实验6 内存管理

1. 实验目的
2. 进一步加深存储管理方法的理解；
3. 掌握内存管理中页面置换调度算法。
4. 预备知识
5. 函数的栈段

* 每个进程都有自己的栈帧，函数执行完后，函数对应的栈帧也就结束了。
* 每个进程都有自己的栈段， 栈段中可以包含多个栈帧
* 函数的生命周期是函数调用到函数执行完毕
* 进程的生命周期，是程序执行开始到执行结束
* 如果变量的空间分配在栈帧里，变量的声明周期就是函数的生命周期。
* 栈段 代码段和数据段的生命周期都是进程
* 栈帧的生命周期是对应的函数的生命周期。
* 自动局部变量和函数的形参 这些变量的空间分布在栈帧里面
* 全局变量 局部静态变量 分配在数据段中。

1. Ubuntu查看系统版本信息的方法cat

查看显示内存中kernel相关的版本、编译等信息

Ubuntu和CentOS通用，可以显示内核版本同时显示发行版本

* root@ Cloud~Platform:~# cat /proc/version

Linux version 5.4.0-77-generic (buildd@lgw01-amd64-028) (gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)) #86-Ubuntu SMP Thu Jun 17 02:35:03 UTC 2021

* root@ Cloud~Platform:~# cat /proc/version

Linux version 4.18.0-147.el8.x86\_64 (mockbuild@kbuilder.bsys.centos.org) (gcc version 8.3.1 20190507

1. LRU算法

**原理简述：**当内存分配页面数（AP）小于进程页面数（PP）时，把最先执行的AP个页面放入内存。当需调页面进入内存，而当前分配的内存页面全部不空闲时，选择将其中最长时间没有用到的那一页调出，以空出内存来放置新调入的页面（LRU）。

**算法特点**：每个页面都有属性来表示有多长时间未被CPU使用的信息。

**算法实现提示：**要得到命中率，必然应该有一个常量total\_instruction来记录页面总共使用的次数，此外还需要一个变量记录总共换入页面的次数diseffect（需要换出页面总是因为缺页中断而产生）。利用公式1-diseffect / total\_instruction\*100% 可以得到命中率。

（1）初始化。设置两个数组page[ap]和pagecontrol[pp]分别表示进程页面数和内存分配的页面数，并产生一个随机数序列main[total\_instruction]（这个序列由page[ap]的下标随机构成）表示待处理的进程页面顺序，diseffect置0。

（2）看序列main[]中是否有下一个元素，如果有，就由main[]中获取该页面下标，并转（3），如果没有则转（6）。

（3） 如果该page[]单元在内存中便改变页面属性，使它保留“最近使用”的信息，转（2），否则转（4），同时diseffect加1。

（4）看是否有空闲页面，如果有，就返回页面指针，并转到（5），否则，在内存页面中找出最长时间没有使用到的页面，将其“清干净”，并返回该页面指针。

（5）在需处理的page[]与（4）中得到的pagecontrol[]之间建立联系，同时让对应的page[]单元保存“最新使用”的信息，转（2）。

（6）如果序列处理完成，就输出计算1-diseffect / total\_instruction\*100%的结果，完成。

1. 实验内容

**例1：创建进程，观测指针的地址和连续存储单元的地址关系。**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <string.h>

int main()

{

printf("pid : %d \n", getpid());

char \*str1 = "china";

char \*str2 = "china";

// 悬空指针

char \*str3 ;

// \*str3 = 'v';

int \*p;

// \*p = 100;

// error

char buf[32] = "china";

// error

//char buf[32];

//buf = "china";// error

strcpy(buf,"china");

printf("str1: %p\n",str1);

printf("str2: %p\n",str2);

printf("&str1: %p\n",&str1);

printf("&str2: %p\n",&str2);

printf("buf: %p\n",buf);

getchar();

return 0;

}

**编译后，运行：**

pid : 2875

str1: 0x4007bf

str2: 0x4007bf

&str1: 0x7fff2935a5c0

&str2: 0x7fff2935a5c8

buf: 0x7fff2935a5d0

**例2 函数的栈段实现**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <string.h>

int g\_val = 10;

int count ()

{

static int i = 1;

i++;

printf(" i = %d\n", i);

printf("&i = %p\n", &i);

return 0 ;

}

int main()

{

printf("pid : %d \n", getpid());

for (int n=0; n<5; n++)

count();

printf("&g\_val = %p\n", &g\_val);

getchar();

return 0;

}

**运行结果：**

pid : 3204

i = 2

&i = 0x601048

i = 3

&i = 0x601048

i = 4

&i = 0x601048

i = 5

&i = 0x601048

i = 6

&i = 0x601048

&g\_val = 0x601048

**例3 内存空间分配和释放**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

int main()

{

printf("pid : %d \n", getpid());

char \*p = (char \*)malloc(1024);

printf("p content %p\n",p);

strcpy(p,"beijing");

printf("p content %s\n",p);

// getchar();

free(p);

printf("p content %s\n",p);

return 0;

}

运行结果：

pid : 6825

p content 0x15e3420

p content beijing

p content

**例4.LRU 算法的实现**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#define AP 10

#define PP 3

#define TOTAL\_INSTRUCTION 20

//队列数据结构

int Queue[PP+1]={};

int head = 0;

int tail = 0;

//特殊的栈

int Stack[PP] = {};

int top = PP-1;

int bottom = PP-1;

//用于存储中间结果

int temp[PP][TOTAL\_INSTRUCTION];

//检查pagecontrol是否还有空位

int isConEmpty(int\* first\_empty,int pagecontrol[],int control\_num)

{

int flag = 0;

for(int iter = 0;iter<control\_num;iter++)

{

if(pagecontrol[iter]==-1)

{

flag = 1;

\*first\_empty = iter;

break;

}

}

return flag;

}

void LRU(int curpage,int page[],int pagecontrol[],int control\_num)

{

int first\_empty = -1;

//先检查在不在pagecontrol里面

for(int iter = 0; iter<PP;iter++)

{

if(pagecontrol[iter]==curpage)

{

int iter2;

for(iter2 = 0;iter2<PP;iter2++)

{

if(Stack[iter2]==iter)

break;

}

for(;iter2<PP-1;iter2++)

{

Stack[iter2]=Stack[iter2+1];

}

Stack[top]=iter;

return;

}

}

//如果pagecontrol中有空位，则直接放入空位

if(isConEmpty(&first\_empty,pagecontrol,control\_num))

{

pagecontrol[first\_empty]=curpage;

//该页面已在内存中

page[curpage] = 1;

//链表记录的是最先放入数据的位置而不是页面号

for(int iter = 0;iter<PP-1;iter++)

{

Stack[iter]=Stack[iter+1];

}

bottom--;

Stack[top]=first\_empty;

}

//如果没有空位，替换最近使用最少

else

{

page[pagecontrol[Stack[bottom+1]]]=0;

pagecontrol[Stack[bottom+1]]=curpage;

page[curpage] = 1;

int tmp = Stack[bottom+1];

for(int iter = 0;iter<PP-1;iter++)

{

Stack[iter]=Stack[iter+1];

}

Stack[top]=tmp;

}

}

int main()

{

//队列初始化为-1

for(int iter=0;iter<PP+1;iter++)

{

Queue[iter]=-1;

}

//栈初始化为-1

for(int iter=0;iter<PP;iter++)

{

Stack[iter]=-1;

}

/\*定义变量

\* page是页面

\* pagecontrol是内存分配的页面，初始化为-1

\* pageseq是页面使用的顺序

\*/

printf("-----------------------------------------------------\n");

printf("本程序模拟LRU算法：\n");

int page[AP];

int pagecontrol[PP];

for(int iter = 0;iter<PP;iter++)

{

pagecontrol[iter] = -1;

}

for(int iter = 0;iter<AP;iter++)

{

page[iter] = 0;

}

int pageseq[TOTAL\_INSTRUCTION] = {};

//为pageseq的每一个元素赋随机值

srand((unsigned int)getpid());

for(int iter = 0;iter<TOTAL\_INSTRUCTION;iter++)

{

pageseq[iter]=rand()%AP;

}

//测试

//int pageseq[TOTAL\_INSTRUCTION] = {7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};

//替换算法类型，0为FIFO，1为LRU

printf("可分配的物理块数量为：%d\n",PP);

printf("页面访问顺序为：\n%d",pageseq[0]);

for(int iter = 1;iter<TOTAL\_INSTRUCTION;iter++)

{

printf(" %d",pageseq[iter]);

}

printf("\n");

int cur = 0;

//diseffect是未命中次数

int diseffect = 0;

while(cur!=TOTAL\_INSTRUCTION)

{

if(page[pageseq[cur]]!=1)

diseffect++;

LRU(pageseq[cur],page,pagecontrol,PP);

for(int iter = 0;iter<PP;iter++)

{

temp[iter][cur]=pagecontrol[iter];

}

//查看下一个待调用的页面

cur++;

}

//计算命中率

double hit\_rate = 100\*(1-((double)diseffect/TOTAL\_INSTRUCTION));

//输出相关信息

printf("-----------------------------------------------------\n");

printf("替换次序(-1表示空)：\n");

for(int iter = 0; iter<PP;iter++)

{

printf("%d",temp[iter][0]);

for(int iter2 = 1;iter2<TOTAL\_INSTRUCTION;iter2++)

{

printf("\t%d",temp[iter][iter2]);

}

printf("\n");

}

printf("-----------------------------------------------------\n");

printf("小结：\n共访问页面次数，即TOTAL\_INSTRUCTION的大小：%d\n",TOTAL\_INSTRUCTION);

printf("未命中次数，即diseffect为：%d\n",diseffect);

printf("命中率，即1-diseffect / total\_instruction\*100%%为：%.2f%%\n",hit\_rate);

printf("-----------------------------------------------------\n");

return 0;

}

运行结果：

本程序模拟LRU算法：

可分配的物理块数量为：3

页面访问顺序为：

4 5 5 6 3 5 4 6 8 1 1 6 2 5 6 7 0 2 8 9

-----------------------------------------------------

替换次序(-1表示空)：

4 4 4 4 3 3 3 6 6 6 6 6 6 6 6 6 6 2 2 2

-1 5 5 5 5 5 5 5 8 8 8 8 2 2 2 7 7 7 8 8

-1 -1 -1 6 6 6 4 4 4 1 1 1 1 5 5 5 0 0 0 9

-----------------------------------------------------

小结：

共访问页面次数，即TOTAL\_INSTRUCTION的大小：20

未命中次数，即diseffect为：15

命中率，即1-diseffect / total\_instruction\*100%为：25.00%

-----------------------------------------------------

1. 实验题目
2. **实现FIFO算法**

**原理简述：**在分配内存页面数（AP）小于进程页面数（PP）时，当然是最先运行的AP个页面放入内存；这时又需要处理新的页面，则将原来放的内存中的AP个页中最先进入的调出（FIFO），再将新页面放入；

以后如果再有新页面需要调入，则都按上述规则进行。算法特点：所使用的内存页面构成一个队列。

**算法实现提示：**

要得到命中率，必然应该有一个常量total\_instruction来记录页面总共使用的次数，此外还需要一个变量记录总共换入页面的次数diseffect（需要换出页面总是因为缺页中断而产生）。利用公式1-diseffect / total\_instruction\*100% 可以得到命中率。

（1）初始化。设置两个数组page[ap]和pagecontrol[pp]分别表示进程页面数和内存分配的页面数，并产生一个随机数序列main[total\_instruction]（这个序列由page[ap]的下标随机构成）表示待处理的进程页面顺序，diseffect置0。

（2）看main[]中是否有下一个元素，若有，就由main[]中获取该页面下标，并转（3），如果没有则转（7）。

（3）如果该页已在内存中，就转（2），否则转（4），同时未命中的diseffect加1。

（4）观察pagecontrol是否占满，如果占满则须将使用队列（在第（6）步中建立的）中最先进入的（就是队列的第一个单元）pagecontrol单元“清干净”，同时将page[]单元置为“不在内存中”。

（5） 将该page[]与pagecontrol[]建立对应关系（可以改变pagecontrol[]的标志位，也可以采用指针链接，总之至少要使对应的pagecontrol单元包含两个信息：一是它被使用了，二是哪个page[]单元使用的。page[]单元也包含两个信息：对应的pagecontrol单元号和本page[]单元已在内存中）。

（6） 将用到的pagecontrol置入使用队列（这里的队列是一种FIFO的数据结构），返回（2）。

（7） 显示计算1-diseffect / total\_instruction\*100%，完成。

五、实验报告

将调试运行成功的程序和写好的实验报告一起压缩打包，以实验X-学号-姓名.rar这样的形式命名，并上传提交。