Memory Management

OS project3 t08902109 賈成銷

1. Motivation

Motivation:

run both the test cases below require lots of memory

/test/matmult.c

/test/sort.c

problem analysis:

想在 memory 有限的情況下運行這兩個程式，需要使用 virtual memory。

在 Virtual Memory 中，用了 3 個 table 來實作整個 VM 的架構。這 3 個 table 分別是 Frame table , Swap table , Pagetable。Frame table 裡面記錄所有關於實體記憶體的資訊; Swap table 裡面記錄所有關於虛擬記憶體的資訊而每一個執行的檔案都有各自的; Pagetable 記錄自己的記憶體空間的資訊。進而通過 page replacement method 根據不同的情況 load page。

2. Implementation

① 實現主記憶體不夠自動存入 virtual memory。

（1）UserProgKernel::Initialize

在裡面新增一個 vm_Disk，目的是當 memory 不夠時保存 page。

```
void
UserProgKernel::Initialize()
{
    ThreadedKernel::Initialize();       // init multithreading

    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    vm_Disk = new SynchDisk("New Disk");
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS

}
```

（2）當主記憶體不夠的時候，要把未分配的 PAGE 放到 virtral memory

需要修改：

\machine\machine.h 宣告

```
                                    // "read-only" to Nachos kernel code

    TranslationEntry *pageTable;
    unsigned int pageTableSize;
    bool ReadMem(int addr, int size, int* value);
    bool usedPhyPage[NumPhysPages];
    bool usedvirPage[NumPhysPages];
    int  ID_num;
    int PhyPageName[NumPhysPages];
    int count[NumPhysPages];
    bool reference_bit[NumPhysPages];
    int sector_number;

    TranslationEntry *main_tab[NumPhysPages];
```

修改 AddrSpace::AddrSpace()，我們直接註解掉下面部分部分

```
AddrSpace::AddrSpace()
{     ID=(kernel->machine->ID_num)+1;
      kernel->machine->ID_num=kernel->machine->ID_num+1;
/*    pageTable = new TranslationEntry[NumPhysPages];
    for (unsigned int i = 0; i < NumPhysPages; i++) {
        pageTable[i].virtualPage = i;   // for now, virt page # = phys page #
        pageTable[i].physicalPage = i;
//      pageTable[i].physicalPage = 0;
        pageTable[i].valid = TRUE;
//      pageTable[i].valid = FALSE;
        pageTable[i].use = FALSE;
        pageTable[i].dirty = FALSE;
        pageTable[i].readOnly = FALSE;
    }
*/
    // zero out the entire address space
//    bzero(kernel->machine->mainMemory, MemorySize);
}
```

修改 AddrSpace::Load(char *fileName)，以便分配 virtral memory

當 virtual memory 不夠用時，查找 virtual page。

```
 }
 //Use virtual memory when memory isn't enough
else{ char *buf;
      buf = new char[PageSize];
      k=0;
      while(kernel->machine->usedvirPage[k]!=FALSE){k++;}


     kernel->machine->usedvirPage[k]=true;
     pageTable[i].virtualPage=k; //record which virtualpage you save
         pageTable[i].valid = FALSE; //not load in main_memory
         pageTable[i].use = FALSE;
             pageTable[i].dirty = FALSE;
             pageTable[i].readOnly = FALSE;
     pageTable[i].ID =ID;
     executable->ReadAt(buf,PageSize, noffH.code.inFileAddr+(i*PageSize));
     kernel->vm_Disk->WriteSector(k,buf); //call virtual_disk write in virtual memory

      }
}
```

修改 AddrSpace::Execute(char *fileName) 及 AddrSpace::SaveState()

確保取得 pageTableSize 的值，pt_is_load 會在 AddrSpace::Execute(char *fileName)

轉換 FALSE-> 確認讀檔沒錯後 ->TRUE 才會去引發 AddrSpace::SaveState() 裡的 if 條件式

```
void
AddrSpace::Execute(char *fileName)
{   pt_is_load=FALSE;
    if (!Load(fileName)) {
        cout << "inside !Load(FileName)" << endl;
        return;                         // executable not found
    }

    //kernel->currentThread->space = this;
    this->InitRegisters();              // set the initial register values
    this->RestoreState();               // load page table register
    pt_is_load=TRUE;
    kernel->machine->Run();             // jump to the user progam

    ASSERTNOTREACHED();                 // machine->Run never returns;
                                        // the address space exits
                                        // by doing the syscall "exit"
}
```

到此完成了主記憶體不夠自動存入 virtual memory。

有關於檔案的傳輸都在 \machine\translate.cc 所以只要修改 translate.cc 即可實現 PAGE FAULT 之轉換。

② Page replacement

（1）>> FIFO

修改 Machine::Translate(int virtAddr, int* physAddr, int size, bool writing) 以便作 PAGE FAULT 之間的轉換。

新增宣告並修改 else if (!pageTable[vpn].valid) 的條件式，把原來的註解掉，表示 page fault 發生了，並開始處理 page fault

```
ExceptionType
Machine::Translate(int virtAddr, int* physAddr, int size, bool wr
{
    int i;
    unsigned int vpn, offset;
    TranslationEntry *entry;
    unsigned int pageFrame;

    int victim;///find the page victim
    int fifo;//For fifo

    unsigned int j;
```

```
victim = fifo % 32;
while(pageTable[victim].reference_bit == true)
        fifo++;      //find reference_bit is FALSE,and it can be replaced

pageTable[victim].reference_bit = true;        //not be replaced


printf("Number = %d page swap out\n",victim);

//get the page victm and save in the disk
bcopy(&mainMemory[victim*PageSize],buf_1,PageSize);
kernel->vm_Disk->ReadSector(pageTable[vpn].virtualPage, buf_2);
bcopy(buf_2,&mainMemory[victim*PageSize],PageSize);
kernel->vm_Disk->WriteSector(pageTable[vpn].virtualPage,buf_1);

main_tab[victim]->virtualPage=pageTable[vpn].virtualPage;
main_tab[victim]->valid=FALSE;


//save the page into the main memory


pageTable[vpn].valid = TRUE;
pageTable[vpn].physicalPage=victim;
kernel->machine->PhyPageName[victim]=pageTable[vpn].ID;
main_tab[victim]=&pageTable[vpn];
 fifo = fifo + 1;               //for fifo
    printf("page replacement finished\n");
```

（2）>> Random

只需將 victim 改為隨機取得：

```
        // pageTable[vpn].reference_bit = FALSE; //for second chance algo.

    kernel->vm_Disk->ReadSector(pageTable[vpn].virtualPage, buf);
    bcopy(buf,&mainMemory[j*PageSize],PageSize);

   }
  else{
        char *buf_1;
        buf_1 = new char[PageSize];
        char *buf_2;
        buf_2 = new char[PageSize];

      //Random
      victim = (rand()%32);

      //Fifo
     // victim = fifo%32;
```

（3）>> LRU

在 class TranslationEntry 裡新增宣告：

```
.ass TranslationEntry {
 public:
   unsigned int virtualPage;   // The page number in virtual memory.
   unsigned int physicalPage;  // The page number in real memory (relative to t
                               //  start of "mainMemory"
   bool valid;         // If this bit is set, the translation is ignored.
                       // (In other words, the entry hasn't been initialized.)
   bool readOnly;      // If this bit is set, the user program is not allowed
                       // to modify the contents of the page.
   bool use;           // This bit is set by the hardware every time the
                       // page is referenced or modified.
   bool dirty;         // This bit is set by the hardware every time the
                               // page is modified.
   int count;    //for LRU

   bool reference_bit; //for second chance algo.

   int ID;
```

修改 Machine::Translate(int virtAddr, int* physAddr, int size, bool writing) 以便作 PAGE FAULT 之間的轉換。

```
            char *buf; //save page temporary
            buf = new char[PageSize];
        kernel->machine->usedPhyPage[j]=TRUE;
        kernel->machine->PhyPageName[j]=pageTable[vpn].ID;

        kernel->machine->main_tab[j]=&pageTable[vpn];
        pageTable[vpn].physicalPage = j;
            pageTable[vpn].valid = TRUE;
            pageTable[vpn].count++; //for LRU
            pageTable[vpn].reference_bit = FALSE; //for second chance algo.
```

```
//Fifo
// victim = fifo%32;

//LRU

int min = pageTable[0].count;
victim=0;
for(int ccount=0;ccount<32;ccount++){
        if(min > pageTable[ccount].count){
                min = pageTable[ccount].count;
                victim = ccount;

        }
}
pageTable[victim].count++;
```

其他部分同 FIFO 和 random。

3. Result

nachos-4.0/code$ ./userprog/nachos -e ./test/matmult

```
de$ ./userprog/nachos -e ./test/matmult
Total threads number is 1
Thread ./test/matmult is executing.
page fault
Number = 7 page swap out
page replacement finished
page fault
Number = 6 page swap out
page replacement finished
page fault
Number = 9 page swap out
page replacement finished
page fault
Number = 19 page swap out
page replacement finished
page fault
```

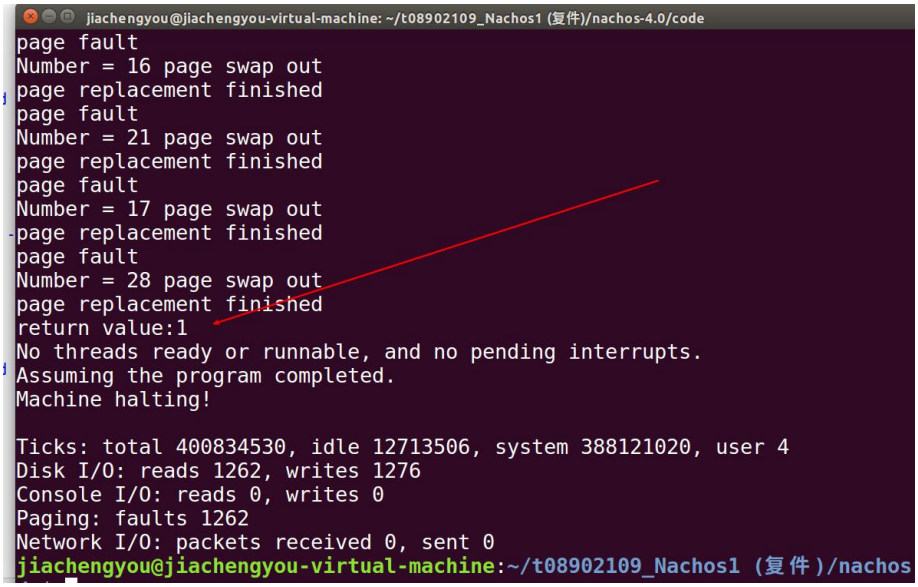
page fault happen and swap out

```
page fault
Number = 16 page swap out
page replacement finished
page fault
Number = 27 page swap out
page replacement finished
page fault
Number = 27 page swap out
page replacement finished
page fault
Number = 18 page swap out
page replacement finished
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 7651580, idle 1325676, system 6325900, user 4
Disk I/O: reads 89, writes 111
```

right answer!

/nachos-4.0/code$ ./userprog/nachos -e ./test/sort


jiachengyou@jiachengyou-virtual-machine: ~/t08902109_Nachos1 (复件)/nachos-4.0/code
page fault
Number = 16 page swap out
page replacement finished
page fault
Number = 21 page swap out
page replacement finished
page fault
Number = 17 page swap out
page replacement finished
page fault
Number = 28 page swap out
page replacement finished
return value:1
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
Ticks: total 400834530, idle 12713506, system 388121020, user 4
Disk I/O: reads 1262, writes 1276
Console I/O: reads 0, writes 0
Paging: faults 1262
Network I/O: packets received 0, sent 0
jiachengyou@jiachengyou-virtual-machine:~/t08902109_Nachos1 (复件)/nachos