西电学堂（有开试卷）

大作业？ 15%

部分积分.

中断

异常

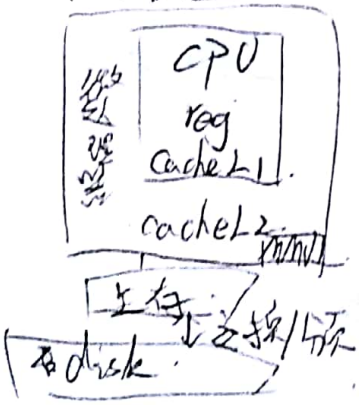
系统调用 win32 posix

函数调用：一个栈空间

系统调用，多个栈空间

(CPU) 用户态，内核态

内存管理



物理 → 主存

逻辑地址空间 → 程序看到的地址

分段 —— 保护

访问相同 共享

更多 虚拟化

编译 汇编 链接，载入

(程序重定位)

MMU LA — PA 映射表

程序访问空间表（

内存碎片

分区的动态分配

分配策略：first fit

best fit

worst

MMU 硬件管理

压缩式

交换式

非连续内存分配

硬件方法

① 分段：连续的逻辑地址分散到物理内地址中

段表时延 段表：[逻辑段，物理段 长度，

② 分页

大小固定。 逻辑(页) 物理帧

帧号，帧偏移 页号可大于帧号，

页号 → 页表 → 帧号

页表大小，

1 缓存 TLB 常用页表项

2 间接 多级页表（若不存在 则不需存储二级页表

反向页表：

1 页寄存器

2 关联存储器，

3 哈希 PID 与逻辑地址

虚拟内存：

1. 覆盖 一个程序内
2. 交换（整体换出 多个程序间
3. 虚存（在分页基础上，交换）

程序的局部性（时间+空间）

缺页异常 ← 页表中 resident 位 = 0

↘驻留位

请求调页，页面置换

保护位，读写，执行
修改位，是否被写过（换出前写回）
访问位，是否被访问

1. 数据
2. 代码
3. 动DLL

} backing store

4 运行中生的数据 swap 分区

页面锁定（常驻内存）

最优页面置换算法：

根据将来是否执行（不现实） 可用于评估

离访问时间最远

先进先出 FIFO

维护表记录驻入时间（链表

最近最久未使用 LRU

最久未被使用 （开销较大） 链表 栈

LRU

最久 { 时钟页面置换算法.

为何位，是否被访问过.

~~OS 将其定期清0~~

指针指向上一个页的位置.

若被访问过，则清0，移向下一项，查找 =0的页

二次机会法，

查看都访问

used dirty

0 0 → replace

0 1 → 0 0

1 0 → 0 0

1 1 → 0 1 二次机会

最不常用：LFU （不太合理，若一开始访问很多之后不访问）

Belady，增加物理页，缺页率反而增加（FIFO中）

算法+页面代码的局部性

全局页面置换算法，

程序分配物理页帧的数目.不固定

不同运行阶段，

工作集：一个进程当前正在使用的逻辑页面集合

W(t, Δ)

工作集大，局部性好.

常驻集：当前时刻，进程实际驻留在内存中.

工作集不一定在内存中

工作集与常驻集相差越小越好.

(常驻集) 工作集页置换算法，

页面不属于工作集时移出，(在缺页空闲页面之前，提前将页面换出.

固定窗口.

缺页率算法 PFF 动态分配物理块

产生缺页中断的时间间隔.

大，(未访问的移除未在时间内访问的页.

小，(当缺页加入工作集.

抖动问题

缺页平均时间 = 缺页服务时间

进程：程序的动态执行过程.

并发，(单核多进程) 并行(多核多进程)

进程控制块 PCB.

进程标识信息.

处理器状态信息.

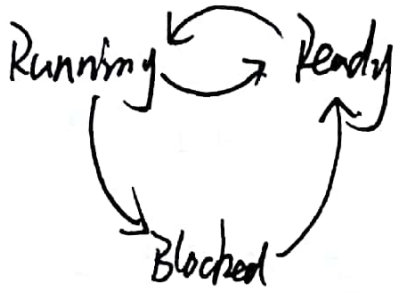
进程控制信息.

进程状态

生命期：

进程执行发起 ↓ 被动

创建 运行 等待(阻塞) 唤醒 结束.



~~Blocked~~

进程挂起：没有占用内存空间

~~阻塞挂起~~，就绪挂起.

解挂.

不同状态不同队列

根据事件不同建多个阻塞队列

线程.

并发执行. 共享地址空间.

进程用来管理资源

线程用来执行 TCB

~~线程~~=

一个线程崩溃，其他有崩溃

~~性能上线程~~

安全 上进程

性能 上线程

进程是资源分配的单位.

线程是CPU调度单位.

用户线程，由线程管理库实现（OS不参与）

~~内核线程~~：OS不调度，~~只调度进~~

由库调度

内核线程，TCB在内核中，OS参与调度

~~轻量级进程~~，

上下文切换（寄存器信息的保存与读取）

~~pid = 0~~
~~(创建一个新进程)~~
~~堆栈段覆盖~~

exec() 加载程序取代当前运行的进程（执行新的程序）

fork() 创建子程序

pid > 0 执行空间在父进程中

fork() 开销较大

copy on write 父→子

父进程等待子进程结束，

Zombie

wait() 父对子进程资源的回收

僵尸，子exit()执行完毕 父wait未执行完毕

fork() 子进程与父进程使用相同代码段

子进程复制父进程的堆栈段和数据段

调度原则

CPU突发与I/O交替.

调度算法

先来先服务 FCFS　　周转时间

短任务优先

最高响应比优先　　$R=(W+S)/S$　　R越大 机会越大

轮询　　时间片　　等待时间大 切换多

多级队列　前台. RR　后台. FCFS

多级反馈队列　　CPU密集 优先级低

I/O密集　　高

FFS 公平共享　多用户

实时调度. 工业控制机院　(规定时间内完成)

强硬时. 必须完成　弱实时　尽量

任务.

静态优先级　动态优先级

选举算法　　最早期限调度

同期任个　　Deadline近个

多处理器调度

优先级反转：

优先级 继承

优先级天花板，当前优先级

同步：

不可剥夺，不可重现

基于监视，加速，模块化

原子操作

临界区，为何全局变量的代码，

互斥，只有一个同临界区，

死锁：互相等待

饥饿：

Lock.acquire() 在锁被释放之前等待，有时

Lock

无止等待

临界区保护：

屏蔽 硬件中断（无效不好）

基于软件：（Dekker 二个线程

```
flag[i] = True.
turn = j;
while (flag[j] && turn == j);
    ;
flag[i] = false
```

基于硬件操作

Test-and-set

1. 从内存读值
2. 比较寄存1.
3. 内存修改为1
返回读值.

交换.

交换内存中的两个值

```
int Lock = 0;
int key
do {
    key = 1;
    while (key == 1) exchange(lock, key);
    ...
    lock = 0
```

高优先级先上等，低优先级得锁

```
class Lock {
    int value = 0
    Lock::Acquire() {
        while(test-and-set(value));
    }
    Lock::Release() {
        value;
    }
```

原子操作支持.

spin.

忙等

可以特别等待队列

当一进程解锁后，唤醒其它等待队列（但上下文切换慢）

信号量：　　读操作可以进行

Sem　整数

V 增加

P 减少

互斥.Clock，二进制

```
mutex = new Semaphore(1);
mutex->P();
...
mutex->V();
```

调度约束：(同步)

```
condition = new Semphore(0);
Thread A                    Thread B
...
condition->P()              ...
                            condition->V();
```

condition<0 等待

P 等待　　V会出信号

消费者问题：(同步+互斥

多个生产者 互斥

buffer 满 (同步互斥)

buffer 满

二进制 信号量 mutex　1

一般信号量 empty buffer　n.

full　0

填　　取

mutex　mutex

```
class : Semaphore {
    int sem
    WaitQueue q;
}
```

实现

```
Semaphore :: P(){
    Sem--;
    if(sem<0){
      Add to q
      block(p)
    }
```

```
Semaphore::V(){
    Sem++;
    if(sem<=0){
      Remove t from q.
      Wakepp(t)
    }
}
```

互斥、条件同步，

锁 lock.

管程 monitor.

条件变量 { wait() 释放 放锁()
signal() 唤醒

```
class Condition(){
    int numWaiting = 0;
    WaitQueue q;
}
```

```
Condition::Wait(lock)
    numWaiting++;
    Add this to q;
    release(lock);
    schedule();
    require(lock);
```

```
Condition::Signal(lock){
    if(numWaiting>0){
      remove a thread from q
      wakeup(t)
      numWaiting--;
    }
```

解决 同步互斥问题：信号量，管程.

读者，写者问题.

Rcount 0

CountMutex | WriteMutex |

读者优先

Writer

```
sem_wait(writeMutex)
    write.
sem_post(---)
```

Reader

```
Sem_wait(RcountMutex)
if(Rcount ==0)
    Sem_wait(W-M-)
++Rcount;
sem_post(CM).
Read;
sem_wait(CM)
--Rcount;
if(Rcount ==0)
    sem_post(W---M)
Sem_post(CM).
```

写者优先

管程实现:

```
AR =0      # 读中 读文.
AW =0
WR =0      # 等待
WW =0

Condition okToRead;
Condition okToWrite;
Lock lck;
```

StartRead::

```
lck::Acquire();
while((AW+WW)>0
    WR++
    okToRead.wait(lck)
    WR--
};
AR++;
lock.Release();
```

DoneRead::

```
lock.Acquire()
AR--
if(AR==0&&WW>0){
    okToWrite.signal;
```

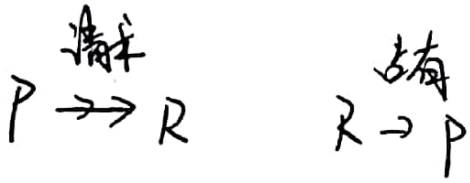
lock.Rel

哲学家就餐.

fork[5] = 1

死锁:

资源: 使用优先: request use release



死锁: 资源与图P之间有环.

有环但不一定产生死锁.

条件:

必要
- 互斥.
- 持有并等待
- 无抢占.
- 循环等待

处理