Operating Systems
Project 3 -
Memory Management

機械所 吳冠甫 R10522532

December 16th, 2022

Contents

Motivation

Goals

Both the folling test cases require lots of memory, and our gola is to run this two program concurrently

1. sort

```
./nachos -e ../test/sort
```

```
Total threads number is 1
Thread ../test/sort is executing.
Assertion failed: line 118 file ../userprog/addrspace.cc
Aborted (core dumped)
```

2. matmult

```
./nachos -e ../test/matmult
```

```
Total threads number is 1
Thread ../test/matmult is executing.
Assertion failed: line 118 file ../userprog/addrspace.cc
Aborted (core dumped)
```

3. sort and matmult concurrently

```
./nachos -e ../test/sort -e ../test/matmult
```

```
Total threads number is 2
Thread ../test/sort is executing.
Thread ../test/matmult is executing.
Assertion failed: line 118 file ../userprog/addrspace.cc
Aborted (core dumped)
```

In this project, I use demand paging to solve the limited memory issues. In a system that uses demand paging, the operating system copies a disk page into physical memory only if an attempt is made to access it and that page is not already in memory. To achieve this process, a page table implementation is used. The page table maps logical memory to physical memory. The page table uses a bitwise operator to mark if a page is valid or invalid. A valid page is one that currently resides in main memory. An invalid page is one that currently resides in secondary memory.

Motivation continued on next page. . .

Implementation

Simulate secondary storage

Create a new `SynchDisk` called `SwapDisk` to simulate the secondary storage. Pages demanded by the process are swapped from secondary storage to main memory.

```
class SynchDisk;
class UserProgKernel : public ThreadedKernel {
public:
    ...
    // add
    SynchDisk *SwapDisk       // SwapDisk saves pages if main memory is not enough
    bool debugUserProg;       // single step user program
    ...
private:
    // remove
    // bool debugUserProg;    // single step user program
    ...
};
```

Listing 1: /code/userprog/userkernel.h

Initialize `SwapDisk`

```
void UserProgKernel::Initialize()
{
    ThreadedKernel::Initialize(RR); // init multithreading
    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    SwapDisk = new SynchDisk("New SwapDisk");// Swap disk for virtual memory
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}

void UserProgKernel::Initialize(SchedulerType type)
{
    ThreadedKernel::Initialize(type);   // init multithreading
    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    // add
    SwapDisk = new SynchDisk("New SwapDisk");// Swap disk for virtual memory
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}
```

Listing 2: /code/userprog/userkernel.cc

Record the information about which frames of main and virtual memeory are occupied.

```
...
// add
bool UsedPhyPage[NumPhysPages];      // Record the pages in the main memory
bool UsedVirtualPage[NumPhysPages]; // Record the pages in the virtual memory
int  ID_number;                      // Machine ID
int  PhyPageInfo[NumPhysPages];      // Record physical page info (ID)
...
```

Listing 3: /code/machine/machine.h

Implementation [Simulate secondary storage] continued on next page. . .

To implement virtual memory system. We modify the `Load` funciton in `/code/userprog/addrspace.cc`. The following for loop is used to find the available space for the page of this process, and the while loop check whether the `j-th` frame is used. The index `j` will increase 1 is being used, until an empty frame or exceed the number of physical pages. There are two different cases as following.

1. When the main memory stil have empty frame
   - We can put the page into the main memory and update the information to the page table
   - This step is achieved by the function `ReadAT`
2. Main memory is fulled
   - We have check the available virtual memory space by the similar while loop and write the page into `SwapDisk` by the `WriteSector` function.

```
class AddrSpace {
    ...
private:
    ...
    bool pageTable_is_load;
}
```

Listing 4: /code/userprog/addrspace.h

```
if (noffH.code.size > 0) {
     for(unsigned int j = 0, i = 0; i < numPages; i++) {
         //
         j = 0;
         while ((kernel->machine->UsedPhyPage[j]!=FALSE) &&
                (j < NumPhysPages)) {
             j++;
         }

         // Case 1: main memory is enough, put the page to main memory
         if(j<NumPhysPages){
             kernel->machine->UsedPhyPage[j]  = TRUE;
             kernel->machine->PhyPageInfo[j]  = ID;
             kernel->machine->main_tab[j]     = &pageTable[i];

             pageTable[i].physicalPage = j;
             pageTable[i].valid        = TRUE;
             pageTable[i].use          = FALSE;
             pageTable[i].dirty        = FALSE;
             pageTable[i].readOnly     = FALSE;
             pageTable[i].ID           = ID;
             pageTable[i].LRU_counter++; // LRU counter when save in memory

             //
             executable->ReadAt(
                 &(kernel->machine->mainMemory[j*PageSize]),
                 PageSize,
                 noffH.code.inFileAddr + (i*PageSize));
         }
         // Case 2: main memory is not enough, use virtual memory
         else{
             //
             char *buffer;
             buffer = new char[PageSize];
             tmp = 0;
```

```
            //
            while (kernel->machine->UsedVirtualPage[tmp] != FALSE) {
                tmp++;
            }

            //
            kernel->machine->UsedVirtualPage[tmp] = TRUE;
            pageTable[i].virtualPage = tmp;   // Record the virtual page we 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(
                buffer,
                PageSize,
                noffH.code.inFileAddr+(i*PageSize));
            kernel->SwapDisk->WriteSector(tmp,buffer); // write in virtual memory (SwapDisk
    )
        }
    }
}
```

Listing 5: /code/userprog/addrspace.cc

Implementation continued on next page. . .

Scheduling

After the implementation above, the operating system is capable to put those pages that cannot put in main memory to virtual memory. In the following steps, we implement the page replacement algorithm **Least Recently Used (LRU)** Implement LRU by adding a hardware counter `LRU_counter`

```
class TranslationEntry {
public:
    ...
    // add
    int LRU_counter;    // counter for LRU
    int ID;             // page table ID
    ...
};
```

Listing 6: /code/machine/translate.h

First, we check the valid-invalid bit of the page table. If is not valid, it means the page is not in the main memory. Thus, we need to load from the secondary storage. There are two cases as follows:

1. There are some empty frame in the main memory
   - Just load the page into it.
2. The main memory is fulled
   - Create two buffers and run LRU algorithm to find the victim.
   - The `ReadSector` and `WriteSector` are used to read/write the temporarily saved pages found by the LRU algorithm
   - The victim is pull out from the main memory, and then swapped by our page into the corresponding frame.

The page table will be updated correspondingly in both cases. In the following implementation, LRU will perform linear search on `LRU_counter` to find the least recently used page.

```
else if (!pageTable[vpn].valid) {
    // not in main memory, demand paging
    kernel->stats->numPageFaults++; // page fault
    j = 0;

    while((kernel->machine->UsedPhyPage[j] != FALSE) &&
          (j < NumPhysPages)) {
        j++;
    }

    // Case 1: load the page into the main memory if the main memory is not full
    if (j < NumPhysPages) {
        //
        char *buffer; //temporary save page
        buffer = new char[PageSize];

        kernel->machine->UsedPhyPage[j] = TRUE;
        kernel->machine->PhyPageInfo[j] = pageTable[vpn].ID;
        kernel->machine->main_tab[j]    = &pageTable[vpn];
        pageTable[vpn].physicalPage     = j;
        pageTable[vpn].valid            = TRUE;
        pageTable[vpn].LRU_counter++;   // counter for LRU

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

Implementation [Scheduling] continued on next page...

```
    }
    // Case 2: main memory is full, page replacement
    else{
        //
        char *buffer1;
        char *buffer2;
        buffer1 = new char[PageSize];
        buffer2 = new char[PageSize];

        //LRU
        int min = pageTable[0].LRU_counter;
        victim=0;
        for(int index=0;index<32;index++){
            if(min > pageTable[index].LRU_counter){
                min = pageTable[index].LRU_counter;
                victim = index;
            }
        }
        pageTable[victim].LRU_counter++;

        // perform page replacement, write victim frame to disk, read
        // desired frame to memory
        bcopy(&mainMemory[victim*PageSize],buffer1,PageSize);
        kernel->SwapDisk->ReadSector(pageTable[vpn].virtualPage, buffer2);
        bcopy(buffer2,&mainMemory[victim*PageSize],PageSize);
        kernel->SwapDisk->WriteSector(pageTable[vpn].virtualPage,buffer1);

        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->PhyPageInfo[victim]  = pageTable[vpn].ID;
        main_tab[victim]                      = &pageTable[vpn];
    }
}
```

Listing 7: /code/machine/translate.cc

Results

Sort

```
Total threads number is 1
Thread ../test/sort is executing.
return value:1
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 441639030, idle 53252866, system 388386160, user 4
Disk I/O: reads 5681, writes 5695
Console I/O: reads 0, writes 0
Paging: faults 5681
Network I/O: packets received 0, sent 0
```

Listing 8: Result of sort

Matmult

```
Total threads number is 1
Thread ../test/matmult is executing.
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 7627030, idle 1301666, system 6325360, user 4
Disk I/O: reads 80, writes 102
Console I/O: reads 0, writes 0
Paging: faults 80
Network I/O: packets received 0, sent 0
```

Listing 9: Result of matmult

Sort and Matmult concurrently

```
Total threads number is 2
Thread ../test/sort is executing.
Thread ../test/matmult is executing.
return value:1
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 450716530, idle 56002285, system 394714240, user 5
Disk I/O: reads 5790, writes 5858
Console I/O: reads 0, writes 0
Paging: faults 5790
Network I/O: packets received 0, sent 0
```

Listing 10: Result of sort and matmult concurrently