Operating System (Fall 2014) Project 3

Virtual Memory

Jongwook Choi

Introduction

third pintos The goal of the project is Virtual Memory The will to implement. There are three requirements significantly below.

- **Paging:** Virtual memory user page Wow frame page By mapping, even if memory is low frame eviction And swapping Through (swap One that is allowed) The itgekkeum can use more memory.
- **Stack Growth:** Stack two or more areas page Even if something hits on, automatically, a memory error without born stack Assigned to the domain page By assigning stack This automatic grow So can.
- **Memory-Mapped Files:** By mapping the file into virtual memory fi le read / write Be a itgekkeum memory-mapped fi le Scheme and related system calls mmap (), munmap () To be implemented.

In the report above 3 Things that you have in the center, each designed in any manner the requirements and implementation issues that can occur and what they will be briefly whether I solve them.

1. Paging

1.1. Data Structure

To implement paging and virtual memory, it is necessary to implement some of the data structure. Things have implemented (1) Frame Table, (2) Supplemental Page Table, (3) Swap Table to be. First, a brief description of an implementation of the data structure and operation.

(1) Frame Table

Frame Iran physical memory The means area. A frame PGSIZE It has a size of both aligned It is. Paging all memory area is available frame table The management should be.

Frame table Add information in the Frame table entry It is one of frame page And exactly one exists for, and save the following information:

```
/ ** Frame Table Entry * /
struct frame_table_entry
   {
      void * Kpage;                          / * Kernel page, mapped to physical address * /

      struct hash_elem helem;                  / * See :: frame_map * /
      struct list_elem lelem;                  / * See :: frame_list * /

      void * Upage;                          / * User (Virtual Memory) Address, pointer to page * /
      struct thread * T;                     / * The associated thread. * /

      bool pinned;                           / * Used to prevent a frame from being evicted, while it is acquiring some resources.
                                                If it is true, it is never evicted. * /
   };
```

- kpage: Hash table Which is a key value of the mapped frame of kernel page An address. PintOS The physical address Wow kernel page The 1:01 Just because it corresponds to a simple kpage And unified physical address And it decided to handle.
- upage: Applicable frame The Road user page of virtual memory address.
- t: Applicable frame Loaded with, owner It refers to the thread.
- pinned: Frame pinning It is necessary to. It is described in detail below.
- helem, lelem: Frame hash table And linked list The need to put element Object.

Frame table Of course, global scope The only exists as. Support operations are as follows:

- void vm_frame_init (): reset. It called once at the beginning.
- void * vm_frame_allocate (enum palloc_flags, void * upage): user virtual address upage The corresponding frame page Generated by the one page mapping The performing and generating page frame of kernel address The returns.

- void vm_frame_free (void *): page frame The acts to release. Frame table The corresponding entry is removed from, the resource (page) It is released.
- vm_frame_pin (), vm_frame_unpin (): Pinning It is necessary to. It is described in detail below.

For the implementation of the above operation, internally (kpage 7→frame table entry) Hash Table (struct hash frame_map) It was used. Thus the key-value kpage That when given frame The information quickly lookup can do.

Aside from it, frame table All that exists inside entry The linked list To manage a separate (struct list frame_list), This is discussed below frame eviction (clock) algorithm It is necessary for the. Frame table The calculation for all the pages are added and deleted and implemented to always be able to ensure the consistency of data (straightforward). Also frame table Operation to access the concurrency Since the considerations, frame table All operations are lock Hold critical section Was to be carried out in (frame_lock).

(2) (Supplemental) Page Table

Page table It is is conceptually user virtual address The physical address It serves to map to. user program in virtual address When the access through, corresponding to frame Of the same effect as accessing the memory area. PintOS The page table Because of the format, (user virtual) page The data structure that stores all the additional information related to this right requires separately Supplemental Page Table (Below SUPT) to be.

SUPT The Threads (process) Each one is created, uaddr (user page) 7→SPTE (supplemental page table entry) It is to be understood as the mapping. SPTE It stores the following information:

```
enum page_status {
    ALL_ZERO,                  // All zeros
    ON_FRAME,                  // Actively in memory
    ON_SWAP,                   // Swapped (on swap slot)
    FROM_FILESYS               // from filesystem (or executable)
};

struct supplemental_page_table_entry
  {
    void * Upage;                      / * Virtual address of the page (the key) * /
    void * Kpage;                      / * Kernel page (frame) associated to it.
                                          Only effective when status == ON_FRAME. If the page is not on the frame, should be
                                          NULL. * /
    struct hash_elem elem;

    enum page_status status;

    bool dirty;                        / * Dirty bit. * /

    // for ON_SWAP
    swap_index_t swap_index; / * Stores the swap index if the page is swapped out.
                                          Only effective when status == ON_SWAP * /

    // for FROM_FILESYS
    struct file * file;
    off_t file_offset;
    uint32_t read_bytes, zero_bytes;
    bool writable; };
```

- upage: It is a key value, user virtual address Stores.
- status: Applicable page A represents a state.
  - ON_FRAME: The case is loaded into the frame. At this time load The frame The address is kpage It must be stored in and frame table The kpage Corresponding to the key, frame table entry To be found.
  - ON_SWAP: The current page frame in evict Became swap disk When present in a. At this time, swap disk Of whether the stored somewhere swap_index Stored in, and with this value subsequently disk Read the appropriate parts of the swap in can do.
  - ALL_ZERO: Page While not loaded in the frame, all the details 0 By means populated state.
  - FROM_FILESYS: Page It is also frame While, the mihanda that if the contents are to be loaded from the file system (executable or memory-mapped Etc). Some of the files where o ff set From what is file, file_offset, read_bytes, zero_bytes It can be seen as such. writable this false If there read-only page to be.
- dirty: page of dirty bit. swapped out Of the page dirty bit It is pagedir_is_dirty () Because not come out through dirty Whether it is necessary to store separately. The details of this in Where Used memory mapped fi les It will be described in.

SUPT Down for the use of Page Fault It will be described in detail in the processing algorithm.

(3) Swap Table

Swap And it provides for the operation.

- swap_index_t vm_swap_out (void * page): Swap-Out To be carried out. page Of the contents swap disk Written to, and capable of identifying the position swap_index The returns.
- void vm_swap_in (swap_index_t swap_index, void * page): Swap-In To be carried out. swap_index In a position single page To read, page It is written in.
- void vm_swap_free (swap_index_t swap_index): Applicable swap region Just throw away.

Swap disk Of the entire sector PGSIZE Divided by the number of possible swap slot Is the number of. Page size end swap sector of size Since than the size of one of the page The swap In order to save PGSIZE / BLOCK_SECTOR_SIZE The consecutive block This is necessary. what swap slot this available Korean paper bitmap And it managed using a data structure, corresponding block and page To map the swap in / out To implement a very straightforward Since a detailed description thereof will be omitted. (see commit b20fe2c9)

1.2. Swapping and Evicting

Frame allocation city, page allocation If that failed, if there is insufficient memory when yimeuroyi swapping A must. That is, some (frame) page The swap disk in swap out And to allocate a new page in the empty space. At this time, the following happens:

- Evict which frame And even the. (second-chance clock algorithm)
- Applicable frame of page mapping To be released. In other words, the owner thread pagedir in, upage Remove the mapping.
- frame Of the contents swap out do. (readonly When one fi lesystem to)
- Applicable frame of frame table It is removed from (page Degree free do)
- After that page To reassign the newly assigned frame Put on table manages the mapping.

Eviction In order to select a frame, second-chance algorithm It was used.

- victim pointer (clock_ptr) To maintain, frame table The elemental circular It traverses.
- reference bit end One If, reference bit The 0 Set and then entry It goes to.
- reference bit end 0 If, eviction Select a destination.

reference bit In order to verify / setup of a simple mapping to the frame upage The pagedir How to determine which was used in. Reference bit To understand the pagedir_is_accessed () A, reference bit The 0 To set
pagedir_set_accessed () You can use the (user program in page access As when internally reference bit end One It is set to). previously frame entry The linked list Since the management can be efficiently implemented in the above algorithm.

```
/ ** Frame Eviction Strategy: The Clock Algorithm * /
struct frame_table_entry * clock_frame_next ( void );
struct frame_table_entry * pick_frame_to_evict ( uint32_t * Pagedir) {

    size_t n = hash_size (& frame_map);
    if (N == 0 ) PANIC ( "Frame table is empty, can not happen - there is a leak somewhere" );

    size_t it;
    for (It = 0 .; it <= n + n; ++ it) // prevent infinite loop. 2n iterations is enough
    {
        struct frame_table_entry * e = clock_frame_next ();
        if (E-> pinned) continue .;
        else if (Pagedir_is_accessed (pagedir, e-> upage)) {
            // if referenced, give a second chance.
            pagedir_set_accessed (pagedir, e-> upage, false );
            continue .; }


        // OK, here is the victim: unreferenced since its last chance
        return e; } PANIC ( "Can not evict any frame -! Not enough memory \ n" );


}
```

1.3. Page loading: Page Fault Handler

If you have access to the page does not load, that is, pagedir That does not exist memory When you approach the area page fault It is generated. At this time, SUPT on page Iteotdamyeon contains not a bad memory area access swapped out It is either lazy-load Due to the frame Because it is not loaded page Again load This action is necessary to. This operation vm_load_page () There is implemented the function.

1. SUPT entry Confirms. If it does not, because it is invalid memory access fault after kill do.
2. Store the contents of the page frame page Obtain the. This time frame is different evict It may be.
3. SUPT Stored in the status The report, loads the appropriate data in the memory.
   - ALL_ZERO: just 0 Fills (memset).
   - ON_FRAME: NO-OP. (Can not happen?) One
   - ON_SWAP: swap in A must. vm_swap_in () Invoke loads the data.
   - FROM_FILESYS: SPTE Stored in the fi le With information such as a pointer reads the data from within the file system. O ff set file_offset From read_bytes It has been read by the remaining area 0 To be filled.
4. upage Wow, 2. Generated by the new frame Maps the (pagedir_set_page ()). Now frame Status of the back ON_FRAME This is.

Data by filling in with the handler page By loading, Lazy load , Which is denoted by page (eg FROM_FILESYS) The resolving It is naturally to perform.

1.4. Process Loading When processing (Lazy-Load)

Over paging scheme Since the implementation, PintOS end process When load all segment In the memory load Instead, you need to segment Only lazy load It is made possible. load_segment () (see process.c: 639) In, the past, assignment page Filling the file from install_page It was right on the other hand, vm_supt_install_filesys () Using lazy load A pointer to the file so that it can be (this file executable Since before the process is terminated is that it is still valid pointer) and o ff set, size Information SUPT It allows the recording.

However, initial user stack When the setting (setup_stack ()), lazy loading The not used. Of course, the first stack Assigned to the domain page And frame this SUPT And frame table The proper treatment needed to be properly loaded.

One Oh, of course, because it is already a new load frame Not just assign the return.

1.5. Process Termination When processing

When the process is completed, a process that was occupied pagedir end destroy In the process of all that has been allocated page It is set to On. In addition SUPT Tables require having also remove. SUPT Table per-thread scope Because just when there is a deletion, if the process were assigned frame or swap slot If you have these in turn would have a proper release. If'll make a turn off (the thread free Search), later evict which frame When the bone is already rateul pagedir Since such information to the destroyed state required can cause problems.

This part of the concept is simple: the implementation ropda somewhat tricky. For more information: commit eb9d8be0 of di ff And message Let's see. Only the most important points of straw to sleep, before we SPTE The frame mapping of the kpage Save it because it was haedu key By a frame table You can delete them from the invalid frame anymore. if page Of state SWAPPED Yeotdamyeon, occupied swap If it is a year off. Add to memory mapped fi le And this, if performed released for it (3. Description) from.

SUPT end destroy Since the process, hash table All element Performed by destructor Which uses a callback response within two frame entry Remove the (vm_frame_remove_entry ()) It has been treated to. When a thread (process) is terminated resources are released, pagedir All associated page It is also free At this time there is double-free It has to be careful not to happen.

1.6. Frame Pinning

User memory When approached, the kernel virtual memory paging Handle, and there still another page fault It may have a problem occurs Wig[2]. For example, write system call To perform user memory To read fi le system on write While page fault When occurs (user page end swap State) , swap in When fi lesystem The other access It is needed. By the way PintOS The
fi lesystem silver lock So you can get a hold once, double-lock A kernel panic occurs by.

To solve this problem, frame pinning You should use a method called. Pinned A frame eviction To be excluded from. Nine to the current method, frame table entry on pinned in boolean Leave the field pinned Ranked or not, vm_frame_unpin (), vm_frame_pin () Of a particular frame as a function of such pinned It was implemented to set up a bit.

- first frame When assigning eviction This does not happen to pin And, frame When the load is complete (vm_load_page () Reference) pin To be released.
- read () And write () of system call Ahead before performing the target bu ff er That all of the user page The pin do. This action is vm_pin_page () It was implemented, page The SUPT in lookup If this is the frame to load its frame of pin do. SUPT There may not be on, lazy-load Because we are using it may be a stack region that have not yet been approached (stack growth) Neglected.
- system call After performing is completed, the above pin Of all the frames were pin To be released. vm_unpin_page () There are operations that are implemented in.

[2] 'Page-merge-mm', 'page-parallel' Such as the test case.

2. Stack Growth

Previously, user program Stacks of One page When consisted only exceed them page fault It has occurred. page Within the acceptable limits for stack this grow The goal is to be able to handle.

first, user program Exceeding the stack page fault It is generated. therefore page fault When occurred, stack access If so, to determine whether there page And a new assignment. sure, stack access It is not a bad memory access It may need to have the appropriate method.

2.1. Stack Access Determine how

First, the stack pointer esp To retrieve. Page fault Address occurred fault_addr When called to time, the two are both satisfied under, to allocate a new page is to be increased in the stack.

- User memory Whether the stack area: PHYS_BASE - MAX_STACK_SIZE <= fault_addr && fault_addr <PHYS_BASE (MAX_STACK_SIZE It is 8MB It was defined as a).
- Whether on the stack frame: esp <= fault_addr or fault_addr == f-> esp - 4 or fault_addr == f-> esp - 32. This is according to the manual, 80x86 PUSH / PUSHA operation This stack pointer 4 Bottom bytes (PUSH) or 32 Bottom bytes (PUSHA) Because use.

By the way, esp That gets to the user mode Wow kernel mode Some depend on. if user mode Ramen intr_frame Immediately it can obtain from, but (f-> esp), read () or write () While performing system calls, etc. etc. stack In the area page fault That may occur. previously Project 2 In, wrong user memory Whether the approach page fault Based method to identify hayeoteumeuro (Accessing User Memory Section), this time, f-> esp The user program stack pointer It does not exist.

to solve this problem, system call At the time of the call to the user's esp The thread Stores it in the structure. then, page fault To occur interrupt handler Based on this example, even with a ohdeora esp The can find out. if (1) user mode if f-> esp The same (2) kernel mode If you have stored in the thread curr-> current_sep If you are used to.

```
@@ -118,4 +118,8 @@ struct thread
          struct list file_descriptors;                    / * List of file_descriptors the thread contains * /

          struct file * executing_file;                    / * The executable file of associated process. * /
+
+         uint8_t * current_esp;                           / * The current value of the user program ' s stack pointer.
+                                                               A page fault might occur in the kernel, so we might
+                                                               need to store esp on transition to kernel mode. (4.3.3) * /
  # Endif
```

```
    / * (4.3.3) Obtain the current value of the user program's stack pointer.
      *   If the page fault is from user mode, we can obtain from intr_frame `f`,
      *   but we can not from kernel mode. We've stored the current esp
      *   at the beginning of system call into the thread for this case. * /
    void * Esp = user? f-> esp: curr-> current_esp;
```

2.2. Stack The increase Paging process

If the foregoing determination conditions are satisfied, fault_page (fault_addr The PGSIZE in align One page Find out the starting address) is assigned to the page here.

therefore supplemental page table on fault_page The user address Loja the (new) entry It adds. Added entry

Uita mouth ALL_ZERO To all data end zero- fi lled It is a method that was used (see vm_supt_install_zeropage ()).

Linux System Uninitialized stack is 0xcc But it contains the value of the waste, such as convenience here, zero It was used.

now vm_load_page () Through the frame If the load on the page memory after the assignment, stack growth Processing of the interrupt handler for the ends.

```
// Stack Growth
bool on_stack_frame, is_stack_addr;
on_stack_frame = (esp <= fault_addr || fault_addr == f-> esp - 4 || fault_addr == f-> esp - 32 ); is_stack_addr = (PHYS_BASE - MAX_STACK_SIZE <= fault_addr &&
fault_addr <PHYS_BASE);
if (On_stack_frame && is_stack_addr) {
  // OK. Do not die, and grow.
  // we need to add new page entry in the SUPT, if there was no page entry in the SUPT. // A promising choice is assign a new zero-page.

  if (Vm_supt_has_entry (curr-> supt, fault_page) == false ) Vm_supt_install_zeropage (curr-> supt,
  fault_page); }


if (! Vm_load_page (curr-> supt, curr-> pagedir, fault_page))
        goto PAGE_FAULT_VIOLATED_ACCESS;
```

Detailed implementation commit 0703878 Let's see.

3. Memory Mapped Files

Memory mapped fi les In order to function mapid_t mmap (fd, addr) And munmap (mapid_t) This call should be implemented. Their implementation is very straightforward Do.

first, struct thread In that process, memory-mapped The descriptor Stores of information.

```
@@ -127,6 +127,9 @@ struct thread
 # ifdef VM
     // Project 3: Supplemental page table. struct
     supplemental_page_table * supt;                    / * Supplemental Page Table. * /
+
+    // Project 3: Memory Mapped Files.
+    struct list mmap_list;                          / * List of struct mmap_desc. * /
 # Endif

 / * Owned by thread.c. * /
```

memory-map descriptor You must store the following information: Project 2 of fi le descriptor And almost decals. File and its size, the mapping page Remember the address.

```
struct mmap_desc {
   mmapid_t id;
   struct list_elem elem;
   struct file * file;

   void * Addr;          // where it is mapped to? store the user virtual address
   size_t size; // file size
};
```

3.1. mmap ()

mmap () system call The implementation is as follows: Also straightforward Do.

1. argument The checks. Manuals speci fi cation Depending on the, page address end 0 If, align If not, fd end 0
   or One If Ann Wu, an invalid memory area, if the existing different memory areas of overlap (SUPT To examine whether a page exists) Etc. are applicable.
2. Open the file and check the size. file_reopen () Use the fi le descriptor Managed by fi le And pointers separately
   You can manage your files. Held here fi le Of course, munmap () In closed.
3. vm_supt_install_filesys () Using, it sets the page (FROM_FILESYS). o ff set Some areas in accordance with the
   Read how mmap_desc , And store them on the structure. At this time in front of the pages except the last page are PGSIZE
   Byte is mapped as the last remaining pages bytes Must maps. So now that page It is fi le system
   The data from the lazy load It is in a state as possible.

3.2. munmap ()

munmap () system call The implementation is as follows: Also straightforward Do.

1. One. given mapping id The corresponding mmap_desc The look (without fail). All of which are assigned to it page The While touring, vm_supt_mm_unmap () Calls each page Each performs a suitable release (described below).
2. vm_supt_mm_unmap () The mapping Each of the page Is, they should be accompanied tasks are performed while off.
   - pagedir in user address It releases the mapping between the (later access All page fault).
   - page of status Depending on the - frame table Or removed from (ON_FRAME), Assigned swap slot Discard (ON_SWAP).
   - Besides, page end dirty state in case of fi le on page I will write the contents is necessary. dirty state If so, the corresponding fi le The area frame page The contents or swap It allows the contents of the record (temporary page on swap-in O 'clock, write it back to the file).

Page not really dirty Whether it is tracking dirty state In order to find out whether pagedir_is_dirty () To use.
dirty Criteria for determining whether the (1) SPTE on dirty Or by marking[3] (2) user page end dirty Or (3) kernel page (frame) end dirty Or (pagedir_is_dirty () use) One of the three is satisfied when (user page It is frame page A kind of aliasing Because it must be so).

When the process is finished (process_exit ()), As I've closed all open files memory-mapped fi le This is also necessary if the parts to turn them all. When the process is finished, munmap () system call By the above procedure call handler (in particular,
write back on the fi le) This was all handled makes it happen.

Memory-mapped fi les Implementation of the detail silver commit 754ce211 And a6ebf11c of di ff And message Make a note.

[3] To do this, of course, frame this evict And swap out When, dirty Whether yeotneunji SPPE It is written to. If you do not write later write back Whether or not to be pagedir Because the information can not be seen alone. later frame load When the SPTE of dirty fl ag It is reset.

4. Test Results

Project 3 Requirements of the Paging, Stack Growth, mmap We implemented a system call or the like, 109 One passed all test cases. These test cases Project 2 Who finished in user program, system call In addition to basic functions, stack growth / paging / mmap Whether the default behavior and eseoui exception of processing and complex situations where multiple processes are running at the same time (parallel merge sort, Via file communication) Such as robustness It verifies.

```
TOTAL TESTING SCORE: 100.0% ALL TESTED PASSED
- PERFECT SCORE

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

SUMMARY BY TEST SET

Test Set                                              Pts Max% Ttl% Max
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  - - -  - - -  - - - - - -  - - - - - -
tests / vm / Rubric.functionality                       55/55 50.0% / 50.0%
tests / vm / Rubric.robustness                          28/28 15.0% / 15.0%
tests / userprog / Rubric.functionality               108/108 10.0% / 10.0%
tests / userprog / Rubric.robustness                    88/88 5.0% / 5.0%
tests / filesys / base / Rubric                         30/30 20.0% / 20.0%
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  - - -  - - -  - - - - - -  - - - - - -
Total                                                           100.0 / 100.0%
```

Conclusion

The difficulty of implementing was equivalent hand. Paging / Frame The related operations by rather complex, the kernel-level API It is also important to clean the design, race condition This is also necessary to implement correctly without taking into account the various bugs, if not to occur. Fortunately, many such that multiple processes are running at the same time, concurrency In the circumstances, appropriate sync with lock Through such process

virtual memory That could be implemented to operate properly. Most of what to something if defined, only those details and the approximate direction of its implementation straightforward Together, documented in a work process (tin, and commit message) Hayeoteumeuro well detail

What you are comment Wow commit message I hope to see.

Some are also areas that need some refactoring rather it seems a little more robust And it remains also regret that you have to start from scratch a beautiful design and implementation. Virtual memory scheme And to properly manage them kernel Was able to understand the operation of the process, the design / implementation / debugging such various difficulties but was also fun and informative time were many.