存储器管理

前言

- 现在的计算机大多都是用虚拟内存管理技术，进程不是一次性的进内存，那段执行，哪段需要进内存，目的是为了提高程序的并行性
- 虚存管理技术都需要对应的数据结构进行管理
- 存储器管理值得是对用户区的内存的管理

概述

存储器管理的目的是要解决我们的地址转换和重定位问题

1. 存储器管理功能
    - 存储器分配 : 多程序和多进程共享内存的问题
    - 地址转换和重定位 : 地址变换方法和对应的地址变换机构
    - 存储器保护 : 防止故障程序破坏OS和其它信息
    - 存储器扩充 : 虚拟内存管理技术
    - 存储器共享 : 并发进程共享程序和数据
2. 地址空间
    - 符号名字空间 : 程序中的各种符号名的集合限定的空间叫做符号名字空间
    - 逻辑地址空间 : 经编译连接后的目标代码所限定的空间
3. 存储空间
    - 物理存储器中的全部的物理单元的集合限定的空间
    - 地址
        - 绝对地址,实地址,物理地址
        - 地址空间 : 和虚存空间挂钩，地址空间装入到存储空间才可以运行
4. 地址重定位
    - 原因
        - 逻辑地址和物理地址不匹配
        - 运行需要物理地址而不是逻辑地址

- 程序的链接
    - 静态链接 : 装入内存之前，将目标程序和库例程链接
    - 动态链接
        - 装入时 : 边装入主存边链接
        - 运行时 : 程序运行时再链接(便于模块的共享，空间开销小，系统效率高)
- 定义

  将程序中的逻辑地址转换成存储空间的物理地址的工作叫做地址重定位
- 实现

  MMU : 硬件，存储器管理单元
- 方式
    - 静态重定位
        - 装入程序的时候，装入程序将用户程序中的指令和数据的地址全部转换成存储空间的绝对地址
        - 无需MMU，容易实现
        - 占有连续的存储区，程序在主存中不允许移动，不灵活，不适合多道程序系统
    - 动态重定位
        - 适合多道批处理和分时系统

        - 设置重定位寄存器,记录转入程序的起始地址
        - 主存利用充分，程序在主存中可以移动
        - 不必占用连续的存储区域，每一块设置一个重定位寄存器即可(PCB记录信息)
        - 便于多用户共享程序和数据(子程序共享)

5. 存储器保护
    - 存储器分配
        - 操作系统占有区
        - 多用户进程占有区
    - 保护
        - 越界检查 : 进程运行产生的所有的访问地址都要检查，防止越界
        - 操作方式检查 : 防止误操作，[r,w,x]
6. 存储器共享
    - 共享存储器可以是数据也可以是程序
    - 被共享的程序不可修改(可重入程序，纯代码)

单用户单道程序存储器分配

单一连续分配 :

- 主存只有一个用户作业，其他都是操作系统的程序，用户进程独占用户区和所有的系统资源
- 系统资源利用率底下

多用户多道程序的存储器分配(分区分配)

- 思想

  主存划分成若干个连续区域，每一个用户进程占有一个区域
- 类型
    - 固定式分区
        - 主存用户区划分成大小不同的分区
        - 选择一个合适的最小空闲分区
        - 没有合适的分区阻塞等待
        - 排队可能不合理，出现很大的浪费
            - 修改措施，主存中只保留一个等待队列，只要有一个分区是空闲的，队列中的程序可以装入运行(不必过分的要求最小空闲区)
        - 分区说明表 : 描述主存各分区的使用情况
        - 简单易于实现
        - 多进程可以共享主存(存储器多进程)，但是主存利用不充分，进程的大小和分区的大小不一定完全匹配
    - 可变式分区
        - 进程要求运行，从空闲的存储空间中划分出大小正好等于进程大小的存储区分配给对应的进程
        - 提高使用存储器的效率
        - 管理分区 :
            - 进程的数目，大小，占用位置管理麻烦，分配和释放工作麻烦
            - 管理分区的数据结构
                - 分区说明表
                    - 已分配表 : 记录分配给进程的分区情况
                    - 未分配表 : 记录主存的空闲区情况
                    - 分配主存 :

在未分配表中找到一个大小足够的空闲区，一分为二

                - 简单直管，但是表可能会浪费
            - 空闲区链
                - 已分配的内存区域放在进程的PCB中
                - 空闲区表使用空闲区链代替
                - 将空闲区的表格信息放在每个分区(不论是否空闲，但是空闲区链中的都是空闲的)的首字或者尾字中，双向链表
                    - 状态位 : 表示分区是否是空闲的
                    - 分区大小 : 字节或者字
                    - 单向指针 : 前项指针或者后向指针
                - 分配主存的方式一致
        - 分配算法
            - 首次适应法 : 空闲区起始地址从小到大，易于合并
            - 最佳适应法
                - 找到满足及产能需要并且最小的空闲区
                - 产生碎片，效率低
                - 改进后将所有的分区从小到大排队，维护开销增大，难以合并
            - 最坏适应法
                - 满足进程需求并且最大的空闲区
                - 改进后所有分区从大到小排队，(大空间进程可能无法执行)但是维护开销增大，难以合并

- 地址重定位和存储器保护
    - 固定式分区
        - 采用静态重定位
        - 两个界限寄存器保护存储器(上下界)
    - 可变式分区
        - 使用动态重定位(程序可以移动)
        - 重定位寄存器 + 限长寄存器
    - 使用MMU实现
- 分区管理的分析
    - 优点
        - 多道程序共享主存
        - 设计和保护手段简单
    - 缺点
        - 主存利用不充分(碎片)
        - 没有实现主存扩容问题(主存的地址空间大于存储空间无法运行)

覆盖和交换技术

前言

- 前面的分区管理技术没有解决存储器的扩容问题，我们可以使用覆盖和交换进行逻辑扩容(外存交换区)
- 该技术解决了大进程和小内存的矛盾，一定程度的逻辑扩充

覆盖技术

1. 定义
    - 同一段主存去可以被一个或者多个作业的不同进程覆盖使用
    - 进程使用功能的时候只会使用功能程序中的几段程序，我们可以将不会共用的程序图片那个用一个主存区

- 相互覆盖的程序叫做覆盖段
- 共享的主存区叫做覆盖区，最大覆盖段决定
- 提供覆盖管理控制程序
    - 覆盖结构是由程序员给出的，一般都是用在系统内部的程序中
        - 操作系统主要部分常驻内存，占有固定区域
        - 磁盘贮存需要调入
- 指标不治本

交换技术

1. 根据需要，将主存中暂时不需要的进程信息移至外存交换区，并将外存中的某一个进程换入，并使其运行
2. 目的
    1. 解决主存容量不够大的问题
    2. 保证分时用户的合理响应
3. 外存
    1. 文件区
    2. 交换区
        1. 进程PCB
        2. 核心栈
        3. 程序
4. 时机
    1. 进程用完分配的时间片，等待输入输出
    2. 资源需求得不到满足
5. 要求
    1. 尽量减少交换次数，代价高，解决方案是和多道程序结合起来使用
    2. 尽量减少交换的信息量，解决方案是将进程的副本保存在外存，更新交换
6. 实现
    1. 打破一个进程在主存中一直运行到结束的期限
    2. 进程的大小仍然受到我们的存储器的影响

页式存储管理(共享不方便,一个页中未必存放的就是完整的程序或者数据)

1. 思想
    - 为了克服分区管理的碎片问题
    - 进程可以不占用连续的存储空间
2. 实现原理
    - 主存分页框，页框的大小随实际情况而定
    - 进程的地址空间也划分成大小同样的页
    - 页框和页实现离散分配
    - 地址需要重新定义
        - 页号
        - 页内地址
        - MMU负责转换地址
    - 页表
        - 一个进程一个页表，记录进程的逻辑页和内存块的映射关系
        - 页表存放在主存，页表起始地址和页表长度在PCB中
        - 页表中的信息
            - 逻辑也和物理页的对应关系，索引

        - 页表的操作方式
    - 每一个处理机都有一个控制寄存器，记录当前正在运行的进程的页表的其实地址和页表长度
3. 页式动态地址变换
    1. 检查页号是否越界(超过页表长度，处理机控制寄存器)
    2. 和页表首地址做线性加法获得物理页号
    3. 物理页号和页内地址拼接
4. 分析
    1. 优点 : 不浪费内存，很大程度解决了内存碎片问题
    2. 缺点 :
        1. MMU复杂
        2. 耗时 : 两次内存访问，页内地址转换一次，访问实际地址一次
5. 优化
    - 引入快表(高速查找寄存器组)
        1. 存储单元可以被同时读取，速度快
        2. 存取速度高于主存，造价高，少量使用
        3. 快表的构造
            1. 页号
            2. 块号
            3. 访问位 : 该项页最近是否被访问过，作为被覆盖的依据
            4. 状态位 : 寄存器是否被占用
        4. 使用
            1. 恢复快表
            2. 快表和页表同时查找
            3. 快表中没有的我们从也表中取出来写入快表(空闲快表项，或者未访问的快表项)
6. 主存分配和主存的保护
    1. 主存分配
        1. 页表 : 虚存转化成主存的器件
        2. PCB : 保存页表信息并保证运行时将该信息写入处理机的控制寄存器
        3. 存储空间使用情况表
            1. 存储分块表
                - 当前空闲块数(快速检查当前的空闲的数目够不够)
                - 记录每一个块是空闲的还是被占用的
                - 空闲块链接在一起
            2. 位示图
                - 0空闲1占用
                - 记录空闲块个数
    2. 存储器保护

        也表中记录了对每个块的操作权限[r,w,x]

段式存储器

1. 思想
    - 解决用户进程的程序段共享问题
2. 实现原理
    1. 进程的地址空间按照进程的逻辑关系划分成段，每个段存在段长和段名
    2. 独立段
        1. 代码段

        2. 数据段
        3. 堆栈段
    3. 分配内存使用段为单位，为进程的每一个分段传建一个连续的主存区，各段之间可以不连续
    4. 段表
        1. 进程的分段的主存的起始地址
        2. 段长
        3. 段操作权限
    5. 处理机使用控制寄存器，PCB对段也有记录
    6. 地址 : 段号 + 段内地址
    7. 别的进程可以通过段号共享段内程序
3. 段式动态地址变换
    - 基本同页式(也可以考虑使用快表优化两次访存的缺陷)
    - 但是对应保护方面有补充
        - 段号合法性 : 段表长度的保护
        - 段内地址合法性 : 段长的保护
        - 操作合法性[r,w,x]
4. 存储器分配 : 可变式分区的分配策略，但是是以段为单位
5. 段页的比较
    1. 段对用户可见，页对用户透明
    2. 段大小不固定，页大小固定
    3. 段也可产生碎片，页式消除碎片
    4. 段式易于动态链接和共享，页式不易实现(静态链接)
    5. 段使用二维地址，页使用一维地址
    6. 段可以动态扩充

虚存管理

1. 虚存
    - 实存管理技术中进程的地址空间必须全部装入主存，但是虚存不必要
    - 为用户构建的巨大的地址空间
    - 用户的有效寻址范围和主存大小无关
    - 扩充主存
2. 程序访问的局部性原理

    根据程序的局部性原理，进程执行没有必要将所有的信息装入主存
    - 时间局部性 : 循环重复执行
    - 空间局部性 : 分支只执行部分程序
3. 技术支持
    1. 大CPU地址结构
    2. 多级存储结构
        - 主存－外存(大外存，合适内存)
    3. MMU地址转换机构 : 实现虚拟地址到实地址的转换
4. 虚拟页式管理
    - 页式管理 + 交换技术
    - 进程添加外页表，记录在辅助存储器中的页表的地址
    - 内容
        - 取页 : windows预调页，请求调页
        - 置页
        - 置换

- 页表增加有效位，记录页是否在主存中，调入不在页表中的页引发引发中断
- 页表增加修改位，记录页调入主存后是否被修改，淘汰一个修改的页需要先写会修改信息
- 页表增加访问位,记录是否曾经被访问过
- 页表增加保护位，页的访问属性
- 调入新页之后需要修改页表和内存分配表

5. 页面淘汰算法
    1. 抖动

       因为淘汰算法的不合理，导致系统频繁的换入换出的现象叫做"抖动"，降低了系统的处理效率
    2. 假定

       进程分配的主存块数固定不变(工作集数目不变)，采用局部淘汰算法，只考虑进程内部的淘汰
    3. 算法

       寻多的淘汰算法都尽可能的记录我们的进程的工作集，减少缺页中断的次数
        - 最佳置换算法 OPT
            1. 选择不再访问或者很长　　时间之后才会访问的页面淘汰
            2. 算法不显示，我们无法提前知道一个页面是否在之后需要，但可以用该标注去衡量最好的效果
        - 先进先出淘汰算法 FIFO
            1. 淘汰在主存主存时间最长的一页
            2. 页表的长度是当前运行的进程分配的主存的块数
            3. 设置一个指针指向当前的最早进入的页
            4. 开销较小，只用移动指针即可
            5. 算法效率不高，因为在主存中驻留时间最长的页面不一定是很长时间以后才会被访问的页面
            6. 容易出现belady异常(页表的数变多的时候的缺页率反而提升)
        - 最近最少使用淘汰算法 LRU
            1. 程序的局部性原理考虑 : 淘汰在最近时间里最少使用的一页
            2. 进程工作集 : 进程在一段时间内集中访问的的固定子集是进程的工作集，在换入换出I/O的时候，一次性交换工作集可以减少缺页中断
            3. 堆栈的使用方法 : 将当前执行的页面(工作集内 / 工作集外)放入堆栈的开头，使用双向链表实现
            4. 执行的开销大，不会出现belady异常
        - 时钟页面置换算法 SA 现在常用该方法
            1. 在每一个也标准功能增加引用位(访问位，0未引用)，只是页表最近是否被引用，将页表的所有项放在循环链中
            2. 系统设置指针指向最早进入主存的页面
            3. 指针指向的引用位0淘汰并置新的引用位1,指针走向下一位 ; 否则清除引用位0指针走下一个单元，重复直到找到0引用位

6. 虚拟页式管理的问题
    - 交换区的管理

      交换区可以看作是主存扩充的方式
        - 文件系统的大文件表示
        - 独立的磁盘分区
    - 页面尺寸
        - 尺寸大，碎片浪费多
        - 尺寸小,程序需要的页就多，页表就会变大
    - 页共享
        1. 为了节约主存和查询效率，用专门的数据结构记录共享页
        2. 一般只有只读页可以共享
        3. 将共享页锁在主存中，页表中增加引用计数，当计数是0的时候允许释放页表空间

- 多级页表结构
    1. 页表本身非常的庞大，不可能为页表分配连续的存储空间，采用多级页表的方式实现
    2. 可以简单的理解成是页表的页表
    3. 简单的32位机举例
        1. 页内索引12 - 4KB
        2. 页表索引12 - 4KB
        3. 页目录索引8 - 256
    4. 减少大量的页表建立的空间消耗,推迟到访问页的时候才建立页表
- 写时复制技术
    1. 节约主存的优化技术，提高了主存的利用率
    2. 当多个进程都在读的时候共享页，有一个集成想要写的话，将页拷贝到对应的进程的页表中，其他的读进程依然更能够像之前的页
    3. 创建子进程时，子进程赋值父进程的地址空间采用该技术

7. 虚拟段式管理
    1. 进程调度的时候只将我们的需要的端调入内存，以段为单位进行中断
    2. 段表内容
        1. 状态位
        2. 修改位
        3. 访问位 : 最近是否访问过
        4. 动态增长位 : 段是否可以动态增长，对段的越界检查更加的严格(中断麻烦)
    3. 段的动态链接和装入 P95
        - 间接编制字 : 硬件指令的间接地址中包含的字
        - 连接中断位L : 是否进行动态链接的标志
        - 原则
            1. 若该段的指令是访问本段地址，则编译成直接型指令
            2. 若访问外段地址，则编译成间接型指令，并形成间接字，在间接字中存放要链接段的段名和段内地址
    4. 段共享

       主存中保存共享段段表
8. 段页式管理
    1. 结合优点，避免缺点
    2. 程序的逻辑结构按段划分，每段按页划分

总结

1. 存储器管理的功能。名字空间、地址空间、存储空间、逻辑地址、物理地址?
    1. 功能
        - 存储器保护
        - 存储器共享
        - 存储器扩充
        - 地址转换和重定位
        - 存储器分配
    2. 名字空间 : 程序中的各种符号名的集合限定的空间叫做符号名字空间
    3. 地址空间 : ?????
    4. 存储空间 : 物理存储器中的全部的存储单元的集合限定的空间
    5. 逻辑地址 :目标代码限定的空间
    6. 物理地址 : 存储器中每个存储单元的实际地址

2. 什么是地址重定位?分为哪两种?各是依据什么和什么时候实现的?试比较它们的优缺点
    1. 地址重定位

       将程序中的逻辑地址转换成存储空间的物理地址的工作叫做地址重定位
    2. 分类
        1. 静态重定位
            1. 装入内存前
            2. 依据转换后的地址
            3. 不需要硬件支持，但是不可移动程序
        2. 动态重定位
            1. 装入时或者运行时
            2. 依据控制寄存器
            3. 可以移动程序，但是需要地址转换单元支持，主存利用充分
3. 内存划分为两大部分：用户空间和操作系统空间。存储器管理是针对用户空间进行管理的
4. 存储保护的目的是什么?对各种存储管理方案实现存储保护时，硬件和软件各需做什么工作?
    1. 目的

       防止故障程序破坏操作系统和存储器内部的信息
    2. 工作内容
        1. 地址越界检查(硬件?????)
        2. 操作方式检查
5. 试述可变式分区管理空闲区的方法及存储区的保护方式。覆盖与交换有什么特点？
    1. 可变式分区
        - 使用空闲区链
        - 重定位寄存器 + 限长寄存器
    2. 覆盖和交换的特点
        - 覆盖 ：逻辑上扩充了主存，打破了进程全部信息进入主存运行的限制
        - 交换 : 打破了一个程序一旦进入主存就一直运行结束的限制
6. 页表的作用是什么?简述页式管理的地址变换过程。能利用页表实现逻辑地址转换成物理地址。管理内存的数据结构有哪些？
    1. 页表的作用

       记录逻辑也和主存块的映射关系
    2. 变换过程
    3. 管理内存的数据结构
        1. 页表
        2. PCB
        3. 存储分块表
7. 什么是页式存储器的内零头?它与页的大小有什么关系?可变式分区管理产生什么样的零头（碎片）？？？？？
    1. 最后的页半空
    2. 页大碎片大，页小碎片小
    3. 最佳适配法可能会产生没有进程可以容纳进去的小碎片，导致无法使用
8. 段式存储器管理与页式管理的主要区别是什么?

   页式管理是以页为单位，段以段为单位
9. 什么是虚拟存储器。虚拟存储器的容量能大于主存容量加辅存容量之和吗?
    1. 系统为了满足应用对存储器容量的巨大需求而构造的一个非常大的地址空间。
    2. 不可以
10. 实现请求页式管理，需要对页表进行修改，一般要增加状态位、修改位、访问位。试说明它们的作用
    1. 有效位 : 标记是否存在于主存中

2. 修改位 : 是否修改过
3. 访问位 : 近期是否访问过

11. 产生缺页中断时，系统应做哪些工作?
12. 什么是程序的局部性原理？什么叫系统抖动?工作集模型如何防止系统抖动?
    1. 局部性原理
        - 时间局部性
        - 空间局部性
    2. 系统抖动 :频繁地调入调出，降低了系统的处理效率
    3. 工作级模型防止系统抖动 : ???????好一点的页面置换算法???????
13. 多级页表的概念，多级页表中页表建立的时机。写时复制技术的概念
    1. 多级页表

       页表不在占用连续的的主存空间
    2. 页表建立的时机

       推迟到访问页的时候才建立页表，减少大量的页表建立的空间消耗
    3. 写时复制

       当多个进程都在读的时候共享页，有一个进程想要写的话，将页拷贝到对应的进程的页表中，其他的读进程依然更能够像之前的页
14. 页的共享问题。需要一个专门数据结构来记录进程间共享页