Pages

TABLE OF CONTENTS

Allocating pages

Use the following functions in `threads/palloc.h` to allocate and deallocate pages in the Pintos kernel.

```
void* palloc_get_page(enum palloc_flags);
void* palloc_get_multiple(enum palloc_flags, size_t page_cnt);
void palloc_free_page(void* page);
void palloc_free_multiple(void* pages, size_t page_cnt);
```

The `palloc` functions use a bitmap to keep track of which pages are free and which pages are allocated. The `flags` argument is a bitmask of the following choices.

```
enum palloc_flags {
  PAL_ASSERT = 001, /* Panic on failure. */
  PAL_ZERO = 002, /* Zero page contents. */
  PAL_USER = 004 /* User page. */
};
```

The set of pages is partitioned into two separate *pools*: a user pool and a kernel pool. Pages in the kernel pool are meant for use inside the kernel (e.g. the stack for kernel threads) and pages in the user pool are meant to be mapped into the virtual address spaces of user processes. The reason for this separation is to prevent failures in the kernel if user programs run out of memory. The `PAL_USER` flag tells the `palloc` function to allocate the requested pages from the user pool. Otherwise, it will allocate the requested pages from the kernel pool. **If you intend to map a page into the virtual address space of a process, you should allocate it from the user pool using `PAL_USER`.**

Mapping a page into a virtual address space

The `pagedir` member of `struct thread` is a pointer to the page table of the process. When switching to a process, Pintos uses the `pagedir_activate` function to start using that process' page table for address translation, by setting the page directory base register (`%cr3`). Note that the entirety of kernel memory is mapped into every process' virtual address space at addresses `PHYS_BASE` and above, so all process' page tables are interchangeable as long as you are accessing kernel memory. For this reason, we sometimes refer to addresses at `PHYS_BASE` and above as *kernel virtual addresses*.

The physical address corresponding to a kernel virtual address can be computed by subtracting `PHYS_BASE` from it. In principle this need not be true, but Pintos sets up its page tables such that it holds. The functions `vtop` and `ptov` (in `threads/vaddr.h`) are helper functions that perform this conversion for you, but we do not expect you to have to use these functions in this assignment.

Given a page allocated in kernel virtual memory, one can map it into the virtual address space of a process by calling `pagedir_set_page`, which handles traversal of the two-level hierarchical page table and allocation of leaves as needed. Two arguments to `pagedir_set_page` are the virtual address in the user process at which the page should be mapped, and the kernel virtual address of the page to map. `pagedir_set_page` looks up the physical page number to create the necessary page table entries for you, **so you should be passing in the kernel virtual address rather than the physical page number.** Once you've mapped the page into the process' virtual address space, you don't have to worry about deallocating it; when the process exits, the `process_exit` function will call `pagedir_destroy`, which will call `palloc_free_page` on all pages mapped into the process' address space. If you would like the page to continue to be allocated even after the process dies, you should first remove it from the page table using `pagedir_clear_page`.

Note that pages that you allocate using `palloc_get_page` could have been previously allocated and then freed, and therefore could contain data from the previous time it was allocated. If it was mapped into a user process, it could contain data from the previous user program. To properly enforce protection, **you should initialize the contents of a page before mapping it into a user process. This is typically done by setting all bytes in the page to zero,** except in special situations where the page should contain specific data (e.g. loading new code into a process or bringing back a page from disk). Under no circumstances should memory used by the kernel, or by other processes, become visible to a process because a physical page frame was reused.

Summary and example

In summary, here is how to map a fresh page into the virtual address space of a process.

1. Allocate the page from the user pool using `palloc_get_page` and passing the `PAL_USER` flag.

2. Zero out the page's contents, either by using `memset` or passing the `PAL_ZERO` flag when allocating the page (e.g. `palloc_get_page(PAL_ZERO | PAL_USER)`).
3. Use `pagedir_set_page` to map the page into the virtual address space of a process.
4. The page will be deallocated when the process exits and `pagedir_destroy` is called. Alternatively, if you would like the page to be deallocated at some other time, you can remove it from the page table with `pagedir_clear_page` and then deallocate it later using `palloc_free_page`.

A simple example of this is in the `setup_stack` and `install_page` functions in `process.c`. **We strongly recommend that you review these functions and understand this simple example before attempting this homework.**
