|  |
| --- |
| Nanyang technological University |
| CZ2005 Operating Systems |
| **Experiment 3: Virtual Memory** |
| |  |  | | --- | --- | | Name: | Yiu Shun Chi | | Martic No.: | N1902077E | | Date: | 29/10/2019 | | Group: | SS8 | |  |  | |
|  |
|  |

**Introduction**

The experiment focused on virtual memory implementation in NachOS. To simulate the working principles, Translation Look-aside Buffer (TLB) and Inverted Page Table (IPT) are coded to provide insight of address translation, while Least Recently Used replacement algorithm is used for page replacement in this experiment. The following part of the report will show detailed implementation of functions and analyze the output of the testing program with explanation.

**Background Condition**

The pre-set conditions of the program provide a convenient environment for subsequent code development. The target functions of the lab are developed base on certain pre-defined data and their structures. This part will go through them with explanation.

Using IPT can reduce memory space by making use of address translation than traditional method of using page tables. Before implementing the search for physical frame in IPT, it is important to observe how IPT is constructed in NachOS. In the experiment, memory tables class is used per page frame, so IPT is made through an array of tables, with size NumPhysPage.

File: *Ipt.h*

class MemoryTable {

public:

MemoryTable(void);

~MemoryTable(void);

bool valid; // if frame is valid (being used)

SpaceId pid; // pid of frame owner

int vPage; // corresponding virtual page

bool dirty; // if needs to be saved

int TLBentry; // corresponding TLB entry

int lastUsed; // used to see record last used tick

OpenFile \*swapPtr; // file to swap to

};

To insert an entry into software-managed BLT, the program should load it into an empty position or replace others when full. Similar to IPT, BLT is also an array but of translation entries, with defined size 3, setting bound for the iterative position searching.

File: *Translate.h*

class TranslationEntry {

public:

int virtualPage; // The page number in virtual memory.

int physicalPage; // The page number in real memory (relative to the

// 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.

};

File: *Machine.h*

#define NumPhysPages 4

#define MemorySize (NumPhysPages \* PageSize)

#define TLBSize 3

The LRU algorithm in this experiment is called if a virtual page cannot be found within IPT, so that it can prioritize to page in the new page entry while page out the least recent used page as restricted with limited space.

**Implementation**

1. Function ***VpnToPhyPage***

This function searches for physical frame of the corresponding virtual page number (vpn) existing in ipt. The method is to iterate through the existing memory table and retrieve the corresponding index in the table which virtual page number is as indicated. Extra validations are required and implemented in the function: whether it is valid (emptiness), and its process ID is the same as the current running thread. The function will return the index if corresponding table is found and return -1 otherwise.

int VpnToPhyPage(int *vpn*)

{

for(int phyPage=0;phyPage<NumPhysPages;phyPage++){

if (memoryTable[phyPage].valid==true && memoryTable[phyPage].pid==currentThread->pid && memoryTable[phyPage].vPage==vpn){

return phyPage;

}

}

return -1;

}

1. Function ***InsertToTLB***

This function inserts virtual page and its corresponding physical page into the TLB. In particular, the code following shows the part where the function searches the position in TLB for entry. There are 2 cases possible: insert the entry into empty position or replace the oldest entry exist in TLB if there is no empty position. For the first case, the method is to loop through the TLB and find out the corresponding index that is empty. For the second case, a static variable is created to keep track on the oldest entry to pertain the value across program execution, even outside the scope of function. Therefore, the incoming entry could replace the oldest entry in TLB.

void InsertToTLB(int *vpn*, int *phyPage*)

{

  int i = 0;*//entry in the TLB*

  static int oldest=0;

  for (;i<TLBSize;i++){

    if (machine->tlb[i].valid == false){

      break;

    }

  }

  if (i==TLBSize){

    i=oldest;

  }

......

oldest=(i+1)%TLBSize;

}

1. Function ***lruAlgorithm***

This function is the core algorithm of performing page replacement. Least recent used algorithm is used in this experiment. This algorithm determines the virtual page in and out in the physical memory and returns the physical frame to be inserted or replaced. The function consists of 2 parts. The first part checks for empty page in the table, if there is, the function returns the corresponding page index in the table. If there is no empty page in the table, it proceeds to the second part where the new coming entry will replace the least recent used entry across the whole table, with reference to the variable *least*.

int lruAlgorithm(void)

{

  int phyPage;

  int least=memoryTable[0].lastUsed;

  for(phyPage=0;phyPage<NumPhysPages;phyPage++) {

    if(memoryTable[phyPage].valid==false){

      return phyPage;

    }

  }

  if (phyPage==NumPhysPages){

    phyPage=0;

    for(int i=1;i<NumPhysPages;i++){

      if(memoryTable[i].lastUsed<least) {

        least=memoryTable[i].lastUsed;

        phyPage=i;

      }

    }

  }

  return phyPage;

}

**Analysis**

After the implementation of code, the program is ready to be executed. The given code included DEBUG function, which reveals the internal process during the execution by outputting message with runtime data, so the flow of the program can be visualized. However, in order to observe every detailed step and in a clearer manner, adding printf function after important operations, such as the following table, allows easier analysis of the program output.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| tick | vpn | pid | IPT[0] | IPT[1] | IPT[2] | IPT[3] | TLB[0] | TLB[1] | TLB[2] | 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 |  |

The first three columns are the tick that the page fault exception occurred, the virtual page number and the corresponding process id. The next four columns represent each entry in the IPT, and each of them consist of the process id, virtual page number, last accessed tick and valid flag. The next three columns represent each entry in the TLB, and each of them consists of the virtual page number, physical frame number, and the valid flag of the entry. The last column records if any dirty page that is paged out.

From the table, it is observed that there is a total of 19 TLB misses, and 14 pages faults in total. The highlighted cells are the entries to be changed in the corresponding tick. And the number of pages out is 2, occurred when tick 28 and tick 62 is accessed. The number of pages used in the program is 5, which are 0,1,9,10 and 26, as seen from the second value of the entries in IPT or the first in TLB.

As defined in NachOS, in file machine.h, the page size is defined as the SectorSize, while the definition and its values can be found in disk.h, which value is 128 bytes. While the number of physical frames and the TLB size is 4 and 3 respectively, referencing from file machine.h. The following shows the source of evidence.

*Machine.h*

#define PageSize SectorSize // set the page size equal to

// the disk sector size, for

// simplicity

#define NumPhysPages 4

#define TLBSize 3 // if there is a TLB, make it small

*Disk.h*

#define SectorSize 128 // number of bytes per disk sector

**Conclusion**

The experiment provides an insight in how virtual memory systems works in Operating System. Through implementing the functions with analyzation of result, step-by-step changes in page table, TLB and IPT during the process leads to deeper understanding in address translation. The LRU algorithm suggests a possible solution of algorithm in page replacement. Actual implementation allows deeper understanding in how the algorithm works as well as its importance. An efficient address translation and appropriate page replacement algorithm contribute to an effective virtual memory system.