6/18/2010

Project 3 Step 2 - Implement IPT

Can be implemented as an array
- has NumPhysPages entries

Still keep project 2 assumptions

Add IPT population code to:
- AddrSpace constructor
- Fork system call - add stack
- Exec system call (maybe?)

[wherever you have a memory BitMap::Find call]

The index position of the IPT is the physical page # - BitMap::Find value
WARNING: Don't use VPN as index to IPT

What data type for an IPT Entry?

- Page Table is type TranslationEntry
- TLB " " "

That's not sufficient for an IPT

- IPT needs 1 more item in each entry - process ID

process ID ⇓ a unique # for each process

You have 2 choices:

1. Copy & paste TranslationEntry code into a new class
   - change class name
   - add new data - processID
   - " " methods
2. Subclass TranslationEntry
   - don't put this in machine dir.

On a PageFaultException, populate TLB from your IPT.

- search your IPT for
  - valid bit is TRUE
  - VPN of BadVAddrReg matches VPN of IPT entry
  - process ID of ~~VPN~~ IPT entry matches the current process' process ID

myIPT.valid == TRUE

Result: All project 2 user programs will run.

Step 3 - Don't preload anything into memory

- Still lots of memory

Now you can have an IPT miss

Comment out/remove preloading code & IPT population code from step 2 ⇒ It gets moved to PageFaultException

AddrSpace constructor & Forksyscall

- no bitMap -> Find
- no ReadAt

From step 2, on a PageFaultException

⇓

TLB miss

Your code will look something like this

~~else if~~

```
int vpn= machine->ReadRegister
        (BadVAddrReg) /
        PageSize;
```

returns the ppn of the matching IPTentry ⇐ `int ppn = findIPTIndex(vpn);`

⇓

`updateTLB(ppn)`

In step 2, ppn will always be found.

For step 3, ~~change~~ the needed vpn may not exist in the IPT

- Have findIPTIndex return -1

```
int ppn = findIPTIndex(vpn)
    // Add for step 3
    if (ppn == -1) {
        // VPN not present in IPT
        // Need to load VPN into memory
        ppn = loadPageIntoIPT(vpn);
    }

    updateTLB(ppn);
```

To handle IPT misses:
Look in the page table for the current process to get its disk location (if on disk)

The Nachos page table does not have this - you must make a new class

Where virtual pages are initially:
executable: code & init data
not on disk: stack & uninit data

ReadAt — no ReadAt

What is the executable location for a virtual page?
noff.h. code.inFileAddr + (vpn * PageSize)

store this in page table ⇒ in AddrSpace constructor

Page Table Adds

- "in executable"
- "in swap File"
- "neither"

} maybe one data item

- File byte offset

In AddrSpace constructor, you populate these fields instead of preloading into memory

AND... Fork syscall (stack)

Result: A memory BitMap::Find will only happen on a PageFaultException

One last thing - on a n IPT miss, don't forget to update the TLB

Result: All project 2 tests run

Project 3 Step 4

Reduce NumPhysPages to 32

- Memory will fill up

On a PageFaultException, on an IPT miss, now inside LoadPageIntoIPT function your Find will return a -1 ⇒ no available memory

```
int ppn = bitMap->Find();
if ( ppn == -1 ){
```

```
*     ppn = evictAPage();
}
→ // Have a ppn
```

You must pick a page for eviction

- Random (-PRAND)
- FIFO (-PFIFO)

default is random

Implementing FIFO

FIFO relates to when a page of memory was allocated

It is NOT 0,1,2,...31,0,1,2,...31

You must use a queue data structure

- Every time a page of memory is allocated, that ppn is appended to queue

When you must evict a page, you select first entry from the queue.

- Add the same ppn to the back of the queue

On a bitMap->Clear(ppn), remove that entry from the queue

A page selected for eviction might be dirty. We have to write the page to the swap file (and at some point load back into memory)

Any dirty page to be evicted, must have its page table entry updated to show disk location

Strong Hints

① Look in StartProcess, near the bottom

// delete executable

~~Don't~~ Executable must stay open as long as a process exists

- Make sure you save the 'executable' variable

executable->ReadAt(____)

is to occur on a PageFault Exception for virtual pages residing in the executable

(2) You must define/declare your swap file
- There is only 1 swap file for Nachos

Make sure you declare & initialize your swap file BEFORE you run any user programs
- open swap file in Initialize() in system.cc

Result: All user programs from project 2 EXCEPT your Carl's Jr big test.

The "best" way to know if programs are running properly : sort or matmult

1023 7220

After step 4, matmult might take upto a few minutes sort can take more than 10 minutes

Requirement: 2 Execs of matmult/sort
2 Forks " "

One last issue:

Nachos does not know about your IPT

Nachos only populates dirty bits in the TLB

Whenever you modify a valid TLB entry, you must propagate the dirty bit to your IPT

- context switch
- on a PageFaultException when updating the TLB

```
if( machine->TLB[currentTLB].valid){
  myIPT[ machine->TLB[currentTLB]
        .physicalPage].dirty
      = machine->TLB[currentTLB]
        .dirty
}
```

Part 3- Nachos Networking

Compile & run from network dir

Network directory Makefile is set to use the TLB

~ line 15 of Makefile
a list of DEFINES
-DUSERPROG (-DVM -DUSE_TLB) ...

remove them

do: gmake clean
gmake

You'll be able the pagetable from project 2.

When you integrate parts 1&2 with part 3 put the DEFINES back.

Implement Remote Procedure Calls for Monitors - Locks, CVs, & monitor variables

- 1 Server Nachos
  - Handle & monitor RPC requests from clients
  - Does not run Nachos user programs

- Up to 5 Nachos clients
  - Issue lock/CV/MV requests to the server
  - Each client has only 1 single-threaded user program

In network directory, a file nettest.cc, has a simple Nachos test-kernel test code

To run this code, you need 2 shell windows. You run Nachos in each window

My machine ID is 0

I'm sending to machine ID 1

You must start both Nachos instances within 3 seconds of each other

-m parameter is the machine ID of that Nachos instance

-o parameter is the machine ID of the other Nachos

The Nachos machine ID is like an IP Internet address

You must the -m parameter whenever doing nachos networking

Nachos networking is just writing/reading to a file
SOCKET0 ← machine ID

To Send/Receive

post office -> Send
-> Receive

Nachos has a mailbox number which is equivalent to the Internet port #
- By default Nachos allows 10 mailbox numbers.

Sending/Receiving require machine ID & mailbox #

By Wednesday,

- Run the network tests
  - Look @ how to make & parse & send/receive messages

- Take each syscall, to be made into RPCs, & design your message formats
  - what data does server need to carry out the request?
  - ~~Is th~~ What is the reply msg format?