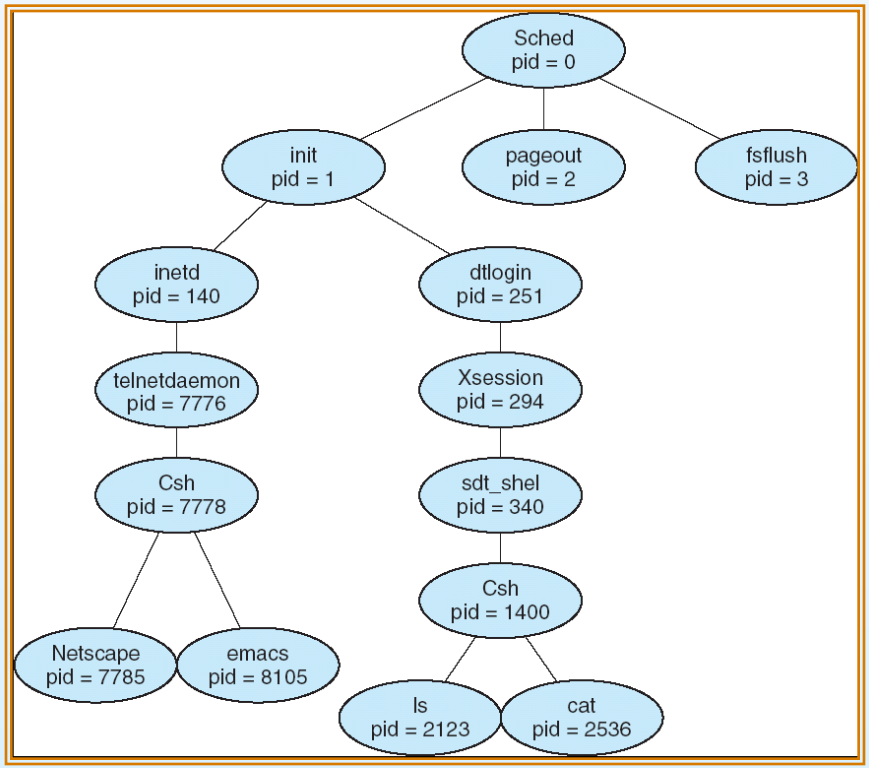
进程的并发执行需要PCB保存和恢复现场



2、进程操作

进程创建

- **父进程**创建**子进程**，如此轮流创建进程下去，构成一棵进程树
- 资源共享
  - 父进程子进程共享所有的资源
  - 子进程共享父进程资源的子集
  - 父进程和子进程无资源共享
- 执行
  - 父进程和子进程并发执行
  - 父进程等待，直到子进程终止



进程创建

- 地址空间
  - 子女复制双亲
  - 子女有一个程序被调入
- UNIX例子
  - **fork** 系统调用创建新进程，子进程完全复制父进程的空间，这种创建方式允许子进程和父进程进行方便的通讯
  - 在**fork** 用一个新程序替代了进程的内存空间之后，采用**exec**系统调用
- 原子操作
  - 该操作不能被打断，要么创建成功，要么创建失败

进程终止

- 进程执行最后一项并退出（exit）
  - 从子进程向父进程输出数据（通过wait）
  - 操作系统收回进程的资源
- 父进程可中止子进程的执行（abort）
  - 子进程超量分配资源
  - 赋予子进程的任务不再需要
  - 如果父进程结束
    - 若父进程终止，一些系统不允许子进程继续存在
      - 所有子进程终止-- **级联终止**
- 父进程可以等子进程结束
  - 如调用wait()系统调用

Windows进程操作

- CreateProcess：进程创建
  - 新进程可以继承父进程的一些资源：打开文件的句柄、各种对象（如进程、线程、信号量、管道等）的句柄、环境变量、当前目录、原进程的控制终端、原进程的进程组（用于发送Ctrl+C或Ctrl+Break信号给多个进程）——每个句柄在创建或打开时能指定是否可继承
  - 新进程不能继承：优先权类、内存句柄、DLL模块句柄
- ExitProcess和TerminateProcess：进程退出
- WaitForSingleObject:等待子进程结束

创建子进程API函数

- CreateProcess函数：

```
BOOL CreateProcess(
    LPCTSTR  lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTURINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);
```

结束进程

- 如果某个process想自己停止执行， 可调用ExitProcess()
  - C程序库中的exit(), exit()在自动执行一些清除垃圾工作后， 再调用ExitProcess()终止进程

    VOID ExitProcess(UNT uExitCode)

- 如果process A 想要process B 停止执行， 可在取得process B 的handle 后， 调用TerminateProcess():

  BOOL TerminateProcess(HANDLE hProcess, UNIT uExitCode)

ExitProcess——主动终止

TerminateProcess——强制终止

例子-子进程

- Child Process:

```
#include "stdafx.h"
#include <conio.h>
int _tmain(int argc, _TCHAR* argv[])
{
        wprintf(L"ParamTest Output:\n");
        wprintf(L" Number of parameters: %d\n", argc);
        wprintf(L" Parameter Info:\n");
        for (int c=0; c < argc; c++)
        {
        wprintf(L" Param #%d: %s\n", c, argv[c]);
        }
        return 0;
}
```

例子-父进程

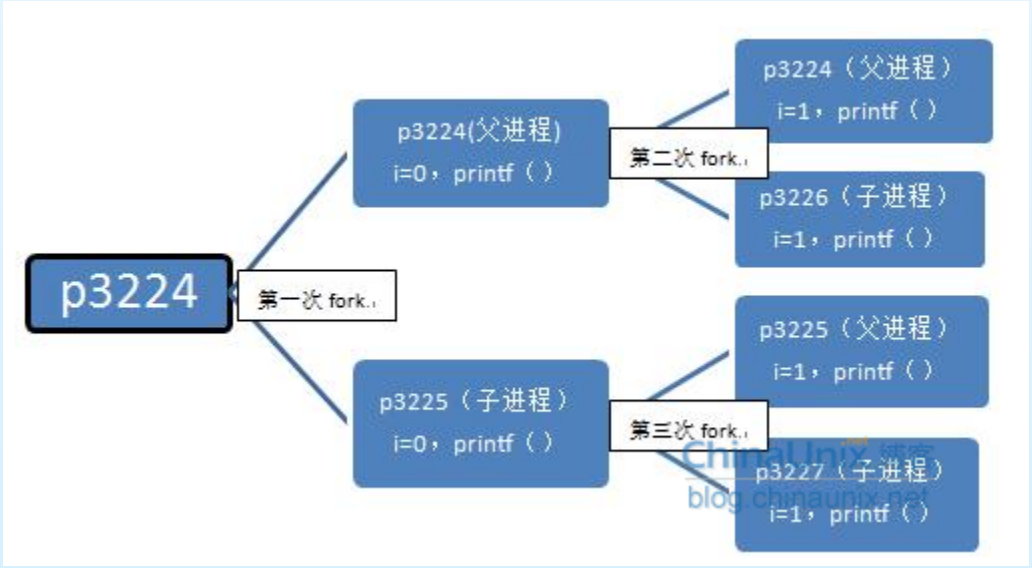
```
#include "stdafx.h"
 #include <windows.h>
 #include <strsafe.h>
 #include <direct.h>
 #include <string.h>
int _tmain(int argc, _TCHAR* argv[])
 {
     PROCESS_INFORMATION processInformation;
      STARTUPINFO startupInfo;
      memset(&processInformation, 0, izeof(processInformation));
      memset(&startupInfo, 0, sizeof(startupInfo));
      startupInfo.cb = sizeof(startupInfo);

      BOOL result;
      result = ::CreateProcess(AppName, CmdLine, NULL, NULL, FALSE,
       NORMAL_PRIORITY_CLASS, NULL, NULL, &startupInfo,
       &processInformation);
```

例子-父进程

```
if (result == 0)    {
        wprintf(L"ERROR: CreateProcess failed!");
    }    else    {
       WaitForSingleObject( processInformation.hProcess, INFINITE );
        CloseHandle( processInformation.hProcess );
        CloseHandle( processInformation.hThread );
    }
return0;
}
```

Linux进程创建

- fork 函数：
  - #include <unistd.h>
  - pid_t fork();
- 当一个进程调用fork 后会创建一个子进程
- 这个子进程和父进程不同：进程ID



父进程和子进程

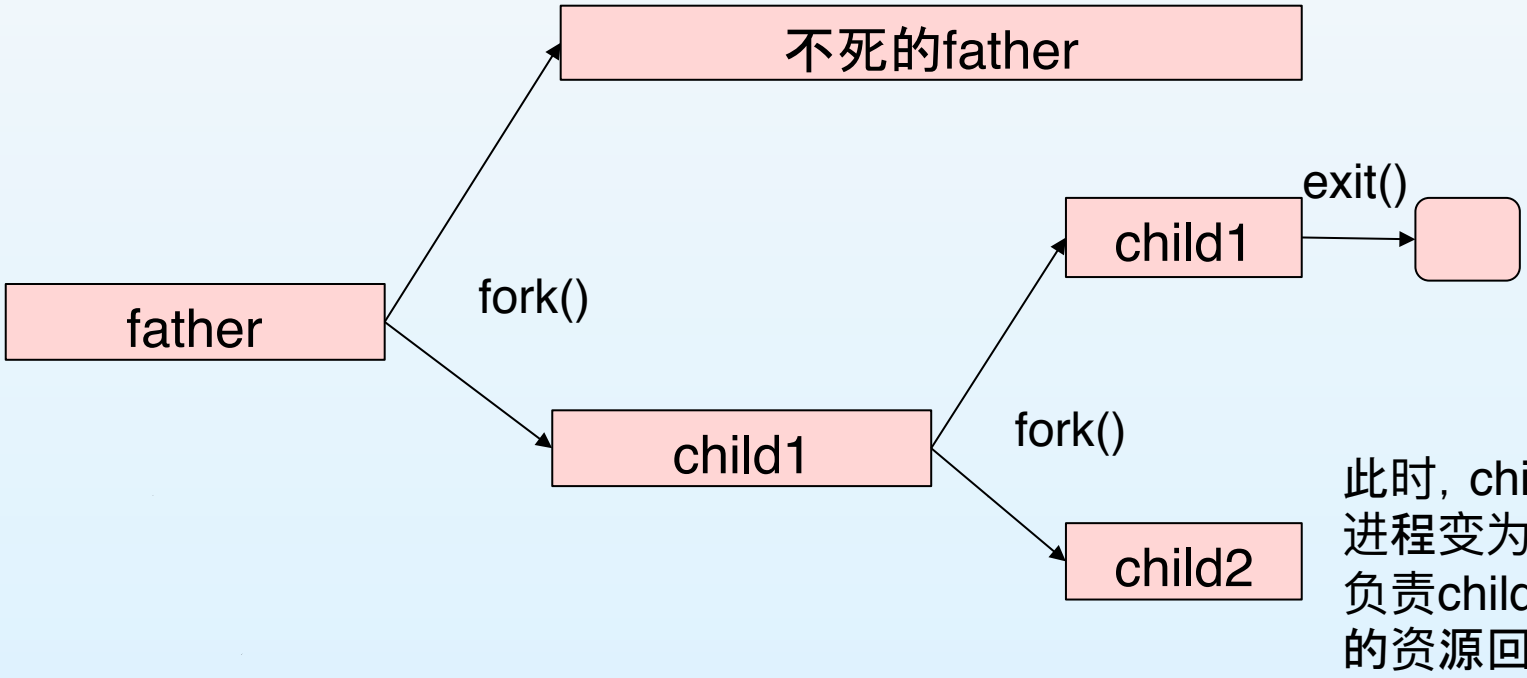
- 区分父进程和子进程:
  - 跟踪fork返回值
    - 失败:-1
    - 否则
      - 父进程fork 返回子进程的ID
      - fork 子进程返回0
- 可根据这个返回值来区分父子进程

执行其它程序

- exec 族调用有着6个函数：
  - #include <unistd.h>
  - int execl(const char *path,const char *arg,...);
  - int execlp(const char *file,const char *arg,...);
  - int execle(const char *path,const char *arg,...char *const envp[]);
  - int execv(const char *path,char *const argv[]);
  - int execvp(const char *file,char *const argv[]);
  - int execve(const char *path,char *const argv[],char *const envp[]);
- exec 族作用：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（在调用进程内部执行一个可执行文件）
- exec族的函数执行成功后不会返回

等待

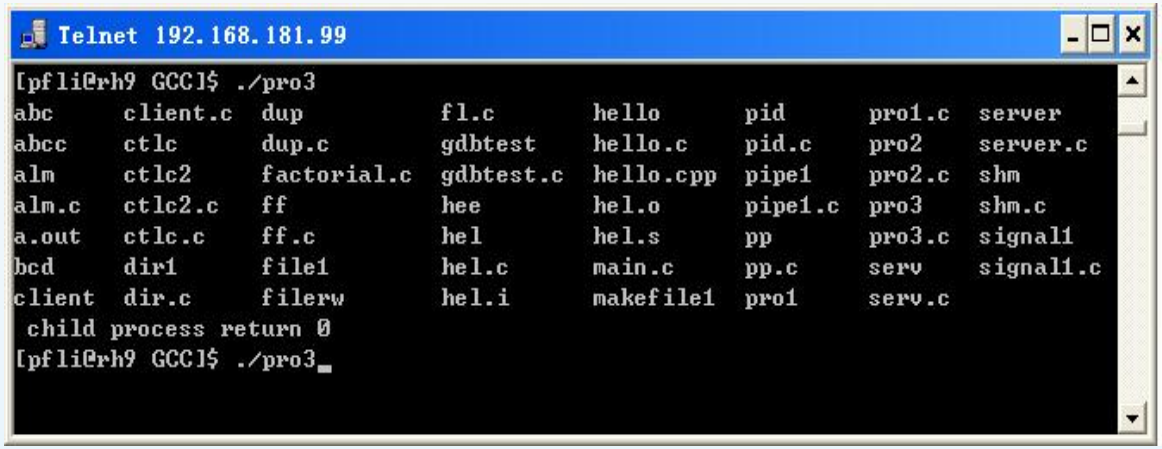
- 父进程阻塞直到子进程完成任务
  - 进程一旦调用了wait函数，将立即阻塞自己，由wait自动分析是否当前进程的某个子程序已经退出，如果让它找到了这样一个已经变成僵尸的子程序，wait就会收集这个子程序的信息，并把它彻底销毁后返回；如果没有找到这样一个子程序，wait就会一直阻塞在这里，直到有一个出现为止
- 调用wait 或者waitpid 系统调用
  - #include <sys/types.h>
  - #include <sys/wait.h>
  - pid_t wait(int *stat_loc);
  - pid_t waitpid(pid_t pid,int *stat_loc,int options);



此时，child2的父进程变为init，init负责child2终止时的资源回收

例子

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

int main()
{
int rtn; /*子进程的返回数值*/
if ( fork() == 0 ) {
/* 子进程执行此命令 */
execlp("/bin/ls","ls -al ",(char *)0);
/* 如果exec函数返回，表明没有正常执行命令，打印错误信息*/
exit( 1 );
}
else {
/* 父进程， 等待子进程结束，并打印子进程的返回值 */
wait ( &rtn );
printf( " child process return %d\n”, rtn );
}
}
```



操作原语

- 原子操作
  - 也称为操作原语
  - 操作系统内核中
  - 由若干条指令构成，用于完成一个特定的功能的一个过程
  - 该操作不能被打断，要么成功，要么失败

  - 创建进程原语 create(n)
  - 撤销进程原语 destroy(n)
  - 阻塞进程原语 block()
  - 唤醒进程原语 wakeup(n)

进程原语

- 进程创建原语的主要功能：
  - 为新建进程申请一个PCB
  - 将创建者（即父进程）提供的新建进程的信息填入PCB
  - 将新建进程设置为就绪状态，并按照所采用的调动算法，把PCB排入就绪队列。

进程原语

- 进程撤销
  - 该进程已经完成所要求的功能而正常终止
  - 由于某种错误导致非正常终止
  - 祖先进程要求撤销某个子进程
- 进程阻塞
  - 进程的执行状态变化到等待状态（一个进程期待某事件的发生，但发生尚不具备条件时），由该进程自己调用阻塞原语来阻塞自己。
  - 进程状态 运行->等待
- 进程唤醒
  - 唤醒一个进程的两种方式：（1）有系统进程唤醒；（2）由事件发生进程唤醒
  - 进程状态 等待->就绪