2021-NTU-SCSE-CZ2005

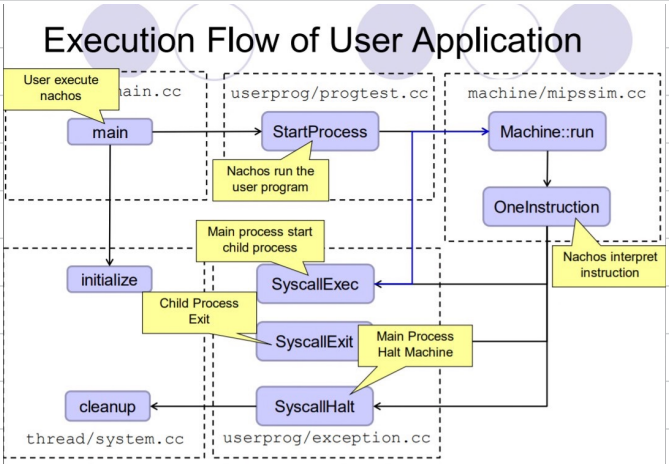
Operating Systems

Laboratory Implementation **4** Guide

Prepared by: Nailah Gucon

User Application

```
int main() {
    int s;
    int rc;
    s = Exec("../test/vm.noff");   → Execute child process
    rc = Join(s);
    Halt();
}
```



Translate a virtual address into a physical address

→ Translation Look-aside Buffer (TLB) : one per machine

→ Inverted Page Table (IPT) : one for the entire system

Nachos Terminology :

Page number == vpn

Frame number == phyPage

Translate (vpn, &phyAddr...)

→ Calculate Virtual Page No (VPN) → Lookup VPN on TLB

- if successful, calculate physical address
- if unsuccessful, TLB miss, generate exception → Call UpdateTLB (vpn) → call again

UpdateTLB (vpn)

:

```
if (possible_badVAddr)   → get bad addr from correct location
    badAddr = possible_badVAddr   → fault in kernel
else
    badAddr = machine->registers[BadVAddrReg];   → fault in User prog

vpn = badVAddr / PageSize;
:
```

Compare PhyPage with vpnToPhyPage(vpn)

- if return a valid phyPage → Update TLB : Call insertToTLB
- if not, perform paging and update TLB : PageOut PageIn (vpn)
  - Determine victim frame using least recently used algorithm
  - Page out victim page
    - Write victim page to swap file
  - Page in the new page
    - Load new physical page from swap file
    - Update the IPT table with VPN/PhyPage combination

Exercise

1. Change the working directory to Experiment 4 by typing `cd ~/nachos-exp3-4/vm`.
2. Modify the following methods in file `tlb.cc`. You may need to reference the following files: `tlb.h, ipt.h, ipt.cc, machine/translate.h, machine/translate.cc, machine/machine.h, and machine/disk.h`
    a. `int VpnToPhyPage(int vpn)` - Gets a physical frame phyPage for a virtual page vpn, if exists in the IPT.
    b. `void InsertToTLB(int vpn, int phyPage)` - Insert a vpn/phyPage entry into the TLB.
    c. `int lruAlgorithm(void)` – Return the freed physical frame according to the least recently used algorithm.
3. Compile Nachos by typing `make`. You should see "`ln -sf arch/intel-i386-linux /bin/nachos nachos`" at the end of the compiling output. If error, type `make clean` to remove all object & executable files and then type `make` again.
4. Execute the test program to test whether the virtual memory is working properly by typing `./nachos -x ../test/vmtest.noff -d > output.txt`.

5. Fill the table whenever there is a TLB miss (i.e., the matching entry is not found in the TLB, but it could be found in the IPT) or an IPT miss (i.e., the requested page is not in the memory at all; it must be a page fault and must trigger page replacement).

   The first 3 columns are the tick that the page fault exception occurred (tick), the virtual page number (vpn) and the corresponding process id (pid). Each process has its own set of virtual page numbers. The next four columns represent each entry in the IPT. Each IPT entry has four values, the process id (pid), virtual page number (vpn), last accessed tick (last used) and valid flag. The next three columns represent each entry in the TLB. Each TLB entry shows the virtual page number (vpn), physical frame number (phy), and the valid flag of the entry. The last column records the dirty page that is paged out, if any. You should complete Table1.csv file (template is provided) and record down the entries of the IPT and TLB before replacement. Please also save the template as a pdf file and highlight the entry that is selected to be updated. Do not highlight cells in the csv file and also do not add any new column header to Table1.csv. Any row which does not have a page out is filled with N.

6. Complete Table2.csv file (template is provided) to record page size, the number of physical frames, and the TLB size defined in Nachos as well as the number of pages used by the test program and the number of TLB misses, page faults and page outs that occurred during the execution of the test program.

Exercise Code:

⇒ IPT is realised as memory table → used by clock algo to choose phyPages when things need to swapped in; Lots of fields for efficiency; one class per page frame

```
//----------------------------------------------------------------------
// VpnToPhyPage
//      Gets a phyPage for a vpn, if exists in ipt.
//----------------------------------------------------------------------

int VpnToPhyPage(int vpn)
{
  int i;
  //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

  for(i =0; i<NumPhysPages; i++){
     if(memoryTable[i].valid && memoryTable[i].pid == currentThread->pid && memoryTable[i].vPage == vpn){
        return i;
     }
  }
  return -1;
}
```

→ inverted page table

→ memory table has as many entries as there are physical pages

→ find corresponding physical page

→ no entry found

```
//----------------------------------------------------------------------
// PageOutPageIn
//      Calls DoPageOut and DoPageIn and handles memoryTable
// bookkeeping. Use lru algorithm to find the replacement page.
//----------------------------------------------------------------------

int PageOutPageIn(int vpn)
{
  int phyPage;

  //increase the number of page faults
  stats->numPageFaults++;
  //call the LRU algorithm, which returns the freed physical frame
  phyPage=lruAlgorithm();

  //Page out the victim page to free the physical frame
  DoPageOut(phyPage);
  //Page in the new page to the freed physical frame
  DoPageIn(vpn, phyPage);

  //update memoryTable for this frame
  memoryTable[phyPage].valid=TRUE;
  memoryTable[phyPage].pid=currentThread->pid;
  memoryTable[phyPage].vPage=vpn;
  memoryTable[phyPage].dirty=FALSE;
  memoryTable[phyPage].TLBentry=-1;
  memoryTable[phyPage].lastUsed=0;
  memoryTable[phyPage].swapPtr=currentThread->space->swapPtr;

  return phyPage;
}
```

```
// Put a vpn/phyPage combination into the TLB. If full, use FIFO replacement

void InsertToTLB(int vpn, int phyPage)
{
  int i = 0; //entry in the TLB
  static int FIFOPointer = 0;

  printf("\n--- [TLB] ---\n");
  for(int x = 0; x < TLBSize; x++)
    { printf("TLB[%i]: vpn=%i, phy=%i, valid=%i\n", x, machine->tlb[x].virtualPage, machine->tlb[x].physicalPage, machine->tlb[x].valid); }
  printf("--- [TLB] ---\n"); printf("\n");

  //We used i as a means to iterate to find the ith entry that is oldest or empty;
  //your code to find an empty in TLB or to replace the oldest entry if TLB is full

  for(i=0; i< TLBSize; i++){
      if(!machine->tlb[i].valid)
        break;
  }

  //If there deems to have no invalid entries, we start back from index 0, as the inserting index, to push the old index out.
  if(i == TLBSize){
     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++;

}
```

```
// Machine::Translate
//   Translate a virtual address into a physical address, using
//  either a page table or a TLB.  Check for alignment and all sorts
//  of other errors, and if everything is ok, set the use/dirty bits in
//  the translation table entry, and store the translated physical
//  address in "physAddr".  If there was an error, returns the type
//  of the exception.
```

```
//---------------------------------------------------------------------
// lruAlgorithm
//      Determine where a vpn should go in phymem, and therefore what
// should be paged out. This lru algorithm is the one discussed in the
// lectures.
//---------------------------------------------------------------------

int lruAlgorithm(void)
{
  //your code here to find the physical frame that should be freed
  //according to the LRU algorithm.
  int phyPage = 0;
  int i=0;

  //Check if there are any invalid entries, if there are any invalid entries, return it to be used;
  for(i=0; i< NumPhysPages; i++){
      if(!memoryTable[i].valid){
        return i;
      }
  }

  if (i==NumPhysPages){
     //We will be comparing the last used against each other, therefore, any variable can be used for last used, since we checking
     //it in a for loop.
     int lastTick = memoryTable[0].lastUsed;
     int storeLast = 0;
     for (i=0; i<NumPhysPages; i++){
        if(memoryTable[i].lastUsed < lastTick){
           lastTick = memoryTable[i].lastUsed;
           storeLast = i;
        }
     }
     phyPage = storeLast;
  }

  return phyPage;
}
```

⇒ Need to find the least recently used entry in memoryTable.

⇒ The last tick that the physical page is accessed, is stored in memoryTable[i].lastUsed.

⇒ Search for invalid entry from beginning of the memoryTable. If there is an invalid entry, return that to be used by the virtual page → !memoryTable[i].valid

⇒ Otherwise, find a victim (entry with smallest lastUsed) → return page number

Sample Output:

```
== Tick 26 ==
        interrupts: on -> off
Time: 26, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b454(83a4098)
Interrupt handler timer, scheduled at 96
In mapcar, about to invoke 804b454(83a4580)
Interrupt handler console read, scheduled at 110
End of pending interrupts
        interrupts: off -> on
Reading VA 0x30, size 4
        Translate 0x30, read: *** no valid TLB entry found for this virtual page 0!
Exception: page fault/no TLB entry

--- [IPT] ---
IPT[0]: pid=0, vpn=0, last used=12, valid=1
IPT[1]: pid=0, vpn=9, last used=25, valid=1
IPT[2]: pid=0, vpn=26, last used=17, valid=1
IPT[3]: pid=0, vpn=1, last used=22, valid=1
--- [IPT] ---

Current VPN:0

--- [TLB] ---
TLB[0]: vpn=1, phy=3, valid=1
TLB[1]: vpn=9, phy=1, valid=1
TLB[2]: vpn=26, phy=2, valid=1
--- [TLB] ---

The corresponding TLBentry for Page 0 in TLB is 1


== Tick 27 ==
        interrupts: on -> off
Time: 27, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b454(83a4098)
Interrupt handler timer, scheduled at 96
In mapcar, about to invoke 804b454(83a4580)
Interrupt handler console read, scheduled at 110
End of pending interrupts
        interrupts: off -> on
Reading VA 0x30, size 4
        Translate 0x30, read: phys addr = 0x30
        value read = 24020002
At PC = 0x30: ADDIU r2,r0,2
```

```
== Tick 28 ==
        interrupts: on -> off
Time: 28, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b454(83a4098)
Interrupt handler timer, scheduled at 96
In mapcar, about to invoke 804b454(83a4580)
Interrupt handler console read, scheduled at 110
End of pending interrupts
        interrupts: off -> on
Reading VA 0x34, size 4
        Translate 0x34, read: phys addr = 0x34
        value read = 0000000c
At PC = 0x34: SYSCALL
Exception: syscall
SyscallExec(), initiated by user program main #0.
        Translate 0x528, read: *** no valid TLB entry found for this virtual page 10!

--- [IPT] ---
IPT[0]: pid=0, vpn=0, last used=28, valid=1
IPT[1]: pid=0, vpn=9, last used=25, valid=1
IPT[2]: pid=0, vpn=26, last used=17, valid=1
IPT[3]: pid=0, vpn=1, last used=22, valid=1
--- [IPT] ---

Current VPN:10
paging out: pid 0, phyPage 2, vpn 26
paging in: pid 0, phyPage 2, vpn 10

--- [TLB] ---
TLB[0]: vpn=1, phy=3, valid=1
TLB[1]: vpn=0, phy=0, valid=1
TLB[2]: vpn=26, phy=2, valid=0
--- [TLB] ---


The corresponding TLBentry for Page 10 in TLB is 2      Translate 0x528, read: phys addr = 0x128
        Translate 0x529, read: phys addr = 0x129
        Translate 0x52a, read: phys addr = 0x12a
        Translate 0x52b, read: phys addr = 0x12b
        Translate 0x52c, read: phys addr = 0x12c
        Translate 0x52d, read: phys addr = 0x12d
        Translate 0x52e, read: phys addr = 0x12e
        Translate 0x52f, read: phys addr = 0x12f
        Translate 0x530, read: phys addr = 0x130
        Translate 0x531, read: phys addr = 0x131
        Translate 0x532, read: phys addr = 0x132
        Translate 0x533, read: phys addr = 0x133
        Translate 0x534, read: phys addr = 0x134
        Translate 0x535, read: phys addr = 0x135
        Translate 0x536, read: phys addr = 0x136
        Translate 0x537, read: phys addr = 0x137
        Translate 0x528, read: phys addr = 0x128
        Translate 0x529, read: phys addr = 0x129
        Translate 0x52a, read: phys addr = 0x12a
        Translate 0x52b, read: phys addr = 0x12b
        Translate 0x52c, read: phys addr = 0x12c
        Translate 0x52d, read: phys addr = 0x12d
        Translate 0x52e, read: phys addr = 0x12e
        Translate 0x52f, read: phys addr = 0x12f
        Translate 0x530, read: phys addr = 0x130
        Translate 0x531, read: phys addr = 0x131
        Translate 0x532, read: phys addr = 0x132
        Translate 0x533, read: phys addr = 0x133
        Translate 0x534, read: phys addr = 0x134
        Translate 0x535, read: phys addr = 0x135
        Translate 0x536, read: phys addr = 0x136
        Translate 0x537, read: phys addr = 0x137
Initializing address space, num pages 27, size 3456
Forking thread userprogram #1 with func=0x804d29b, arg=138037072, join=NO
        interrupts: on -> off
Putting thread userprogram #1 on ready list.
        interrupts: off -> on
```

Table1.csv:

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			
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	N
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	N
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	N
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	N
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	N
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	Y
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	N
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	N
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	N
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	N
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	Y
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	N
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	N
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	N
107	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	N
110	9	0	0,0,109,1	1,26,66,0	1,0,76,0	1,9,73,0	0,0,1	0,2,0	26,1,0	N
112	10	0	0,0,109,1	0,9,111,1	1,0,76,0	1,9,73,0	0,0,1	9,1,1	26,1,0	N
113	26	0	0,0,109,1	0,9,111,1	0,10,113,1	1,9,73,0	0,0,1	9,1,1	10,2,1	N
115	0	0	0,0,109,1	0,9,111,1	0,10,114,1	0,26,114,1	26,3,1	9,1,1	10,2,1	N

Table2.csv:

PageSize	128
Number of physical frames	4
TLB size	3
Ticks	117
Paging Faults	14
Paging Out	2
TLB Miss	19