CE2005 Operating Systems Experiment 3
Group SE1

Mehmet Fazıl Aydınlı
N1701894K

November 22, 2017

1 Introduction

In this lab experiment, we are required to complete a virtual memory implementation where a software based Translation Look Aside Buffer (TLB) and Inverted Page Table (IPT) with least recently used page replacement algorithm adopted.

The use of IPT saves memory space as all the processes use it for adress translation instead of having their own page tables. However it is relatively slower than regular page table implementation because it requires intensive search to find which process uses the which frame. For our experiment, IPT is implemented as a collection of memory tables.

Listing 1: ipt.h

```
1   class MemoryTable {
2       public:
3        MemoryTable(void);
4        ~MemoryTable(void);
5
6       bool valid;     // if frame is valid (being used)
7       SpaceId pid;        // pid of frame owner
8       int vPage;          // corresponding virtual page
9       bool dirty;     // if needs to be saved
10      int TLBentry;       // corresponding TLB entry
11      int lastUsed;       // used to see record last used tick
12      OpenFile *swapPtr;  // file to swap to
13  };
```

As we know, TLB is a cache hardware that helps to reduce memory access time. In NachOS, software-managed TLB is used and it is basically a collection of translation entries.

As defined in machine.h header file, number of page frames is four and the size of TLB is three.

Listing 2: machine.h

```
1   #define NumPhysPages    4
2   #define MemorySize   (NumPhysPages * PageSize)
3   #define TLBSize         3
```

In the case of virtual address translation, the system first tries to find the corresponding frame number in TLB. If found, TLB hit happens. If not, the system searchs IPT, and if succesfully finds it then updates TLB and goes back to TLB lookup stage. If the virtual page cannot be found in IPT, then the system pages out a victim page determined by least recently used algorithm and pages in the new table.

2 Implementations

2.1 InsertToTlb

This function puts a virtual page number and its associated physical page into the TLB. To find the insert entry, it first checks whether there is an old entry with valid bit is false, meaning that it's process is context switched out or dead. If there is such an entry, then function returns to it. If not, a simple FIFO is used to kick out the oldest entry in the TLB.

Listing 3: InsertToTLB

```
1   void InsertToTLB(int vpn, int phyPage){
2       static int FIFOPointer = 0;
3       int i = 0;
4       // search for unvalid entry
5       for (i = 0; i < TLBSize; i++){
6           if (machine->tlb[i].valid == false){
7               break;
8           }
9       }
10      // if all valid, use FIFO
11      if (i == TLBSize){
12          i = FIFOPointer;
13      }
14      // update FIFO info
15      FIFOPointer = (i + 1) % TLBSize;
16  ...
```

2.2 VpnToPhyPage

This function tries to find a corresponding physical frame for a virtual page number in IPT. If it does not exist, returns -1. Look up procedure in IPT is fairly simple, as it is a collection of memory tables, what need to be done is to check whether the memory table is valid with process id and virtual page number matches.

Listing 4: VpnToPhyPage

```
1   int VpnToPhyPage(int vpn){
2       int i = 0;
3       // search IPT
4       for (i = 0; i < NumPhysPages; i++){
5           // check for validity, pid and vpn match
6           if (memoryTable[i].valid == true &&
7               memoryTable[i].pid == currentThread->pid &&
8               memoryTable[i].vPage == vpn){
9               // if found, return index
```

```
10                  return i;
11                  break;
12              }
13          }
14          // if not found, return -1
15          if (i == NumPhysPages){
16              return -1;
17          }
18      }
```

2.3 lruAlgorithm

To find the victim page, least recently used algorithm is used. It uses the lastUsed information stored in IPT. Firsty, it checks whether there is an invalid frame to easily select the victim page. But if all pages are valid, then the algorithms compares the last used information in each frame to select the victim page.

Listing 5: InsertToTLB implementation

```
1       int lruAlgorithm(void){
2           int phyPage;
3           int i = 0;
4               // search for invalid entry
5               for (i = 0; i < NumPhysPages; i++){
6                   if (memoryTable[i].valid == false){
7                       // return an invalid one
8                       phyPage = i;
9                       break;
10                  }
11              }
12          // if all valid, use the least recently used one
13          if (i == NumPhysPages){
14              // randomly assign a min, it does not matter
15              //since we will compare all entries.
16              int minTick = memoryTable[0].lastUsed;
17              int minTickIndex = 0;
18              for (i = 0; i < NumPhysPages; i++){
19                  if (memoryTable[i].lastUsed < minTick){
20                      minTick = memoryTable[i].lastUsed;
21                      minTickIndex = i;
22                  }
23              }
24              // return the least recently used
25              phyPage = minTickIndex;
26          }
```

```
27          return phyPage;
28      }
```

3 Analysis

After implementing the missing functions, the program is executed with test input file. In order to get the values of IPT and TLB, I added DEBUG commands before the changes in each exception as shown below. Also I used the default debug outputs to fill the table such as paging in, paging out information.

Listing 6: DEBUG commands for IPT and TLB values

```
1       DEBUG('p', "Before updating IPT %i, pid, vpn, last used and
2           valid is %i %i %i %i \n", phyPage,
3           memoryTable[phyPage].pid, memoryTable[phyPage].vPage,
4           memoryTable[phyPage].lastUsed, memoryTable[phyPage].valid);
5
6       DEBUG('p', "Before updating TLB %i, vpn, phy and valid is
7            %i %i %i \n", i,  machine->tlb[i].virtualPage,
8            machine->tlb[i].physicalPage, machine->tlb[i].valid);
```

The page size defined in machine.h is referred as Sector Size, and it's value can be found in disk.h. In this experiment, the page size is 128 bytes. The number of physical frames is 4 and the TLB size is 3, as mentioned in Listing 2.

Listing 7: disk.h

```
1       ...
2       #define SectorSize  128 // number of bytes per disk sector
3       ...
```

In the following table, first three columns represent the tick when the page fault exception is occurred, the virtual page number and the corresponding process id. The following four columns are four IPT entries, and each IPT entries has four values; *process id, virtual page number, last accessed tick, valid flag*. The next three columns are three TLB entries, and each entry has three values; *virtual page number, physical frame number and valid flag*. The last column records the dirty page that is paged out. Rows are recorded before the selection, and the updated entries are highlighted.

As we can in the table, the test program uses 5 different virtual pages, numbered as 0,1,9,10 and 26. As the number of physical frames is four, system pages out whenever the fifth different page is in need and it happened in ticks 28 and 62. So there are two page outs in this test run.

There are total of 19 TLB misses as it is basically number of highlighted TLB entries and 14 page faults, also highlighted. FIFO algorithm is used in ticks 20,26,41,42,4769,74,123 and 125 as all TLB entries were valid at those ticks.

tick	vpn	pid	IPT[0]	IPT[1]	IPT[2]	IPT[3]	TLB[0]	TLB[1]	TLB[2]	Phy page out
10	0	0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0	0,0,0	
13	9	0	0,0,12,1	0,0,0,0	0,0,0,0	0,0,0,0	0,0,1	0,0,0	0,0,0	
15	26	0	0,0,12,1	0,9,15,1	0,0,0,0	0,0,0,0	0,0,1	9,1,1	0,0,0	
20	1	0	0,0,12,1	0,9,19,1	0,26,17,1	0,0,0,0	0,0,1	9,1,1	26,2,1	
26	0	0	0,0,12,1	0,9,19,1	0,26,17,1	0,0,0,0	1,3,1	9,1,1	26,2,1	
28	10	0	0,0,28,1	0,9,25,1	0,26,17,1	0,1,22,1	1,3,1	0,0,1	26,2,0	2
41	9	0	0,0,40,1	0,9,25,1	0,10,28,1	0,1,22,1	1,3,1	0,0,1	10,2,1	
42	26	0	0,0,40,1	0,9,42,1	0,10,28,1	0,1,22,1	9,1,1	0,0,1	10,2,1	
47	0	0	0,0,40,1	0,9,46,1	0,10,28,1	0,26,44,1	9,1,1	26,3,1	10,2,1	
59	0	1	0,0,49,1	0,9,46,1	0,10,28,1	0,26,44,1	9,1,0	26,3,0	0,0,0	
62	9	1	0,0,49,1	0,9,46,1	1,0,61,1	0,26,44,1	0,2,1	26,3,0	0,0,0	3
64	26	1	0,0,49,1	0,9,46,1	1,0,61,1	1,9,64,1	0,2,1	9,3,1	0,0,0	
69	1	1	0,0,49,1	1,26,66,1	1,0,61,1	1,9,68,1	0,2,1	9,3,1	26,1,1	
74	0	1	1,1,71,1	1,26,66,1	1,0,61,1	1,9,73,1	1,0,1	9,3,1	26,1,1	
117	0	0	1,1,71,0	1,26,66,0	1,0,76,0	1,9,73,0	1,0,0	0,2,0	26,1,0	
120	9	0	0,0,119,1	1,26,66,0	1,0,76,0	1,9,73,0	0,0,1	0,2,0	26,1,0	
122	10	0	0,0,119,1	0,9,121,1	1,0,76,0	1,9,73,0	0,0,1	9,1,1	26,1,0	
123	26	0	0,0,119,1	0,9,121,1	0,10,123,1	1,9,73,0	0,0,1	9,1,1	10,2,1	
125	0	0	0,0,119,1	0,9,121,1	0,10,124,1	0,26,124,1	26,3,1	9,1,1	10,2,1	