CZ2005 Operating Systems

Kwek Dan Yi, Germaine, U1722265L

Lab group BCG2

**Implementation of Missing Routines**

int VpnToPhyPage(int vpn)

{

  //your code here to get a physical frame for page vpn

  //you can refer to PageOutPageIn(int vpn) to see how an entry was created in ipt

  //declare the offset

  int i;

  //loop through each record in the memory table, aka the IPT

  for (i=0; i<NumPhysPages; i++)

  {

    //if that record contains a valid page, and its pid matches that of the current thread's, and its virtual page matches that of the vpn passed in

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

    {

      //correct record is found, and we return the offset, which gives the physical page

      return i;

    }

  }

  //if the correct record cannot be found, we return -1

  return -1;

}

void InsertToTLB(int vpn, int phyPage)

{

  int i = 0; //entry in the TLB to replace

  //your code to find an empty in TLB or to replace the oldest entry if TLB is full

  //declare a variable to use to loop through the TLB to search for an invalid record

  int j;

  //declare a variable to indicate if an invalid record is found

  //initialise it to not found

  int found = 0;

  //initialise the first in first out pointer

  static int FIFOPointer = 0;

  //loop through each record in the TLB

  for (j=0; j<TLBSize; j++)

  {

    //if the process is not valid

    if (!machine->tlb[j].valid)

    {

      //set i to be the index of that record

      i = j;

      //indicate that an invalid record has been found

      found = 1;

      //break

      break;

    }

  }

  //if an invalid record cannot be found, use first in first out

  if (found == 0)

  {

    //set i to be FIFOPointer

    i = FIFOPointer;

  }

FIFOPointer = (i + 1) % TLBSize;

  // copy dirty data to memoryTable

  if(machine->tlb[i].valid){

  memoryTable[machine->tlb[i].physicalPage].dirty=machine->tlb[i].dirty;

  memoryTable[machine->tlb[i].physicalPage].TLBentry=-1;

  }

  //update the TLB entry

  machine->tlb[i].virtualPage = vpn;

  machine->tlb[i].physicalPage = phyPage;

  machine->tlb[i].valid = TRUE;

  machine->tlb[i].readOnly = FALSE;

  machine->tlb[i].use = FALSE;

  machine->tlb[i].dirty = memoryTable[phyPage].dirty;

  //update the corresponding memoryTable

  memoryTable[phyPage].TLBentry=i;

  DEBUG('p', "The corresponding TLBentry for Page %i in TLB is %i ", vpn, i);

  //reset lastUsed to current ticks since it is being used at this moment.

  //for the implementation of LRU algorithm.

  memoryTable[phyPage].lastUsed = stats->totalTicks;

  //increase the number of tlb misses

  stats->numTlbMisses++;

}

int lruAlgorithm(void)

{

  //your code here to find the physical frame that should be freed

  //according to the LRU algorithm.

  int phyPage; //the page we want to replace

  //declare a variable to loop through the memory table, aka the IPT

  int i;

  //declare a variable to indicate the current smallest last used value

  int smallestLastUsedVal;

  //declare a variable to indicate the record that holds the current smallest last used value

  int lruPage;

  //for each record in the memory table, aka the IPT

  for (i=0; i<NumPhysPages; i++)

  {

    //if that page is not valid, page it out

    if (!memoryTable[i].valid)

    {

      //capture the record in phyPage

      phyPage = i;

      //return the phyPage

      return phyPage;

    }

  }

  //if no invalid record is found, we have to search for and return the least used page

  //initialise the least recently used page to the first record

  lruPage = 0;

  //initialise the smallest last used value to that of the first record in the memory table

  smallestLastUsedVal = memoryTable[0].lastUsed;

  //loop though each record in the memory table

  for (i=1; i<NumPhysPages; i++)

  {

    //if the last used for that page is less than that currently held

    if (memoryTable[i].lastUsed < smallestLastUsedVal)

    {

      //reassign the last used value

      smallestLastUsedVal = memoryTable[i].lastUsed;

      //reassign the index that holds least used page

      lruPage = i;

    }

  }

  //upon exiting the loop, we have searched through every record and have the overall least used page

  //assign that to phyPage

  phyPage = lruPage;

  //return phyPage

  return phyPage;

}

**Explanation of Program**

When an address translation needs to be made to translate a virtual page number into a physical frame number, the TLB is first examined to check if there is a matching entry for that virtual page number. If there is, the corresponding physical frame number from that same entry will be used as address translation. This constitutes a TLB hit.

If there is a TLB miss, the IPT will be looked up. We go through every entry in the IPT to look for one which contains the matching process index and virtual page number, and whose process is not dead. When such an entry is found, we return its offset as that gives the physical frame number. This constitutes an IPT hit and we will use it to update the TLB. If there is an IPT miss, it means that the physical frame has not even been loaded into the memory. In that case, we have to load it from the disk and a page in and page out will be performed.

Let us discuss what happens if there is an IPT hit. We will first go through the TLB to find an entry with an invalid process. If such an entry exists, we will replace it with the valid entry from the IPT. If there is no entry with an invalid process, the entry to be replaced will be that pointed to by the FIFOPointer, or first in first out pointer.

If there is an IPT miss, we need to load the frame from the disk to memory. A page in and page out will need to be performed. To decide which page to page out, a page replacement policy is used. In this case, we adopt the least recently used algorithm.

The least recently used algorithm first checks for an invalid entry in the IPT. If such an entry is found, it will be paged out. If there is no such entry, we search for and page out the least recently used page, aka that with the smallest last used field.

After updating the IPT, we can update the TLB.

Once the TLB has been updated, a second lookup will be performed to do the address translation.

**Test Program Output**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| tick | vpn | pid | IPT[0] | IPT[1] | IPT[2] | IPT[3] | TLB[0] | TLB[1] | TLB[2] | Page Out |
|  |  |  | pid, vpn, last used, valid |  |  |  | vpn, phy, valid |  |  | pid, vpn |
| 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 | 0,9,25,1 | 0,26,17,1 | 0,1,22,1 | 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 | 26,2,1 | 0,26 |
| 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 | 0,26 |
| 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 |  |

**Analysis of Results**

Tick 10: TLB miss. Searching for pid 0, vpn 0. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[0] is updated as it is the first entry found with a valid bit of 0. In the same way, TLB[0] is updated.

Tick 13: TLB miss. Searching for pid 0, vpn 9. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[1] is updated as it is the first entry found with a valid bit of 0. In the same way, TLB[1] is updated.

Tick 15: TLB miss. Searching for pid 0, vpn 26. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[2] is updated as it is the first entry found with a valid bit of 0. In the same way, TLB[2] is updated.

Tick 20: TLB miss. Searching for pid 0, vpn 1. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[3] is updated as it is the first entry found with a valid bit of 0. TLB[0] is updated because it is the oldest entry to be updated.

Tick 26: TLB miss. Searching for pid 0, vpn 0. The IPT is then searched and a matching entry is found in IPT[0]. TLB[1] is updated because it is the oldest entry to be updated.

Tick 28: TLB miss. Searching for pid 0, vpn 10. The IPT is then searched, but still incurs a page fault. A page in and page out is done and IPT[2] is updated as it has the smallest last used tick value. In the same way, TLB[2] is updated because it is the oldest entry to be updated.

Tick 41: TLB miss. Searching for pid 0, vpn 9. The IPT is then searched and a matching entry is found in IPT[1]. TLB[0] is updated because it is the oldest entry to be updated.

Tick 42: TLB miss. Searching for pid 0, vpn 26. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[3] is updated as it has the smallest last used tick value. TLB[1] is updated because it is the oldest entry to be updated.

Tick 47: TLB miss. Searching for pid 0, vpn 0. The IPT is then searched and a matching entry is found in IPT[0]. TLB[2] is updated because it is the oldest entry to be updated.

Tick 59: TLB miss. Searching for pid 1, vpn 0. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[2] is updated as it has the smallest last used tick value. TLB[0] is updated because it is the first entry found with a valid bit of 0 after a context switch occurred.

Tick 62: TLB miss. Searching for pid 1, vpn 9. The IPT is then searched, but still incurs a page fault. A page in and page out is done and IPT[3] is updated as it has the smallest last used tick value. TLB[1] is updated because it is the first entry found with a valid bit of 0 after a context switch occurred.

Tick 64: TLB miss. Searching for pid 1, vpn 26. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[1] is updated as it has the smallest last used tick value. TLB[2] is updated because it is the first entry found with a valid bit of 0 after a context switch occurred.

Tick 69: TLB miss. Searching for pid 1, vpn 1. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[0] is updated as it has the smallest last used tick value. TLB[0] is updated because it is the oldest entry to be updated.

Tick 74: TLB miss. Searching for pid 1, vpn 0. The IPT is then searched and a matching entry is found in IPT[2]. TLB[1] is updated because it is the oldest entry to be updated.

Tick 117: TLB miss. Searching for pid 0, vpn 0. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[0] is updated as it is the first entry with a valid bit of 0 found after pid 1 went dead. TLB[0] is updated because it is the first entry found with a valid bit of 0 after a context switch occurred.

Tick 120: TLB miss. Searching for pid 0, vpn 9. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[1] is updated as it is the first entry with a valid bit of 0 found after pid 1 went dead. TLB[1] is updated because it is the first entry found with a valid bit of 0 after a context switch occurred.

Tick 122: TLB miss. Searching for pid 0, vpn 10. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[2] is updated as it is the first entry with a valid bit of 0 found after pid 1 went dead. TLB[2] is updated because it is the first entry found with a valid bit of 0 after a context switch occurred.

Tick 123: TLB miss. Searching for pid 0, vpn 26. The IPT is then searched, but still incurs a page fault. A page in is done and IPT[3] is updated as it is the first entry with a valid bit of 0 found after pid 1 went dead. TLB[0] is updated because it is the oldest entry to be updated.

Tick 125: TLB miss. Searching for pid 0, vpn 0. The IPT is then searched and a matching entry is found in IPT[0]. TLB[1] is updated because it is the oldest entry to be updated.

**Statistics**

Page size: 128B

Number of physical frames: 4

TLB size: 3

Number of pages used: 5

Number of TLB misses: 19

Number of page faults: 14

Number of page outs: 2