



COMP3231/9201/3891/9283 Operating Systems 2018/s1

UNSW

Administration

- [Notices](#)
- [Course Outline](#)
- [UNSW Timetable](#)
- [Consultations](#)
- [Group Nomination](#)
- [Survey Results!!](#)

Work

- [Lectures](#)
- [Tutorials](#)
- [Extended Lectures](#)

Support

- [Piazza Forums](#)
- [Wiki](#)

Assignments

- [Submission](#)

Guidelines

- [Style Guide](#)
- [Warm-up Exercise](#)

(Bonus)

- [Assignment 1](#)
- [Assignment 2](#)
- [Assignment 3](#)

Resources

OS/161

- [General](#)
- [Man Pages](#)
- [Sys161 Pages](#)

Subversion

- [Intro Video](#)
- [Main website](#)
- [Subversion Book](#)

C coding

- [Info Sheet](#)
- [FAQ](#)

Debugging

- [Quickstart](#)
- [GDB and OS/161](#)

General

-

[Programming/debugging](#)

FAQ

- ["Hardware" Guide](#)
- [R3000 Reference](#)

[Manual](#)

- [Intro. to Prog.](#)

[Threads](#)

- [Unix Manual](#)

Previous years

- [2017 S1](#)
- [2016 S1](#)
- [2015 S1](#)
- [2014 S1](#)
- [2013 S1](#)
- [2012 S1](#)

Assignment 3: Virtual Memory

Contents

1. [Due Dates and Mark Distribution](#)
2. [Introduction](#)
3. [Setting Up](#)
4. [Tutorial Exercises](#)
5. [Coding Assignment](#)
6. [Submission](#)
7. [Advanced Assignment](#)

1. Due Dates and Mark Distribution

Due Date: 23:59:59, Sunday 3st June

Marks: The base assignment is worth 30 marks (of the 100 available for the class mark component of the course)

The 10% bonus for one week early applies.

The advanced assignment is compulsory for COMP3891/COMP9283 students. Only a subset of the advanced assignment is required. **Marks are capped at 5 marks of the 100 class marks available**. The paging component is for optional bonus marks only.

COMP3231/COMP9201 students can do the advanced part with the permission of the lecturer, and only if basic assignment is completed a week prior to the deadline. Marks obtained bonus marks that can make up for any shortfall in the class component, up to a maximum of 10 for the course across all assessments.

The familiarisation questions contained herein are the subject of your week 11 tutorial. Please answer the questions and bring them to your tutorial.

2. Introduction

- [2011 S1](#)
- [2010 S1](#)
- [2009 S1](#)
- [2008 S1](#)
- [2007 S1](#)
- [2006 S1](#)
- [2005 S2](#)
- [2005 S1](#)
- [2004 S2](#)
- [2004 S1](#)

Staff

- [Kevin Elphinstone](#)
- (LiC)
- [Alex Kroh \(Admin\)](#)

Grievances

- [Student Reps](#)



In this assignment you will implement the virtual memory sub-system of OS/161. The existing VM implementation in OS/161, *dumbvm*, is a minimal implementation with a number of shortcomings. In this assignment you will adapt OS/161 to take full advantage of the simulated hardware by implementing management of the MIPS software-managed Translation Lookaside Buffer (TLB). You will write the code to manage this TLB. You will also write code to manage system memory.

The System/161 TLB

In the System/161 machine, each TLB entry includes a 20-bit virtual page number and a 20-bit physical page number as well as the following five fields:

- global: 1 bit; if set, ignore the PID bits in the TLB.
- valid: 1 bit; set if the TLB entry contains a valid translation.
- dirty: 1 bit; enables writing to the page referenced by the entry; if this bit is 0, the page is only accessible for reading.
- nocache: 1 bit; unused in System/161. In a real processor, indicates that the hardware cache will be disabled when accessing this page.
- pid: 6 bits; a context or address space ID that can be used to allow entries to remain in the TLB after a context switch.

All these bits/values are maintained by the operating system. When the valid bit is set, the TLB entry contains a valid translation. This implies that the virtual page is present in physical memory. A TLB miss occurs when no TLB entry can be found with a matching virtual page and address space ID (unless the global bit is set in which case the address space ID is ignored) and a valid bit that is set.

For this assignment, you may ignore the pid field. Note, however, that you must then flush the TLB on a context switch (why?).

The System/161 Virtual Address Space Map

The MIPS divides its address space into several regions that have hardwired properties. These are:

- kseg2, TLB-mapped cacheable kernel space
- kseg1, direct-mapped uncached kernel space
- kseg0, direct-mapped cached kernel space
- kuseg, TLB-mapped cacheable user space

Both direct-mapped segments map to the first 512 megabytes of the physical address space.

The top of kuseg is 0x80000000. The top of kseg0 is 0xa0000000, and the top of kseg1 is 0xc0000000.

The memory map thus looks like this:

Address	Segment	Special properties
0xffffffff	kseg2	
0xc0000000		
0xbfffffff	kseg1	
0xbfc00180		Exception address if BEV set.
0xbfc00100		UTLB exception address if BEV set.
0xbfc00000		Execution begins here after processor reset.
0xa0000000		
0x9fffffff	kseg0	
0x80000080		Exception address if BEV not set.
0x80000000		UTLB exception address if BEV not set.
0x7fffffff	kuseg	
0x00000000		

3. Setting Up Assignment 3

You will (again) be setting up the VCS repository that will contain your code. Remember to use a 3231 subshell (or continue using your modified PATH) for this assignment, as outlined in ASST0. Also, double-check you have retained your umask from ASST1 and ASST2.

```
% umask
0007
```

If not, you have done something wrong with your umask setup and need to fix it or you will have trouble sharing files in your group account directory.

Only one group member should do the following. However, it would be beneficial if a group sets up the repo together to understand the process.

Like assignment 1, you will be using a repository located in your group account directory (/home/osprjXXX) for this assignment. Initialise this repository now using the appropriate VCS guide.

For SVN users, see

https://wiki.cse.unsw.edu.au/cs3231cgi/SVNrec#Repository_sharing.

For git users, see

https://wiki.cse.unsw.edu.au/cs3231cgi/GitGuide#Repository_sharing.

Like the other assignments, the sources can be found at `/home/cs3231/assigns/asst3/src`. All group members should now check out a private working copy to `/home/$USER/cs3231/asst3-src`.

You will also need to increase the amount of physical memory to run some of the provided tests. Update `~/cs3231/root/sys161.conf` so that the `ramsize` is as follows

```
31 mainboard ramsize=16777216 cpus=1
```

Or, download a fresh, appropriately configured, version from [here](#), and install it.

You are now ready to start the assignment.

Configure OS/161 for Assignment 3

Remember to set your `PATH` environment variable as in previous assignments (e.g. run the `3231` command).

Before proceeding further, configure your new sources, and build and install the user-level libraries and binaries.

```
% cd ~/cs3231/asst3-src
% ./configure
% bmake
% bmake install
```

You have to reconfigure your kernel before you can use the framework provided to do this assignment. The procedure for configuring a kernel is the same as before, except you will use the `ASST3` configuration file:

```
% cd ~/cs3231/asst3-src/kern/conf
% ./config ASST3
```

You should now see an `ASST3` directory in the compile directory.

Building for ASST3

When you built OS/161 for `ASST0`, you ran `bmake` from `compile/ASST0`. When you built for `ASST1`, you ran `bmake` from `compile/ASST1` ... you can probably see where this is heading:

```
% cd ../compile/ASST3
% bmake depend
% bmake
% bmake install
```

If you now run the kernel as you did for previous assignments, you should get to the menu prompt. If you try and run a program, it will fail with a message about an unimplemented feature (specifically referring to the `as_*` functions you must write). For example, run `p`

/bin/true at the OS/161 prompt to run the program /bin/true in ~/cs3231/root.

You are now ready to start the assignment.

4. Tutorial Exercises

Please answer the following questions and bring them to your tutorial in week 11. You should be familiar enough with navigating the kernel source that you can find the answers to the below questions by yourself (Hint: use the *grep* utility). You may also find the [MIPS r3000 reference](#) useful.

1. What is the difference between the different MIPS address space segments? What is the use of each segment?
2. What functions exist to help you manage the TLB? Describe their use. (Hint: look in kern/arch/mips/include/tlb.h)
3. What macros are used to convert from a physical address to a kernel virtual address?
4. What address should the initial user stack pointer be?
5. What are the entryhi and entrylo co-processor registers? Describe their contents.
6. What do the as_* functions do? Why do we need as_prepare_load() and as_complete_load()?
7. What does vm_fault() do? When is it called?
8. Assuming a hashed page table (4k pages), show for the following virtual addresses:
 1. The page number and offset;
 2. the translated address (after any page allocation); and
 3. the contents of the page table after the TLB miss.

The page table is initially empty. You may assume that the allocator returns frames in order, so that the first frame allocated is frame 1, then frames 2, 3, 4, etc.

You can assume the page table hash function simply returns the page number as a result of hashing.

- o 0x100008
- o 0x101008
- o 0x1000f0
- o 0x41000
- o 0x41b00

- 0x410000

5. Coding Assignment

This assignment involves designing and implementing a number of data-structures. Before you start, you should work out what data you need to keep track of, and what operations are required.

Like previous assignments, you are required to submit a small design doc that identifies the major issues you tackled in this assignment, and also describes your solutions to these issues.

The document will be used to guide our markers in their evaluation of your solution to the assignment. In the case of a poor results in the functional testing and poor documented code, we will base our assessment on these components alone. If you can't describe your own solution clearly, you can't expect us to reverse engineer the code to a poor and complex solution to the assignment.

Place your design document in `design.txt` (which we have created for you) at the top of the source tree to `OS/161` (i.e. in `~/cs3231/asst3-src/design.txt`).

When you later commit your changes into your repository, your design doc will be included in the commit, and later in your submission.

Also, please word wrap your design doc if you have not already done so. You can use the `unix fmt` command to achieve this if your editor cannot.

Memory Management

This assignment requires you to keep track of physical memory. The current memory management implementation in `dumbvm` never frees memory; your implementation should handle both allocation and freeing of frames.

You will need a *frametable* containing information about the memory available in the system. In the basic assignment you will need to keep track of whether a frame is used or not. In the advanced part, you will need to keep track of reference statistics and other information about the frames.

The functions that deal with memory are described in `kern/include/vm.h`. You may assume that only one page will be allocated at a time—designing a page allocator that can allocate multiple pages at a time is surprisingly tricky. However, make sure that you never allocate memory (though `kmalloc`) that is larger than a page!

Note that `alloc_kpages()` should return the virtual address of the page, i.e., an address in `kseg0`.

Warning: `alloc_kpages()` can be called before `vm_bootstrap()`. The means that your implementation of `alloc_kpages()` must work before your `frametable` is initialised. You should just call `ram_stealmem()` if the `frametable` hasn't been initialised.

Address Space Management

OS/161 has an address space abstraction, the `struct addrspace`. To enable OS/161 to interact with your VM implementation, you will need to fill in the functions in `kern/vm/addrspace.c`. The semantics of these functions is documented in `kern/include/addrspace.h`.

You may use a fixed-size stack region (say 16 pages) for each process.

Address Translation

The main goal for this assignment is to provide virtual memory translation for user programs. To do this, you will need to implement a TLB refill handler. You will also need to implement a page table. For this assignment, you will implement a **hashed page table (HPT)**.

Note that a hashed page table is a fixed sized data structure allocated at boot time and shared between all processes. We suggest sizing the table to have twice as many entries as there are frames of physical memory in RAM.

Given the HPT is a shared data structure, it will have to handle concurrent access by multiple processes. You'll need to synchronise access to avoid potential races. A single lock is satisfactory approach.

Each entry in the HPT typically has a process identifier, the page number, a link to handle collisions, and a frame number and permissions in `EntryLo` format for faster TLB loading. A HPT entry should not need to exceed 4 32-bit words.

See the wiki for a sample hash function (you're free to tune what we suggest). Hash collisions are always possible even with a good hash function, and you can use either internal or external chaining to resolve collisions. We suggest external chaining to avoid the HPT filling in the presence of sharing (in the advanced/bonus assignments), however internal chaining is sufficient for the basic assignment.

One can use the OS/161 address space pointer (of type `struct addrspace`) as the value of the current process ID. It is readily accessible where needed, and is unique to each address space.

The following questions may assist you in designing the contents of your page table

- What information do you need to store for each page?
- How does the page table get populated?
- When are frames allocated to back pages.

Note: Applications expect pages to contain zeros when first used. This implies that newly allocated frames that are used to back pages should be zero-filled prior to mapping.

Testing and Debugging Your Assignment

To test this assignment, you should run a process that requires more virtual memory than the TLB can map at any one time. You should also ensure that touching memory not in a valid region will raise an exception. The huge and faultier tests in testbin may be useful. See the Wiki for more options.

Apart from GDB, you may also find the `trace161` command useful. `trace161` will run the simulator with tracing, for example

```
% trace161 -t t -f outfile kernel
```

will record all TLB accesses in outfile.

Don't use `kprintf()` for `vm_fault()` debugging. See Wiki for more info.

Hints

The assignment consists of two components. Extending the memory management with a frametable to track used and free frames, and implementing a page table to support virtual memory. You can tackle these independently.

One could start on the frame table, and `alloc_kpage()` and `free_kpage()`. Note that `alloc_kpage()` and `free_kpage()` are the underlying routines used by `kmalloc()` and `kfree()`, so ensure your implementation is correct. A broken `alloc_kpage()` will result in a broken `kmalloc()` and an unpredictably broken OS/161! See the Wiki to some testing suggestions.

To implement a page table, have a close look at the `dumbvm` implementation, especially `vm_fault()`. Although it is simple, you should get an idea on how to approach the rest of the assignment.

One approach to implementing the assignment is in the following order:

1. Identify an approach to allocate the memory that will become your frame table and implement it.

2. Initialise the frame table (`vm_bootstrap` is a plausible location to call the code).
3. Write routines in `kern/vm/rametable.c` to manage free frames and allocate pages. Note: the frame table is a global resource, therefore you'll need to deal with concurrency. It is okay to use the spinlock already in place for *stealmem*, or you can use interrupt disabling/enabling for this table.
4. Understand how the page table works, and its relationship with the TLB.
5. Understand the specification and the supplied code.
6. Work out a basic design for your page table implementation.
7. Modify `kern/vm/vm.c` to insert , lookup, and update page table entries, and keep the TLB consistent with the page table.
8. Implement the TLB exception handlers in `vm.c` using your page table.
9. Implement the functions in `kern/vm/addrspace.c` that are required for basic functionality (e.g. `as_create()`, `as_prepare_load()`, etc.). Allocating user pages in `as_define_region()` may also simplify your assignment, however good solution allocate pages in `vm_fault()`.
10. Test and debug this. Use the debugger!
If you really get stuck, submit at least this much of the solution, and you should get some marks for it.

Note: Interrupts should be disabled when writing to the TLB, see `dumbvm` for an example. Otherwise, unexpected concurrency issues can occur.

`as_activate()` and `as_deactivate()` can be copied from `dumbvm`.

6. Basic Assignment Submission

As with the previous assignments, you again will be submitting a `.diff` of your changes to the original tree.

You should first commit your changes back to the repository. Note: You will have to supply a comment on your changes. You also need to coordinate with your partner that the changes you have (or potentially both have) made are committed consistently by you and your partner, such that the repository contains the work you want from both partners. Refer to the appropriate wiki page for instructions.

For SVN users, see <https://wiki.cse.unsw.edu.au/cs3231cgi/SVNrec>.

For git users, see <https://wiki.cse.unsw.edu.au/cs3231cgi/GitGuide>.

Beware! If you have created new files for this assignment, they will not be included in your submission unless you add them to the repository.

Once your solution is committed, generate a diff.

Testing Your Submission

Look [here](#) for information on testing and resubmitting your assignment.

Submitting Your Assignment

Now submit the diff as your assignment.

```
% cd ~  
% give cs3231 asst3 asst3.diff
```

You're now done.

Even though the generated .diff file should represent all the changes you have made to the supplied code, occasionally students do something "ingenious" and generate non representative output.

We strongly suggest keeping your checkout intact to allow for recovery of your work if need be.

7. Advanced Assignment

The advanced assignment is for COMP3891/COMP9283 students and consist of a student-chosen subset of the problem below, and capped at 5 marks.

COMP3231/COMP9201 students can do the advanced part with the permission of the lecturer, and only if basic assignment is completed a week prior to the deadline. Marks obtained are added to any shortfall in the participation mark component. The limited bonus marks are also available.

- (easy) 3 marks Shared pages and copy-on-write.
- (easy) 2 marks Implement sbrk() to enable user-level malloc() to function.
- (hard) 5 marks.
 - Implement a simplified mmap() and munmap(). Note: you only need to support the simplified case of mapping a file, and munmap() the entire region that was mapped. The prototypes are expected to be

```
void *mmap(size_t length, int prot, int fd, off_t offset);  
int munmap(void *addr);
```

Where prot can be PROT_READ and/or PROT_WRITE.
Compared to traditional mmap, there are no flags, and the OS chooses the virtual address to locate the region. You

- must ensure that applications can open a file, updated it, and have the updated file propagate to the filesystem.
- And, implement demand-loading. You should load pages only when they are referenced by the user process, as opposed to at process creation.
- **Bonus section.** (seriously hard) 3 marks Implement paging. You should implement some page replacement algorithm and demonstrate your solution running under memory pressure.

Advanced Assignment Submission

Submission for the advanced assignment is similar to the basic assignment, **except the advance component is *given to a distinguished assignment name: asst3_adv***. Again, you need to generate a diff based on the original source tree.

Note: The advanced component will be automarked only. We'll make the tests we use to mark the advanced part available for your own testing. See the wiki for details.

Submit your solution

```
% cd ~  
% give cs3231 asst3_adv asst3_adv.diff
```

You're now done.

FAQ and Gotchas

See <https://wiki.cse.unsw.edu.au/cs3231cgi/2018s1/Asst3> for an up to date list of potential issues you might encounter.

Page last modified: 11:07am on Monday, 7th of May, 2018

[Print Version](#)

CRICOS Provider Number: 00098G