



# COMP3231/9201/3891/9283 Operating Systems 2021/T1

UNSW

## ASST3: Virtual Memory

### Table of Contents

- [Checklist](#)
- [Due Dates and Mark Distribution](#)
- [Introduction](#)
  - [The System/161 TLB](#)
  - [The System/161 Virtual Address Space Map](#)
- [Setting Up Assignment 3](#)
  - [Configure OS/161 for Assignment 3](#)
    - [Building for ASST3](#)
- [Coding Assignment](#)
  - [Address Space Management](#)
  - [Address Translation](#)
  - [Testing and Debugging Your Assignment](#)
  - [Hints](#)
- [FAQ, Gotchas and Video](#)
- [Basic Assignment Submission](#)
- [Advanced Assignment](#)
  - [Advanced Assignment Submission](#)

## Checklist

- Read this spec.
- Set up the assignment.
- Watch the assignment overview video.
- Do the week 9 tutorial exercises yourself.
- Attend the tutorial.
- Read the FAQ for assignment 3 on the wiki.
- Do the assignment.
- Don't forget to commit your most recent changes.
- Submit your assignment.

## Due Dates and Mark Distribution

**Due Date & Time:** 16:00 Friday 23rd April

**Extension: New Due Date & Time:** 16:00 Monday 26th April

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

The 2% per day bonus for each day early applies, capped at 10%, as per course outline.

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. Obtained bonus marks can make up for any shortfall in the class mark component, up to a maximum of 5 marks for this assignment.

There are familiarisation questions in your week 9 tutorial. Please answer the questions and bring them to your tutorial. Feel free to ask any assignment related questions need to clarify your understanding.

## Introduction

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.

## The System/161 TLB

In the System/161 machine, each TLB entry includes a 20-bit virtual page number and a 20-bit physical frame 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.
- asid: 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 (i.e. your code). 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 largely ignore the ASID field set to zero for your TLB entries. 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.
0x80000000		
0x7fffffff	kuseg	
0x00000000		

## Setting Up Assignment 3

We assume after ASST0, ASST1, and ASST2 that you now have some familiarity with setting up for OS/161 development. If you need more detail, refer back to ASST0.

Clone the ASST3 source repository from [gitlab.cse.unsw.edu.au](https://gitlab.cse.unsw.edu.au). Note: replace XXX with your 3 digit group number.

```
% cd ~/cs3231
% git clone https://zNNNNNNN@gitlab.cse.unsw.edu.au/COMP3231/21T1/grpXXX-asst3.git asst3-src
```

Note: The gitlab repository is shared between you and your partner. You can both push and pull changes to and from the repository to cooperate on 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 (the failure is due to the unimplemented `as_*` functions that you must write). For example, run `p /bin/true` at the OS/161 prompt to run the program `/bin/true` in `~/cs3231/root`.

```
OS/161 kernel [? for menu]: p /bin/true
Running program /bin/true failed: Function not implemented
Program (pid 2) exited with status 1
Operation took 0.173469806 seconds
OS/161 kernel [? for menu]:
```

**Note:** If you don't have a `sys161.conf` file, you can use the one from ASST1.

The simplest way to install it is as follows:

```
% cd ~/cs3231/root
% wget http://cgi.cse.unsw.edu.au/~cs3231/21T1/assignments/asst1/sys161.conf -O sys161.conf
```

You are now ready to start the assignment.

## Coding Assignment

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

## Address Space Management

OS/161 has an address space data type that encapsulates the book-keeping needed to describe an address space: the struct `addrspace`. To enable OS/161 to interact with your VM implementation, you will need to implement the functions in `kern/vm/addrspace.c` and potentially modify the data type. The semantics of these functions is documented in `kern/include/addrspace.h`.

Note: 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. Each process will have its own virtual address spaces which is mapped to physical memory. Note: You are not required to implement demand paging (that's an advanced task). To provide virtual memory, you will need to implement a TLB refill handler. You will also need to implement a per-process page table that is used to store the translations used to refill the TLB.

For this assignment, you will implement a **3-level (hierarchical) page table**. The first level of the page table must be indexed using the 8 most significant bits of the page number, the second-level nodes of the page table must be indexed using the next 6 bits of the page number, and the third-level page table nodes must be indexed using the least significant bits of the page number. **Thus the first-level node will have 256 ( $2^8$ ) entries, and the second-level and third-level nodes will have 64 ( $2^6$ ) entries.**

Note that a hierarchical page table is a *lazy data-structure*. This means that the contents of the page table, including the second-level and third-level nodes in the hierarchy, are only allocated when they are needed.

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?
- Is the data structure global or a per-process data structure?

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 the Wiki for more info.

## Hints

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:

- Review how the specified page table works from the lectures, and understand its relationship with the TLB.
- Review the assignment specification and its relationship with the supplied code.
  - *dumbvm* is not longer compiled into the OS/161 kernel for this assignment (kern/arch/mips/vm/dumbvm.c), but you can review it as an example implementation within the interface/framework you will be working within.
- Work out a basic design for your page table implementation.
- Modify kern/vm/vm.c to insert , lookup, and update page table entries, and keep the TLB consistent with the page table.
- Implement the TLB exception handler vm\_fault() in vm.c using your page table.
- 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().
- Test and debug this. Use the debugger!

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.

## FAQ, Gotchas and Video

Don't forget to look at <https://wiki.cse.unsw.edu.au/cs3231cgi/2021t1/Asst3> for an up to date list of potential issues you might encounter.

There is also an overview video on the assignment available on the lectures page in the course account <https://cgi.cse.unsw.edu.au/~cs3231/lectures.php>.

## Basic Assignment Submission

The submission instructions are available on the [Wiki](#). Like previous assignments, you will be submitting the git repository bundle via CSE's give system. For ASST3, the submission system will do a test build and run a simple test to confirm your bundle at least compiles.

**Warning! Don't ignore the submission system! If your submission fails the submission process, you may not receive any marks.**

**Warning! Don't forget to commit your changes prior to generating your bundle.**

To submit your bundle:

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

You're now done.

Even though the generated bundle should represent all the changes you have made to the supplied code, occasionally students do something "ingenious". So always keep your git repository so that you may recover your assignment should something go wrong.

## Advanced Assignment

The advanced assignment consists of a student-chosen subset of the problems below. The total marks available are capped at 5 marks.

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.

- (easy) 2 marks Shared pages and copy-on-write.
- (easy) 2 marks Implement `sbrk()` to enable user-level `malloc()` to function with more memory than its initially allocated pool.
- (hard) 3 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.
- (seriously hard) 5 marks Implement paging. You should implement some page replacement algorithm and demonstrate your solution running under memory pressure.

Given you're doing the advanced version of the assignment, I'm assuming you are competent with managing your git repository and don't need detailed directions. We expect you to work on a specific branch in your repository to both build upon your existing assignment, while keeping your advanced assignment separate at the same time.

Here are some git commands that will be helpful.

- One member of your group should create the branch and push it back to gitlab.

```
git checkout -b asst3_adv  
git push --set-upstream origin asst3_adv
```

- To switch back to the basic assignment at some point.

```
git checkout master
```

- To switch to the advanced assignment at another point.

```
git checkout asst3_adv
```

## Advanced Assignment Submission

Submission for the advanced assignment is similar to the basic assignment, **except the advance component is given to a separate assignment name: asst3\_adv**. Again, you need to generate a bundle based on your repository. Note: Our marking scripts will switch to the asst3\_adv branch prior to testing the advanced assignment.

Submit your solution

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

You're now done.

---

*Page last modified: 2:18pm on Friday, 16th of April, 2021*

[Screen Version](#)

CRICOS Provider Number: 00098G