第二章 作业管理和用户接口

在这一章中，我们讨论OS向上提供的用户接口，即系统命令接口和系统调用接口。系统命令接口可完成用户作业的组织和控制。

2.1 作业组织和控制

返回

2.1.1 作业和作业处理过程

1. 作业的概念
2. 作业的组成
3. 作业的处理过程
4. 作业输入方式
5. 作业控制表(JCB, Job Control Block)

1. 作业的概念

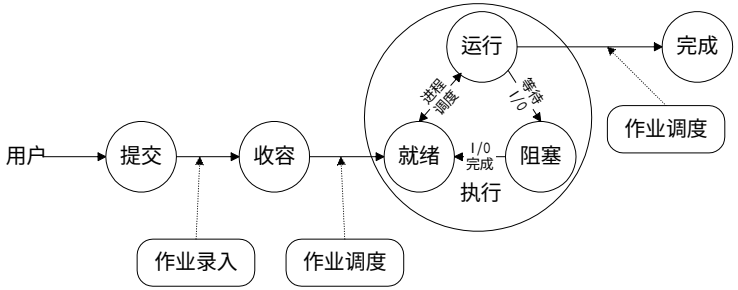
- 一个作业是指在一次应用业务处理过程中，从输入开始到输出结束，用户要求计算机所做的有关该次业务处理的全部工作。
  - 用户的观点：在一次业务处理过程中，从输入程序和数据到输出结果的全过程。作业步：形成中间结果文件。
  - 系统的观点（针对作业进行资源分配）：作业由程序及数据（作业体）和作业说明书（作业控制语言）
- 作业由不同的顺序相连的作业步组成。
- 作业步是在一个作业的处理过程中，计算机所做的相对独立的工作。

2. 作业的组成

作业由程序、数据和作业说明书三部分组成。

- 作业说明书包括作业基本情况、作业控制、作业资源要求的描述；它体现用户的控制意图。如：预计运行时间、要求的资源情况、执行优先级等。
  - 作业基本情况：用户名、作业名、编程语言、最大处理时间等；
  - 作业控制描述：作业控制方式、作业步的操作顺序、作业执行出错处理；
  - 作业资源要求描述：处理时间、优先级、内存空间、外设类型和数量等；

3. 作业的处理过程



- 作业提交：
  - 作业的输入；
- 作业执行：
  - 先到"就绪"，经调度"运行"，有I/O请求"等待"，I/O完成到"就绪"
- 作业完成：
  - 作业的输出；

4. 作业输入方式

- 无通道处理方式：
  - 联机（降低了CPU效率）：由主机直接控制输入/输出；I/O与作业处理不能并行。
  - 脱机（人工干预）：通过磁带或磁盘在外围处理机与主机之间交换作业（需人工移动）；用于主机不太快的情况。
  - 直接耦合：通过共享外存在外围处理机与主机之间交换作业；用于高速主机。
- 有通道处理：输入/输出由主机和通道来承担。
  - 假脱机(spooling, Simultaneously Peripheral Operation On Line)：系统把作业处理的全过程划分为相对独立的三个部分--输入流、处理流和输出流。spooling-in/spooling-out进程：控制输入/输出；
  - 网络：在网络上一台机器进行作业输入/输出，在另一台主机上运行；

5. 作业控制表(JCB, Job Control Block)

在运行过程中，系统对作业进行管理的必要信息。

- 作业名
- 估计执行时间
- 优先数（用于调度）
- 作业说明书文件名
- 程序类型（需调用的系统程序）
- 资源要求：（静态，或中间可以随作业步变化－－效率不高；动态分配
- 作业状态：提交、后备、执行、就绪、等待、完成；

2.1.2 作业调度

检查系统是否满足作业的资源要求，并一定算法选取作业。作业调度也称为宏观调度。

- 作业调度算法的评价因素
  - 作业吞吐量：运行尽可能多的作业；
  - 充分利用资源：CPU忙、I/O设备忙；
  - 对各作业公平、合理，使用户满意：执行时间长短、等待时间等；

2. 作业调度算法

实际的算法可能会是多种算法的综合。

- 先来先服务（FCFS）：按照作业进入系统的先后次序进行调度，先进入系统者先调度；即启动等待时间最长的作业。
  - 优点：实现简单、公平
  - 缺点：没考虑资源利用率和作业的特殊性
- 短作业优先（SJF）：以要求运行时间长短进行调度，即启动要求运行时间最短的作业。
  - 优点：易于实现，强调了资源的充分利用，保证了系统的最大吞吐量（单位时间里处理作业的个数）。
  - 缺点：不公平，会造成长作业长期等待。
  - 结论：假设系统中所有作业同时到达，可以证明采用SJF能得到最短的作业平均周转时间。

- 高响应比优先（HRF）：响应比最高的作业优先启动。
  - 响应比=（等待时间+估计运行时间）/ 估计运行时间
  - 该算法是FCFS和SJF的结合，克服了两种算法的缺点
  - 优点: 公平，吞吐率大
  - 缺点: 增加了计算，增加了开销
- 高优先级优先：由用户指定作业优先级，优先级高的作业先启动。
- 资源均衡型调度：把作业分类，作业调度从不同类型作业中去调度作业
  - 根据作业对资源要求分类：I/O型、CPU型和均衡型

2.1.3 作业控制语言

- 脱机作业控制：用户输入作业说明书，整个作业的运行由系统控制。
- 联机作业控制：通过人-机会话方式控制作业运行。用户登录（控制台登录或远程登录），由系统自动执行一些命令脚本后，并进入shell（字符或GUI界面），接受用户的命令和操作，最后退出系统。

1. 命令行
2. 环境变量

1. 命令行

- 命令行：一行可有一个或多个命令，每次一行，包含一个或多个命令。
  - shell给出提示符时可输入，以回车键提交。如：
  - “ls -a -l”列出当前目录文件列表；
  - "gunzip mp1.tar.gz; tar -xvf mp1.tar; \rm -r -f mp1.tar"为解压缩后再展开。
- 命令格式：一个命令可有命令参数，格式包括选项/开关 (option/switch)或参数(argument)。
  - 如UNIX系统： cp -r doc /tmp－－argv[0], argv[1], ...（含子目录的文件复制：/tmp为目标地址）

- 命令分类：内部命令和外部命令
  - 内部命令：直接由shell本身完成，功能简单、使用频繁；如：DOS的copy命令。
  - 外部命令：运行相应的可执行文件，在使用时加载。如：DOS的xcopy命令。
- 命令简化：利用参数替换可简化命令输入，通配符(?, *)用于匹配一组文件名
  - 如：UNIX的cp命令：当前目录上有两个"1.tar"和"2.tar"时，"cp *.tar /tmp"等同于"cp 1.tar /tmp; cp 2.tar /tmp"

2. 环境变量

环境变量(environment variable)－－应用进程地址空间中的特殊变量区。

- 环境变量也可以作为shell参数，如命令提示符的式样，外部命令的查找目录路径等。
  - 如：PATH=/bin:/usr/bin:/usr/sbin:.
  - HOME=/home/thisuser
- 环境变量是传递命令参数的另一种途径。如：
  - cd $HOME;
- 环境变量可按名字访问，可以新建、赋值或撤销。
- 有效范围：只对本进程里的环境变量能够直接进行操作。此外，在执行新进程时（UNIX中是exec()调用），可以给出环境变量的初始值，通常就是直接复制当前进程的所有环境变量－－继承。

2.2 作业管理举例

返回

2.2.1 MS DOS的作业管理

2.2.1.1. DOS命令处理程序

command.com（或其他程序如DOS shell）：驻留内存，在系统运行期间不再退出。为了给应用程序的执行提供更大内存空间，又分为常驻部分和暂驻部分（可被应用程序覆盖）

- 命令分类：
  - 内部命令：如dir, cd, copy
  - 外部命令：如format, xcopy
- 命令行选项
  - 命令行选项通常是：/option；如："/?"选项可显示各命令的命令行选项列表。
  - 通配符：由外部命令自己处理。如：xcopy *.c . －－argv[1]="*.c"

- 输入输出重定向和管道(pipe)
  - <, >, >>, |, 基于临时文件
  - "<"为输入重定向，如："find "string" < temp.txt"将显示文件"temp.txt"中有"string"串的行；"more < temp.txt"将逐屏显示输出文件"temp.txt"的内容；
  - ">"为输出重定向，">>"为添加输出重定向。如：
    - "dir > temp.txt"将把"dir"命令在屏幕上的输出保存在新文件"temp.txt"中；
    - 而"dir >> temp.txt"将屏幕输出追加在文件"temp.txt"的结尾。
  - 管道"|"是将前一个命令的屏幕输出作为后一个命令的键盘输入。如："dir | sort"将把"dir"命令的输出按行进行排序。
- 环境变量
  - set PATH=c:\tools;%PATH%－－原PATH=c:\dos 则后PATH=c:\tools;c:\dos
- 系统引导时加载
  - 系统引导时加载：autoexec.bat

2.2.1.2. DOS批处理：由command.com执行

有简单的变量替换，有条件转移和跳转、循环和注释语句rem

- 循环：循环执行命令。
  - FOR /R [[drive:]path] %variable IN (set) DO command [command-parameters]
  - 遍历根在[drive:]path上的目录树，在树的每个目录中执行 FOR 语句。如果在 /R 后没有指定任何目录规范，那么假设为当前目录。如果 set 仅是一个句号(.)，那么它 将仅列出目录树。

关于循环的实例

如下面批处理将显示当前目录及其子目录所有文件名(含路径名)；

```
for /R %%f in (*.*) do echo %%f
```

如下面批处理将显示当前目录及其子目录所有后缀为ppt的文件名(含路径名)；

```
for /R %%f in (*.ppt) do echo %%f
```

可能的显示结果：

C:\users\xyong\work\2001-02-20 chapter1.ppt

C:\users\xyong\work\temp\2001-02-18 chapter1.ppt

C:\users\xyong\work\temp\2000-08-09 Linux Lecture\2000-08-15 Linux.ppt

分支

IF [NOT] ERRORLEVEL number command
IF [NOT] string1==string2 command
IF [NOT] EXIST filename command

NOT	只有在条件为假时，才需运行命令。
ERRORLEVEL number	如果最近程序运行返回的退出码等于或大于指定的号码时，则设定条件为真。
command	当符合条件时，指定要运行的命令。
string1==string2	当指定字符串匹配时，设置条件为真。
EXIST filename	当指定文件名存在时，设置条件为真。

如：
IF ERRORLEVEL 1 goto okay
:okay
echo okay

2.2.2 UNIX的作业管理

2.2.2.1. shell命令处理程序

1. shell的类型

UNIX上有许多种shell，主要功能是相同的，在细节上有一些区别。几种shell 都有它们的优点和缺点。

- Bourne shell(/bin/sh)
  - Bourne shell 的作者是 Steven Bourne。它是 UNIX 最初使用的shell 并且在每种 UNIX 上都可以使用。Bourne shell 在 shell 编程方面相当优秀，但在处理与用户的交互方面作得不如其他几种 shell。
  - Bourne shell 最大的缺点在于它处理用户的输入方面。在 Bourne shell 里键入命令会很麻烦，尤其当你键入很多相似的命令时。
- C shell(/bin/csh)
  - C shell 由 Bill Joy 所写，它更多的考虑了用户界面的友好性。它支持象命令补齐（command-line completion）等一些 Bourne shell 所不支持的特性。普遍认为C shell 的编程接口做的不如 Bourne shell, 但 C shell 被很多 C 程序员使用因为 C shell的语法和 C语言的很相似，这也是C shell名称的由来。

- Korn shell (ksh)
  - Korn shell (ksh) 由 Dave Korn 所写。它集合了C shell 和 Bourne shell 的优点并且和 Bourne shell 完全兼容。
- Bourne Again shell (bash)
  - Bourne Again shell (bash)是 Bourne shell 的扩展。bash 与 Bourne shell 完全向后兼容，并且在 Bourne shell 的基础上增加和增强了很多特性。bash 也包含了很多 C 和 Korn shell 里的优点。bash 有很灵活和强大的编程接口，同时又有很友好的用户界面。 bash 有几种特性使命令的输入变得更容易。
- 其他shell
  - 除了这些 shell 以外，许多其他的 shell 程序吸收了这些原来的 shell 程序的优点而成为新的 shell 。如：
  - tcsh (csh 的扩展)
  - Public Domain Korn shell (pdksh, ksh 的扩展)

2. 初始化文件

- B Shell执行.profile
- C Shell执行.cshrc和.login两个文件
- K Shell执行.profile和$ENV两个文件
- (sh) .profile与(csh) .login对应，进行注册时的初始化；而在csh在非注册启动时都读入.cshrc；

3. 基本特征

- 内部命令：如cd, exec——区分大小写，exec的功能是执行一个命令；
- 外部命令：如ls, mkdir
- 命令行选项通常是：-option
  - 如："ls -a -l"中的-a表示列出所有文件，-l表示列出所有信息。
- 通配符：由shell处理后再传递给外部命令。
  - 如：cat *.c 则argv[1]="a.c", argv[2]="b.c"，而 cat "*.c" 则argv[1]="*.c"(cat的功能是读入所有文件，并显示)

4. 输入输出重定向

基于内核的缓冲区

- "<"为标准输入重定向；
- ">"和">>"为标准输出重定向；
- "2>"和"2>>"为标准错误输出重定向（2表示标准错误输出的设备号，只对sh有意义）；
- " >&"是标准输出和标准错误输出重定向；

行输入重定向：用定界符间的内容作为标准输入。
如：下面命令的标准输入为邮件内容。

```
mail user2 << WARNING
...
WARNING
```

5. 管道

管道可以把一系列命令连接起来。第一个命令的输出会通过管道传给第二个命令而作为第二个命令的输入，第二个命令的输出又会作为第三个命令的输入，以此类推。而管道行中最后一个命令的输出才会显示在屏幕上（如果命令行里使用了输出重定向的话，将会放进一个文件里）。通过管道，可以将多个简单程序组合完成复杂的功能。

- 如："ls -l | wc -l"可给出文件数目。
- 如："cat sample.text | grep "High" | wc -l" 这个管道将把 cat 命令（列出一个文件的内容）的输出送给grep命令。grep 命令在输入里查找单词 High，grep命令的输出则是所有包含单词 High的行，这个输出又被送给 wc命令。带 -l选项的 wc命令将统计输入里的行数。

6. 后台执行

后台执行：cmd &;

- 如："xterm -display 166.111.68.56:0.0 &"为在后台启动一个xterm窗口，并显示到主机166.111.68.56上。

7. 环境变量和内部变量

内部变量不能被子进程继承（如同C里的局部变量）；改环境变量就会自动改内部变量，反之不然。

- "set"可给出内部变量列表，"env"可给出环境变量列表。继承只对环境变量有效。
- sh: PATH=/usr/bin:$PATH ; export PATH－－注意：在export前为内部变量，之后为环境变量。
- csh: set setenv PATH /usr/bin:$PATH--注意：在csh中环境变量的赋值(setenv)没有等号，而内部变量的赋值(set)有等号。

8. 别名

给复杂命令定义别名：alias/unalias

alias 的格式:

alias aliasname=string ….

把 aliasname 直接用来取代后面的 string，如有任何跟在后面的 argument 将会出现的其后。利用该功能，使用者可以将常用却冗长的指令以其他的名字存起。

如："alias dir='ls -a -l'"为"ls -a -l"定义了一个别名"dir"；

9. 常用的外部命令

- man查看手册
- echo, wc, grep, sed, awk（用于文本扫描和处理）, sort, cut（对每行进行特定删除处理）字符串操作;
- pwd, ls, mkdir, rmdir, cp, rm, mv, ln文件和目录操作;
- chmod, chown, chgrp（修改文件所在的用户组）文件权限和属主;
- cat, more, tail（显示文件的最后部分）查看文件;
- test, expr检测和数值计算;
- vi全屏幕编辑;

2.2.2.2. 批处理

1. 变量替换
2. 条件转移
3. 循环
4. 分支
5. 函数

- 称为"脚本" shell script：注释用 '#'，续行用 " \"
- 解释执行，效率较低；而且要加载外部命令；
- 指定解释执行脚本的程序：#!/bin/sh 或 #!/opt/bin/perl
- perl, Practical Extraction and Report Language是一个文本文件分析工具。
- 执行脚本(sh)：cmd([csh]source cmd), exec cmd, cmd(sh < cmd, sh cmd)
- "source"命令是运行tcl脚本；"exec"是用指定命令新建shell，以取代当前shell；"sh"命令是运行sh脚本。

1. 变量替换

变量赋值格式：变量名=值

值串中包括空格、制表符或换行符时，值要放在""号内；

变量引用格式：$变量名

如引用前后紧接其它字符，引用方式为：${变量名}或"$变量名"；

$variable, 双引号：“进行变量替换”, 单引号：'不作任何替换'；

如：下面是一个变量定义和引用的例子：

```
temp="test message"
echo $temp
echo AAA${temp}BBB
echo CCC"$temp"DDD
echo '$temp'
```

其运行结果为：

```
test message
AAAtest messageBBB
CCCtest messageDDD
$temp
```

2. 条件转移

if command; then ... ; else ... ; fi

例：

```
# 调用test命令，注意：方括号和判断条件之间必须有空格 !!
# 下面脚本在有参数时显示"Have arguement(s)"，无参数时显示"No arguement"；
# 其中，"[  "a$1" = "a"  ]"用于判断$1是否为空串；
if [  "a$1" = "a"  ]; then
  echo "No arguement"
else
  echo "Have arguement(s)"
fi
```

运行结果为：

```
[xyong@well ~/work]./temp.txt
No arguement
[xyong@well ~/work]./temp.txt we
Have arguement(s)
```

例：
下面脚本在命令行参数个数不对（其中，"test $# = 4"测试命令行参数个数是否为4），提示出错信息；
```
if test $# = 4 ; then
        echo $4 $3 $2 $1
else
        echo $0 usage: arg1 arg2 arg3 arg4
fi
```

运行结果如下：
```
[xyong@well ~/work]$ ./temp.txt
./temp.txt usage: arg1 arg2 arg3 arg4
[xyong@well ~/work]$ ./temp.txt a b c d
d c b a
```

3. 循环

while [1]; do ... ; done

例：
下面脚本在文件lockfile可读时每暂停5秒重复测试一次（其中，"test -r lockfile"判断文件"lockfile"是否存在）；
```
while test -r lockfile; do
        sleep 5
done
```

运行结果为：有文件"lockfile"时，脚本一直不结束；直到该文件被删除后，脚本才结束。

4. 分支

基于模式匹配的多路分支结构，它依据word的不同，执行不同的命令序列；
```
case word in
        pattern1) pat1-list;;
        pattern2) pat2-list;;
esac
```

例：
下面脚本依据命令行参数的不同，显示不同内容；
```
case $1 in
  *.c) echo C:"$1" ;;
  *.a) echo A:"$1" ;;
  *.f) echo F:"$1" ;;
esac
```

运行结果如下：
```
[xyong@well ~/work]$ ./temp.txt c.a
A:c.a
[xyong@well ~/work]$ ./temp.txt c.c
C:c.c
[xyong@well ~/work]$ ./temp.txt c.f
F:c.f
```

5. 函数

例：
#这里$*和$1是new_func的参数，而$0为脚本的参数）
```
new_func () {
  echo $*
  echo $1
}
new_func $0 arg2
```

运行结果为：
```
[xyong@well ~/work]./temp.txt asb
./temp.txt arg2
./temp.txt
```

2.3 系统调用(SYSTEM CALL)

系统调用是操作系统提供给软件开发人员的唯一接口，开发人员可利用它使用系统功能。OS核心中都有一组实现系统功能的过程（子程序），系统调用就是对上述过程的调用。

返回

2.3.1 系统调用及其功能

每个操作系统都提供几百种系统调用，包括：外存文件与目录的读写，各种I/O设备的使用，在程序中启动另一个程序，查询和统计系统资源使用情况等等。

1. 系统调用的功能

1) 设备管理：设备的读写和控制；

Ioctl 设备配置
Open 设备打开
Close 设备关闭
Read 读设备
Write 写设备

2) 文件管理：文件读写和文件控制；

Open 文件打开
Close 文件关闭
Read 读文件
Write 写文件
seek 读写指针定位
Creat 文件创建
Stat 读文件状态
Mount 安装文件系统
chmod 修改文件属性

3) 进程控制：创建、中止、暂停等控制；

Fork 创建进程
Exit 进程自我终止
Wait 阻塞当前进程
Sleep 进程睡眠
Getpid 读父进程标识

4) 进程通信：消息队列、共享存储区、socket等通信渠道的建立、使用和删除；

5) 存储管理：内存的申请和释放；

6) 系统管理：设置和读取时间、读取用户和主机标识等；

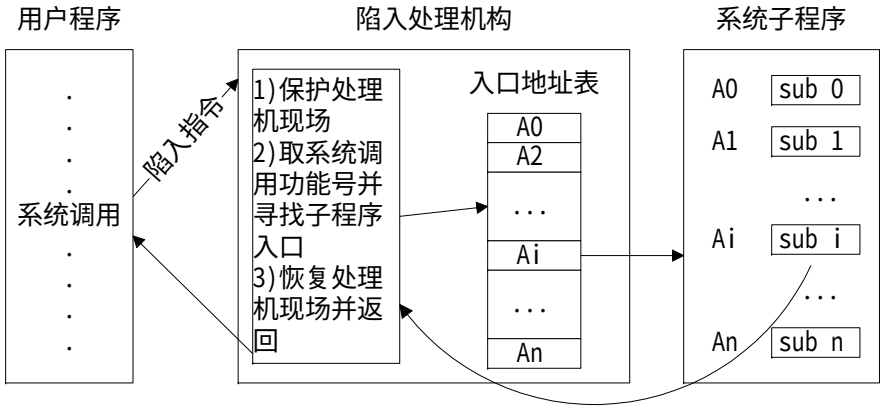
gtime 读取时间
Stime 设置时间
getuid 读取用户标识

2. 通过系统调用接口使用系统命令

- 通过系统调用接口也可使用系统命令。
  - C语言里的system()函数可调用shell来完成命令
  - 如 UNIX系统： system("cp -r doc /tmp")

2.3.2 系统调用的实现过程

实际上系统调用语句本身是硬件提供的（机器指令），但其所调用的功能是操作系统提供的。每种机器的机器指令集中都有一条系统调用指令。



- 设置系统调用号和参数。
  - 调用号作为指令的一部分（如早期UNIX），或装入到特定寄存器里（如：DOS int 21h，AH=调用号。）
  - 参数装入到特定寄存器里，或以寄存器指针指向参数表（内存区域）。
- 执行trap(int)指令：入口的一般性处理，查入口跳转表，跳转到相应功能的过程。
  - 保护CPU现场(将PC与PSW入栈)，改变CPU执行状态（处理机状态字PSW切换，地址空间表切换）
  - 将参数取到核心空间
- 执行操作系统内部代码；
- 执行iret指令：将执行结果装入适当位置（类似于参数带入），恢复CPU现场（以栈顶内容置PSW和PC）。

2.3.3 系统调用举例

凡是与硬件相关、与应用无关的工作，都通过操作系统程序来完成。

1. 利用系统调用向打印机输出5个字符

方法1：调用DOS功能向打印机输出

```
MOVE A,PARA1
MOVE B,PARA2
INT 21H
```

方法2：用OUT指令直接打印

```
L1：MOVE A，I
    IN ADDR1，B
    OR B，BS
    JNC L1
    OUT ADDR2，A
    RET
```

利用系统调用和out指令完成打印功能的比较

- I/O设备的硬件接口，一般由四种寄存器（地址，数据，状态，控制）或其子集组成，任一程序中若要使用I/O设备来输出数据或接受输入，必须通过对这四种寄存器读写的I/O机器指令进行。
- 使用系统调用的程序段则简单许多。程序员不再需要与接口寄存器打交道，只需一个简单的调用即可。

2. 利用系统调用实现硬盘文件内容读写

```
MOVE DX，OFFSETBUFF
MOVE CX，BYTE
MOVE BX，HANDLE
MOVE AH，3FH
INT 21H
```

- 在应用程序中直接利用I/O指令进行硬盘文件内容读写将是十分复杂和困难的。原因为：
  - 磁盘的控制和状态接口寄存器比打印机的复杂的多，而且还有地址接口寄存器，涉及像磁道号，磁面号，扇区号这样复杂的外存物理地址；
  - 通常一个外存存储介质上可以存放多个文件，随着文件的建立、删除和拷贝在不停地变化，每个文件的长度随着其内容的变化而不停地变化，
  - 安全问题；

2.3.4 系统调用与普通过程调用的相同点和不同点

- 相同点
  - 改变指令流程
  - 重复执行和公用
  - 改变指令流程后需要返回原处
- 不同点
  - 系统调用是动态调用，而CALL调用方式是静态调用；
  - 执行状态不同
  - 进入方式不同
  - 与进程调度的关系不同：
  - 嵌套或递归调用

1. 系统调用是动态调用，而CALL调用方式是静态调用；

系统调用是动态调用，程序中不包含被调用代码，好处：
（1）用户程序长度缩短
（2）当OS升级时，调用方不必改变
系统调用方式的调用地址和返回地址都是不固定的，系统调用指令中不包含调用地址，只包含功能号，是按功能号（在可执行目标程序中）调用的。在操作系统内部，由系统调用处理程序通过系统调用分支表（OS的一个数据结构）将功能号转换为相应的指令地址。
系统调用返回指令中不包括返回地址，通过栈保存和弹出返回地址。系统调用返回地址不固定，因为用户程序在不同的地方调用OS。
CALL调用方式是静态调用，被调用代码与调用代码在同一程序之内。CALL调用方式，其调用地址是固定的，包含在调用语句中；返回地址是不固定的（同一子程序可能被不同处多次调用），在程序执行过程中通过栈的实现来保存和弹出返回地址。

2. 执行状态不同

调用和返回经历了不同的系统状态。通常核心和应用程序的代码分别运行在CPU的不同的状态下（系统态/核心态/管态和用户态/目态），所用地址空间也不同——核心的代码可以直接访问应用进程的地址空间，反之不然。

- 状态切换：系统调用、中断、异常
- trap陷入
- 特权指令（访问关键寄存器、停机指令）和I/O敏感指令（中断屏蔽、端口读写）

3. 进入方式不同

利用int或trap指令进行系统调用；利用call或jmp指令进入普通的过程调用；

- CALL指令的内部实现过程：
  - 返回地址压栈（即该CALL指令所在的地址）；
  - 将该CALL指令中所含的地址（即被调用代码所在地址）送入PC
- RET指令的内部实现过程：
  - 从栈顶弹出返回地址送入程序计数器PC

4. 与进程调度的关系不同

采用抢先式调度的系统，在系统调用返回时，要进行重新调度的检查——是否有更高优先级的任务就绪（创建或唤醒）。

5. 嵌套或递归调用

对系统调用，一般不允许在同一个进程中发生嵌套或递归（不同进程可以重入同一个系统调用）。

2.4 图形用户接口 (GUI, GRAPHIC USER INTERFACE)

在命令行方式下，用户与操作系统的交互要求用户记忆命令格式。在图形用户接口方式下，用户可利用鼠标对屏幕上的图标进行操作，完成与操作系统的交互，从而减少记忆内容，方便用户使用。它的技术基础是高分辩显示器和鼠标。

返回

2.4.1 概述

1. 窗口系统(window system)的特点
2. 窗口系统的图形元素及其状态
3. 窗口管理器(window manager)

1. 窗口系统(window system)的特点

- 利用图形元素表示功能：将各种图形元素显示在屏幕上，用户可以通过操纵图形元素（如菜单、图标）来执行相应的功能
- 同屏多窗口与并发进程相对应：屏幕上同时显示多个窗口；一个进程可以对应一个或多个窗口；窗口动态创建、改变、撤销
- 输入方式：鼠标指针点击（或其他定位设备）和键盘输入；通常是即时交互
- 一致的图形元素风格可方便用户学习和使用：如按钮、滚动条
- 优点：操作直观（不必记命令行参数），可与多个进程交互，便于进行多媒体处理－－简而言之：交互的并发性好、传递信息量大

2. 窗口系统的图形元素及其状态

- 窗口(window)：屏幕上的矩形区域（可以通过掩模mask来显示任意形状）
  - 包括：标题条(title bar)、边框(border)、窗口角(corner)、系统菜单框(system menu box)、最大化/最小化按钮(maximize/minimize)、滚动条(scroll bar)等
  - 状态：当前/非当前窗口(active/inactive) －－接受输入，最大化/最小化/恢复原大小(restore)，窗口的前后遮盖－－Z轴，焦点(focus)－－接受键盘输入（而非鼠标）
  - 桌面(desktop)和墙纸(wallpaper)
- 图标(icon)：一个小图象(如32x32或64x64 pixel)，通常供鼠标指针点击。通过不同的图标可以标识不同的对象。如：可执行程序、最小化的窗口、文件－－动画图标
- 鼠标指针(mouse pointer)：鼠标指针通常对应屏幕上的光标(cursor)。
  - 光标在屏幕上只有一个，在不同屏幕位置（上下文）可以呈现不同形状，可以独立于鼠标来直接操纵光标。
  - 鼠标点击：左键/右键/中键(left/right/middle button)，单击(click)/双击(double-click)，拖曳(drag)/拖放(drag-and-drop)；
- 按钮(button)：鼠标点击或按回车键/空格键时执行相应功能，如 menu button；提供单项或多项选择，如radiobutton和checkbutton；当前按钮及其切换。
- 菜单(menu)：临时窗口，菜单条(menu bar)、弹出式菜单、下拉式菜单（上下文相关菜单）
- 对话框(dialog box)：临时窗口，显示提示信息(message)或填写用户设置。

3. 窗口管理器(window manager)：形成统一的使用风格

- 处理窗口的普遍特性，如：窗口的大小、位置（窗口的标题条、边框、控制菜单框）－－窗口中由应用程序管理的部分称为"客户区(client area)"
- 协调各窗口间的相互关系，如：窗口之间的前后遮盖关系，桌面

2.4.2 MS Windows

1. 特点
2. MS Windows结构
3. 基本概念
4. 消息处理：两种方法

1. 特点

- 是OS 的一部分，提供默认的窗口风格（如菜单、对话框）
- 除Windows2000外，窗口应用程序只能在控制台（本地）执行。

2. MS Windows结构

可分成三个部分。

- OS系统服务(KERNEL)：内存管理、程序加载（包括DLL）、任务调度、文件管理
- 用户接口(USER)：窗口和消息管理，以及菜单、控制、对话框、定时器等
- 图形设备接口(GDI, Graphic Device Interface)：管理显示器，为USER与应用程序提供与硬件设备独立的接口

3. 基本概念

- 消息(message)：消息作为窗口的输入，如用户操作、其他窗口或系统发出的请求或通知。
- 窗口过程(window procedure)：消息由各窗口自己的窗口过程来作处理。
  - 窗口过程的调用参数：接收窗口句柄（可以在多个窗口共用一个窗口过程时加以区分）、消息ID（消息的类型）、消息参数（16+32位或32+32位值，整数或指针）
  - 还可以取得消息的发生时间和屏幕坐标

4. 消息处理：两种方法

- 排队消息：放到进程（线程）的FIFO消息队列里。如应用程序发送的消息，系统的鼠标、键盘、定时器、窗口绘制和退出等。排队消息所用的API：
  - 发送消息到消息队列PostMessage；
  - 从消息队列读取消息GetMessage, PeekMessage；
  - 分发一个消息到相应窗口DispatchMessage；
- 非排队消息：直接发送到指定窗口的窗口过程。非排队消息所用的API：
  - SendMessage，直到接收方窗口过程处理完才返回
  - 接收方正处于GetMessage，而接受并处理SendMessage送来的消息之后，仍处于GetMessage。为防止死锁，一般要：
    if (InSendMessage()) ReplyMessage(TRUE);

- 消息循环：不断移出消息，并加以处理。用户不作处理的消息，应传给默认窗口过程加以处理：DefWindowProc()
- WM_PAINT消息：通知窗口客户区中的某部分(region)已被改变，应用程序应该对其重新绘制。总是被排在线程消息队列的最后，并且多个WM_PAINT消息中的region会自动被合并为一个。

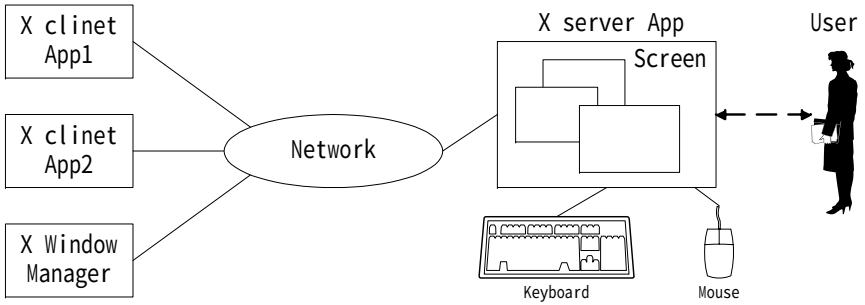
2.4.3 X Window

1. 特点
2. X Window的结构
3. 基本概念
4. X lib和工具箱(toolkit)
5. X Window的使用

1. 特点

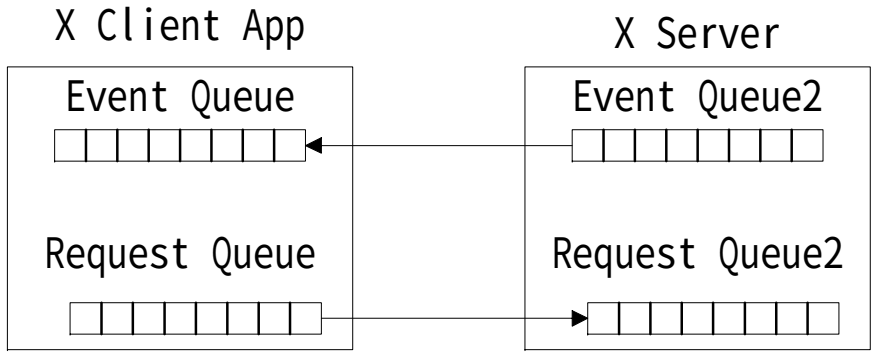
- Client-Server结构：X client和server都是应用程序
  - 一个server可以为不同计算机上的多个client提供服务，server对有关I/O设备具有访问权
  - 一个client也可以连接不同计算机上的多个server
- 显示设备独立性和支持多种网络协议：(在TCP/IP, DECnet之上的X protocol) －－X不是计算机操作系统的一部分：其他大部分窗口系统都是以OS核心为基础的，如：MS Windows, Macintosh, SunView
- 支持自由风格(policy free)：只提供机制不提供风格；包括窗口管理器、菜单、按钮、滚动条等的管理和操作，如：Motif, Open Look, Tcl/Tk等

2. X Window的结构



X Window的结构

- 各client、窗口管理器和X server可以在一台或多台计算机上
- 用户对server所控制设备的操作都使server发送事件(event)，如：鼠标移动、点击，键盘输入；
- 事件是一个数据块，内容包括：事件类型、相关的窗口等
- client对来自server的事件进行处理，然后向server发送请求(request)，如：窗口操作、显示图形或正文
- X protocol使用的传送信道：异步字节流。server不断发送事件而由client中的先进先出FIFO队列加以缓冲，client不断发送请求而由server中的FIFO队列加以缓冲。



X Window的队列

3. 基本概念

- 屏幕(screen)和显示器(display)：屏幕指显示设备硬件，而显示器指X server（及其控制的键盘、鼠标和屏幕）。X允许一个显示器控制多个屏幕（即显示设备硬件）。
- 资源(resource)：是X window使用的窗口、位图(bitmap)、字体(font)、调色板(color map)和其他数据结构的总称。
  - 资源在X server中创建和存储，按照client的请求来加以管理，而client退出时撤销对应的资源。
  - 通过资源标识(resource ID，整数类型)来标识不同的资源（包括系统资源）－－资源的自身不体现风格
- 窗口树：层次，子窗口只显示出被父窗口剪切(clip)的区域

4. X lib和工具箱(toolkit)

- X lib函数库，将接收到的X protocol转换为事件，并将请求转换为发送X protocol。其代码链接在X client中。如：
  - 打开显示器：XOpenDisplay(char *display_name)－－与X server建立连接
  - 取得X connection的文件描述符：XConnectionNumber()

- X toolkit：提供各种构件的例程库，便于用户使用，体现不同的风格。如：Motif, Open Look, Tcl/Tk－－构件的数据结构存储在X client
  - Xt Intrinsics：对构件的管理和操作，如：建立和撤销widget、管理资源（包括widget的初始值）、处理事件并调用相应的处理程序（回调过程call-back）
  - widget set：构件集合－－构件类(widget class)和构件实例(widget instance)
- 构件(widget)：由多个资源（最终是resource ID）复合而成，具有特定外观和功能的部件，如：按钮、菜单和滚动条－－如同建筑上的预制件。本质是：对某些事件给出了默认响应（通常是改变外观），并可以挂接回调过程来进行用户定义的处理，方便用户使用。如Tcl/Tk：button .app.button2 -image icon2 -command "incr x0"
- 各个widget也构成widget tree，最顶层是top-level widget（在toolkit初始化时建立）
  - editres命令：查看widget tree，并对X 应用进程中的资源进行动态修改

5. X Window的使用

远程登录，X server机为hostS（本地用户），client机为hostC（远地）

1）本地，起动X server

2）本地，允许访问X server：xhost +hostC

3）本地，发起远程登录：telnet hostC并输入用户名和口令

4）远地，运行X终端程序：/usr/openwin/bin/xterm -display hostS:0.0 & 或：setenv DISPLAY hostS:0.0 ; /usr/openwin/bin/xterm &

5）本地，在 xterm窗口内输入命令行，可以起动其他X client程序，如：xclock

2.4.4事件驱动模式(event-driven)

1. 面临的问题
2. 事件驱动模式
3. 分发驱动模式
4. 举例

1. 面临的问题

几个处理分支在各自条件下，间歇地、重复地执行，次序不确定。如：

```
While  (!done)  {
  if  (C1)  P1;          /* C1：管道1可读                */
  if  (C2)  P2;          /* C2：管道2可写                */
  if  (C3)  P3;          /* C3：定时器的时间到           */
}
```

- 若C1, C2, C3在判断时是阻塞式(blocking)，则一个分支在阻塞等待时，使得其他两个即使条件具备也不被立即执行
- 若C1, C2, C3在判断时是非阻塞式(non-blocking)，则C1, C2, C3不间歇地反复查询，CPU开销大

2.　事件驱动模式

```
While  (!done)  {
  NextEvent(Event);
        /* 获取下一个事件，若无事件则等待直到有事件*/
  switch  (Event.Type)  {
          case  C1:       P1;
          case  C2:       P2;
          case  C3:       P3;
  }
}
```

- 将多路判断和等待汇集为一路
  - UNIX系统中相应的系统调用或库函数为：select()（Solaris 2.3库函数，Linux系统调用）, poll()（Solaris 2.3系统调用）
- 特点：
  - 节省运行CPU开销——无事件时等待而不是反复查询，有事件时才处理
  - 当处理分支较多时，switch语句变得复杂和不易掌握

3. 分发驱动模式

```
CreateObject(object1, P1);
CreateObject(object2, P2);
CreateObject(object3, P3);
While  (!done)  {
  NextEvent(Event);
        /* 获取下一个事件，若无事件则等待直到有事件*/
  Dispatch(Event);      /* 分发事件给相应的回调过程*/
}
```

- 特点：
  - 通过对象管理机构（如X toolkit）创建多个对象或构件，输入事件通过对象各自的回调过程(call-back procedure)来处理
  - 通过对象管理机构，将输入事件分发给适当的元素－－无须了解分发过程的实现

4. 举例

MS Windows的消息处理：

```
WinMain(...) {
  CreateWindow("MainWndClass", "SampleName", ...);
/* 窗口过程的人口指针包括在WNDCLASS结构中，通过
RegisterClass注册窗口类*/
  while (GetMessage(&msg, ...)) {/* 收到 WM_QUIT消息时，返
回值为NULL */
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
}
```

X Window的事件处理：Xt Intrinsics

```
main(..) {
  topWidget = XtAppInitialize(...);
  XtRealizeWidget(...);
  while (1) {  /* 这里的while循环等价于XtAppMainLoop 函数 */
    XtAppNextEvent(..., &event);
    XtDispatchEvent(&event);
  }
}
```

小结

- 作业组织和控制：脱机、联机（命令行）
- 系统调用：与普通过程调用的区别、与高级语言函数库的区别、实现过程
- 作业管理举例：DOS、UNIX(shell)
- 图形用户接口(GUI)：概述、X Window、MS Windows、事件驱动模式

作业

1） 在Windows2000下用批处理命令实现UNIX的grep命令的主要功能。

2） 在Linux下用Bash实现Windows的dir命令的主要功能。