Project 6: Clock Page Replacement

- Continue to build on code from project 5
  - No API changes for this project
- New twist: smaller memory, so `PhysMem::page_alloc` may fail
  - If no free pages, you must pick an existing `PPage` to evict
  - If the page is dirty, you write it back when evicting
  - Use clock algorithm to approximate LRU eviction
- Why would you want to do this?
  - Easily add encryption to program (e.g., database) that uses `mmap`
  - Don’t want kernel to page out secret plaintext to disk
    - To avoid paging, `PhysMem` uses `locked` physical memory
    - Systems can limit locked mem. (`ulimit -l`—e.g., 64 MiB on myth (`vm.cc` actually limits locking to 1 MiB to be polite))
Clock algorithm

- Idea: evict page that has not been accessed recently
  - In kernel, would use accessed (A) bit in page table entry (PTE)
  - We will emulate accessed bit by making pages inaccessible

- Idea: FIFO eviction, but skip accessed pages

- Keep pages in circular FIFO list

- Scan each physical pages:
  - Step “hand” from last evicted
    - Accessed (A = 1)? Don’t evict, but set $A \leftarrow 0$ (re-map PROT_NONE)
    - Not accessed (A = 0)? Evict

- A.k.a. second-chance replacement
Clock algorithm

- **Idea:** evict page that has not been accessed recently
  - In kernel, would use accessed (A) bit in page table entry (PTE)
  - We will emulate accessed bit by making pages inaccessible

- **Idea:** FIFO eviction, but skip accessed pages

- **Keep pages in circular FIFO list**

- **Scan each physical pages:**
  - Step “hand” from last evicted
    - Accessed (A = 1)? Don’t evict, but set $A \leftarrow 0$ (re-map PROT_NONE)
    - Not accessed (A = 0)? Evict

- **A.k.a. second-chance replacement**
Clock algorithm

- Idea: evict page that has not been accessed recently
  - In kernel, would use accessed (A) bit in page table entry (PTE)
  - We will emulate accessed bit by making pages inaccessible

- Idea: FIFO eviction, but skip accessed pages

- Keep pages in circular FIFO list

- Scan each physical pages:
  - Step “hand” from last evicted
  - Accessed (A = 1)? Don’t evict, but set $A \leftarrow 0$ (re-map PROT_NONE)
  - Not accessed (A = 0)? Evict

- A.k.a. second-chance replacement
Clock extensions (not for project)

- Large memory may be a problem
  - Most pages referenced in long interval

- Add a second clock hand
  - Two hands move in lockstep
  - Leading hand clears A bits
  - Trailing hand evicts pages with A=0

- Can also take advantage of dirty bit
  - Each page can be (Unaccessed, Clean), (Unaccessed, Dirty), (Accessed, Clean), or (Accessed, Dirty)
  - Consider clean pages for eviction before dirty

- Or use \( n \)-bit accessed \( count \) instead just A bit
  - On sweep: \( count = (A \ll (n - 1)) \mid (count \gg 1) \)
  - Evict page with lowest \( count \)
Clock extensions (not for project)

- **Large memory may be a problem**
  - Most pages referenced in long interval

- **Add a second clock hand**
  - Two hands move in lockstep
  - Leading hand clears A bits
  - Trailing hand evicts pages with A=0

- **Can also take advantage of dirty bit**
  - Each page can be (Unaccessed, Clean), (Unaccessed, Dirty), (Accessed, Clean), or (Accessed, Dirty)
  - Consider clean pages for eviction before dirty

- **Or use** $n$-bit accessed count instead just A bit
  - On sweep: $\text{count} = (A \ll (n - 1)) \mid (\text{count} \gg 1)$
  - Evict page with lowest count
Clock extensions (not for project)

- Large memory may be a problem
  - Most pages referenced in long interval

- Add a second clock hand
  - Two hands move in lockstep
  - Leading hand clears A bits
  - Trailing hand evicts pages with A=0

- Can also take advantage of dirty bit
  - Each page can be (Unaccessed, Clean), (Unaccessed, Dirty), (Accessed, Clean), or (Accessed, Dirty)
  - Consider clean pages for eviction before dirty

- Or use $n$-bit accessed $count$ instead just $A$ bit
  - On sweep: $count = (A \ll (n - 1)) \mid (count \gg 1)$
  - Evict page with lowest $count$
Implementation hints

• Use a global auxiliary page table, rather than per MCryptFile
  - Page fault in one file may evict page from a different file
  - Global variables often not great for modularity, but here process
    has only one address space, so a single Aux PT is appropriate

• Track PPages with a std::vector the size of PhysMem
  
  ```
  std::vector<something> frame_table;
  :
  // in set_memory_size:
  frame_table.resize(npages, nullptr);
  ```

  - something could be VPage, AuxPTE* (as you see fit)
  - Okay, because overhead is a fixed fraction of memory
  - Clock hand is counter mod npages

• Write the PPage alias back to disk
  - With clock, VPages will be PROT_NONE when evicted
  - Can pass the (always accessible) PPage to aligned_pwrite instead
C++ container range queries

- Often want to traverse container elements in range \([L, H]\)
  - Signifies *half-open interval* of all \(i\) such that \(L \leq i < H\)
  - Example: Given `VMRegion *r`, traverse Aux PT entries in the range: 
    \([r->map_base(), r->map_base() + r->nbytes_]\)

- **Use the `lower_bound` method to query range**
  - `c.lower_bound(v)` → lowest element \(\geq v\), or `end()` if none
  - Start traversal at `container.lower_bound(L)`
  - Stop traversal (don’t process) `container.lower_bound(H)`

- `upper_bound(H)` *is end of query for closed interval* \([L, H]\)
  - First element > \(H\) (or `end()` if none)
  - `upper_bound(H)-1` is mirror image of `lower_bound(L)`
  - If you have duplicate keys, then query half-open interval \([L, H]\) 
    from `lower_bound(L)` to `upper_bound(H)-1`

- **Demo:** `range.cc, dirty.cc (corrected)`
**C++ lambda this captures**

- **Capture** `this` to make use of current object in a lambda
  - `[this](){ some_method(); }`

- **Captures the pointer** `this`, **not the object** `*this`
  - Subsequent changes to the object will be reflected in lambda
  - But lambda becomes invalid when object destroyed

- You probably want to use this technique for page fault handler to capture the current `MCryptFile` object

- Can also implicitly capture this with `[](){...}` or ` [=](){...}`
  - Using ` [=]` to capture `this` is a warning as of C++20, so be explicit

- **Demo:** `capture.cc`
C++ exceptions

- **To throw an exception**: `throw e;`
  - `e` can be any type (probably a design mistake in C++)
  - By convention, use only subclasses of `std::exception`, has virtual function `char *what()` for human-readable info

- **Handle exceptions with try, then one or more catch blocks**
  - `catch (const T &t) { ... }` catches anything that is a `T`
  - Only the first matching `catch` block is used
  - Can end with `catch(...)` to catch anything
  - Can re-throw current exception with `throw;` (no `e`)
  - **Function-try-block** can catch exceptions in initializers

- **When exception escapes scope, destroys all variables**
  - Convenient to avoid resource links
  - But means you shouldn’t let exceptions escape destructors

- **Demo**: `throw.cc`
Exceptions vs. returning error

+ Error handling needs less typing w. exceptions (fewer bugs)
+ Only convenient way to return errors from constructors
+ Easy to convey extra information in exception

```cpp
std::unique_ptr<thing> openit(); // OK, no info if NULL
std::string openit(thing *); // OK, less convenient
// Returning thing and/or error string gets ugly:
std::pair<thing, std::string> openit();
std::variant<thing, std::string> openit();
```

- Much higher overhead than return value
  - Don’t use as distinguished return value (e.g., item not in set)
- Makes destructors more brittle
- Need some hacks to implement “finally”

Demo: `finally.cc` (unfinished demo from syncimpl section)
**void *mmap (void *addr, size_t len, int prot, int flags, int fd, off_t offset)**

- Map file specified by `fd` at virtual address `addr`
- If `addr` is NULL, let kernel choose the address

**prot – protection of region**

- OR of PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE

**flags**

- MAP_ANON – anonymous memory (`fd` should be –1)
- MAP_PRIVATE – modifications are private
- MAP_SHARED – modifications seen by everyone
More VM system calls

- int msync(void *addr, size_t len, int flags);
  - Flush changes of mmapped file to backing store
- int munmap(void *addr, size_t len);
  - Removes memory-mapped object
- int mprotect(void *addr, size_t len, int prot);
  - Changes protection on pages to bitwise or of some PROT_... values
- int mincore(void *addr, size_t len, char *vec);
  - Returns in vec which pages present
struct sigaction {
    union {
        /* signal handler */
        void (*sa_handler)(int); // Normal handler
        // But if you set SA_SIGINFO in sa_flags, can use this:
        void (*sa_sigaction)(int, siginfo_t *, ucontext_t *);
    };
    sigset_t sa_mask; /* signal mask to apply */
    int sa_flags;
};

int sigaction (int sig, const struct sigaction *act,
               struct sigaction *oact)

- Can specify function to run on SIGSEGV
  (Unix signal raised on invalid memory access)
  - siginfo_t::si_addr is faulting virtual address
  - Even more info in ucontext_t
  - Look at /usr/include/sys/ucontext.h
How `vm.cc` works

- **PhysMem** creates a temporary file for physical memory
  - Can’t create aliases for pages created MAP_ANON
  - Keeps file descriptor open but immediately deletes file
  - Uses `mmap` to map entire file (at `pool_base()`) with MAP_SHARED

- **VMRegion** maps whole anonymous region with PROT_NONE
  - Reserves virtual addresses, but doesn’t change kernel page tables

- **VMRegion::map** uses `mmap` to override on page mapping
  - Use `PPage` address to get offset in temporary file
  - Re-map `mmap` 4 KiB in file at `VPage` address with requested permissions, creating an alias for same physical page

- **VMRegion::map** uses `mmap` to request 4 KiB anonymous PROT_NONE at `VPage`

- **Signal handler** gets faulting address from `siginfo_t`
  - Uses global `itree` to find appropriate `VMRegion` and handler
Lesson: VM tricks not just for the kernel

- Combination of `mprotect/sigaction` very powerful
  - E.g., fault, unprotect page, return from signal handler
  - Can basically use OS VM tricks in user-level programs

- Technique used in object-oriented databases
  - Bring in objects on demand
  - Keep track of which objects may be dirty
  - Manage memory as a cache for much larger object DB

- Other interesting applications
  - Useful for some garbage collection algorithms
  - Snapshot processes (copy on write)

- Classic research paper advocating these techniques: [Appel]