进程操作

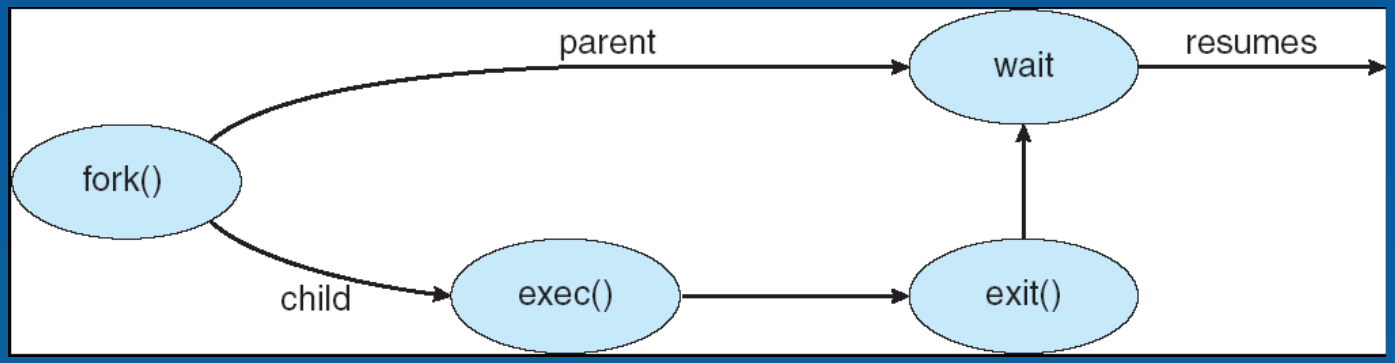
进程创建

- 父进程创建若干子进程；后者再创建其子进程；与此类推，构成了反映“传承”关系的一棵进程树
- 子进程的资源
  - 子进程共享父进程的所有资源
  - 子进程共享父进程的部分资源
  - 子进程不从父进程共享资源，重新独立申请
- 执行代码的执行顺序
  - 父进程和子进程并发执行
  - 父进程在子进程执行期间等待，待子进程执行完毕后才恢复执行余下代码

进程创建（Cont.）

- 地址空间中的 image
  - 子进程复制了（ duplicate ，不是 copy ）父进程的 image
  - 子进程全新装入一个程序，得到不同于父进程的 image
- 举例： UNIX 的进程创建
  - **fork** 系统调用创建一个新（子）进程
  - **fork** 之后， **exec** 系统调用装入一个新程序

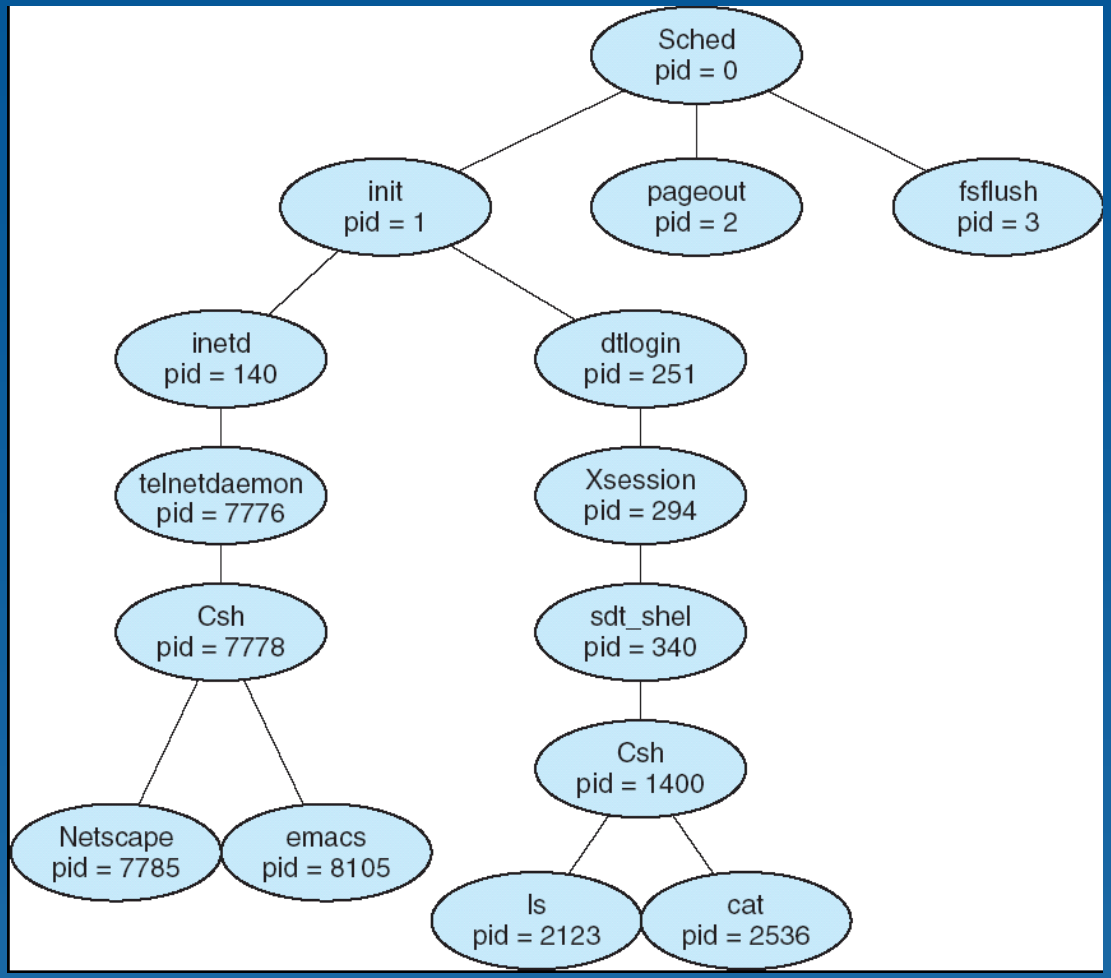
Unix 环境里创建子进程



Unix 环境 fork 一个进程

```
int main()
{
Pid_t  pid;
   /* fork another process */
   pid = fork();
   if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-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");
        exit(0);
   }
}
```

Solaris 系统中常见的一棵进程树



进程终止

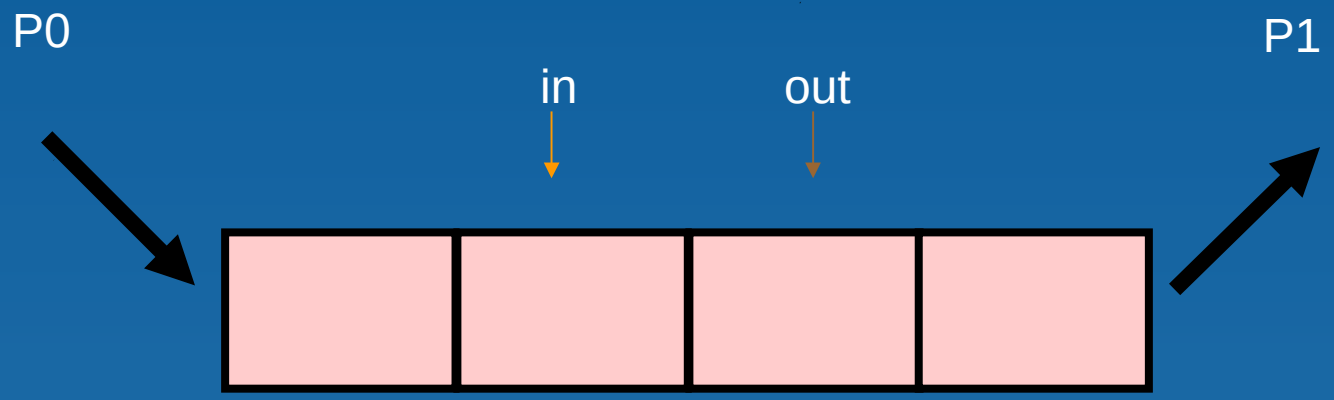
- “进程终止”语义之一：子进程执行完最后一条指令后，要求操作系统将自己杀出（**exit**）。语义动作含：
  - 子进程传递数据给父进程（通过父进程的 **wait** 操作）
  - 子进程的资源被操作系统收回
- “进程终止”语义之二：父进程终止子进程的执行（**abort**）。终止原因可能是：
  - 子进程超额使用系统资源
  - 早前交给子进程执行的任务，过期无效了
  - 如果父进程终止了，它的子进程怎么办?
  - 有些操作系统把这些子进程也全部终止
    - All children terminated - *cascading termination*

进程间合作

- 独立进程不会影响其它进程的执行，也不被影响
- 合作进程影响其它进程，或者受其影响
- 进程间合作是必须的，带来的好处：
  - 共享信息
  - 加速（计算）执行任务
  - 模块化
  - 方便调用，等等……

经典案例：生产者 - 消费者问题

- 生产者进程“生产”出信息，存储在缓冲区，供消费者进程“消费”
  - *unbounded-buffer* 缓冲区的容量无限
  - *bounded-buffer* 缓冲区的容量有限



Bounded-Buffer – “共享内存”解决方案

- 数据结构
  - #define BUFFER_SIZE 10
  - Typedef struct {
  - . . .
  - } item;
  - item buffer[BUFFER_SIZE];
  - int in = 0;
  - int out = 0;
- 解决方案可行
- 但是只能用足 BUFFER_SIZE-1 个缓冲单元

Bounded-Buffer – 生产者进程的“存”

```
while (true) {
    /* Produce an item */
        while (((in = (in + 1) % BUFFER
SIZE count)
                        == out)
        ;    /* do nothing -- no free
buffers */
     buffer[in] = item;
     in = (in + 1) % BUFFER SIZE;
    {
```

Bounded Buffer – 消费者进程的“取”

```
while (true) {
        while (in == out)
                ; // do nothing --
nothing to consume

     // remove an item from the
buffer
     item = buffer[out];
     out = (out + 1) % BUFFER SIZE;
return item;
    {
```

Interprocess Communication (IPC)

- 进程间通信 IPC，提供一套进程通信、进程同步的机制
- 消息系统 – 进程间相互通信的途径，不需要有共享变量的介入

- IPC 机制有 2 个最基本的进程操作:
  - **send**(*message*)
  - **receive**(*message*)
- 变种: Direct Communication
  - **send** (*P*, *message*) – 直接发消息给进程 P
  - **receive**(*Q*, *message*) – 直接接收来自进程 Q 的消息
- 变种: Indirect Communication
- **send(A**, *message*) – 发送消息给邮件服务器 A
- **receive**(*A*, *message*) – 从邮件服务器 A 接收消息

同步通信 vs 异步通信

- **同步通信**
  - **发送操作 send：** 发送进程等待，直至接收进程确认收到消息
  - **接收操作 receive：** 接收进程等待，直至有个消息到达
- **异步通信**
  - **发送操作 send：** 发送进程发出消息后即返回，该干什么干什么，不理会消息是否送达
  - **接收操作 receive：** 接收进程执行一次接收动作，要么收到一条有效消息，要么收到空消息


End