SWE3004: Operating Systems
prof. Euiseong Seo

Project 5. Page Fault Handler

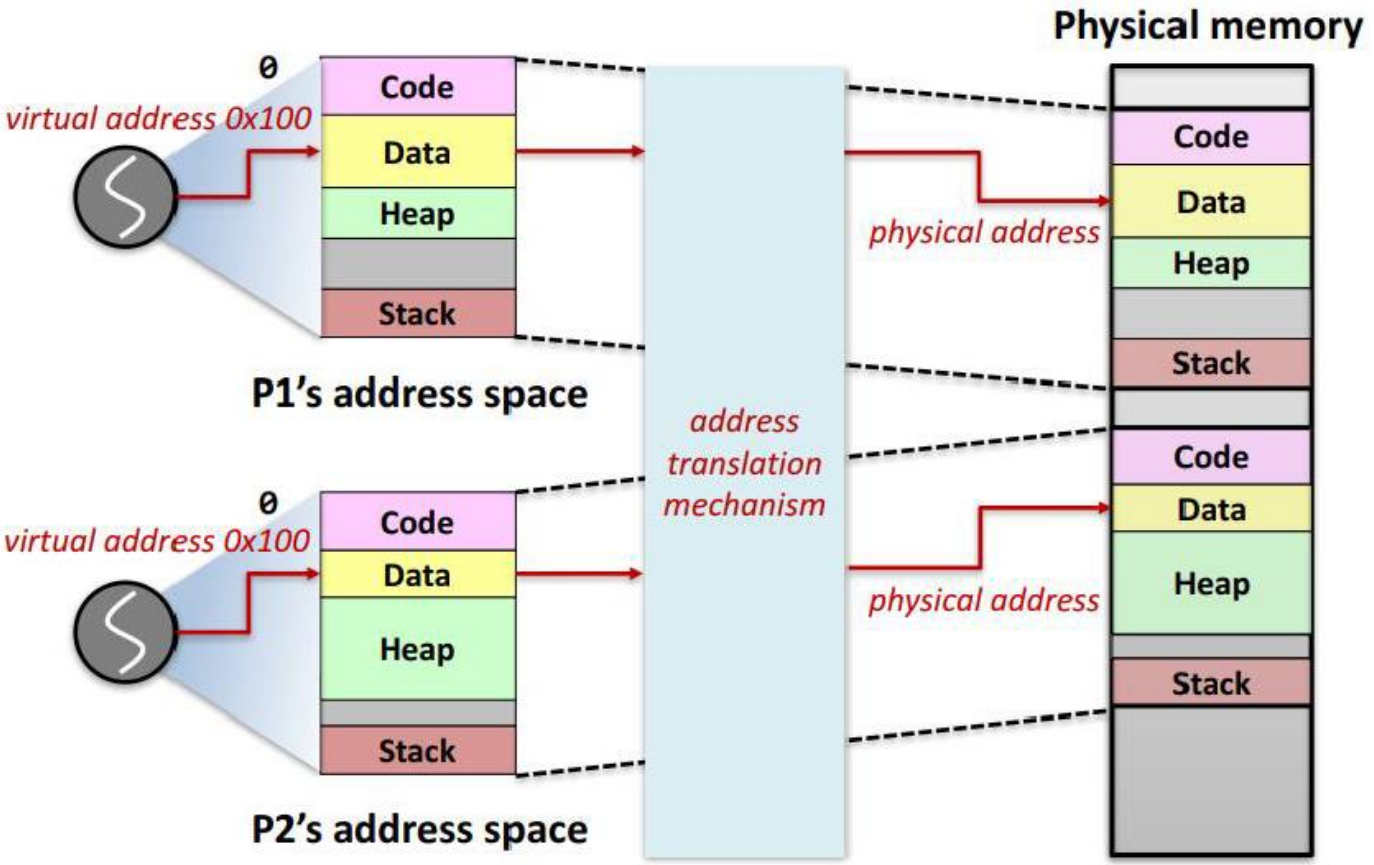
2019.5.15 (Wed.)

TAs
김종석(ks77sj@gmail.com) /
최동규(gmj03003@gmail.com)

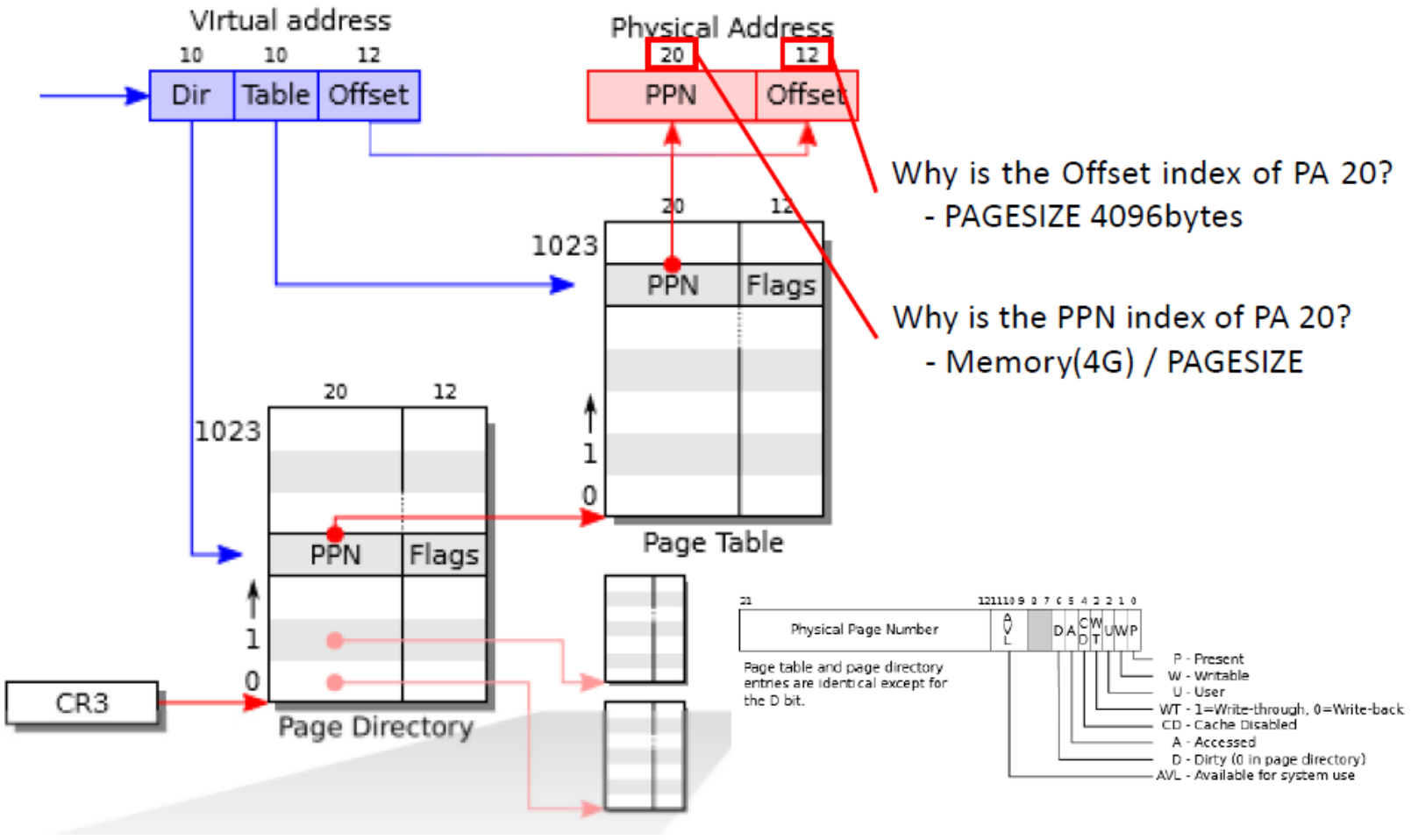
Project Plan

- Total 7 projects
  0) Starting xv6 operating system (5%)
  1) System call (10%)
  2) Thread (20%)
  3) Syncronization (15%)
  4) Scheduling 1(15%)
  5) Page fault handler (15%)
  6) Copy on Write (20%)

What is VM(Virtual memory)?



Address Translation in Intel x86



mmu.h

```
103 // A virtual address 'la' has a three-part structure as follows:
104 //
105 // +--------10------+-------10-------+---------12----------+
106 // | Page Directory |   Page Table    | Offset within Page  |
107 // |      Index     |      Index      |                     |
108 // +----------------+-----------------+---------------------+
109 //  \--- PDX(va) --/ \--- PTX(va) --/
110
111 // page directory index
112 #define PDX(va)         (((uint)(va) >> PDXSHIFT) & 0x3FF)
113
114 // page table index
115 #define PTX(va)         (((uint)(va) >> PTXSHIFT) & 0x3FF)
116
117 // construct virtual address from indexes and offset
118 #define PGADDR(d, t, o) ((uint)((d) << PDXSHIFT | (t) << PTXSHIFT | (o)))
119
120 // Page directory and page table constants.
121 #define NPDENTRIES      1024    // # directory entries per page directory
122 #define NPTENTRIES      1024    // # PTEs per page table
123 #define PGSIZE          4096    // bytes mapped by a page
124
125 #define PGSHIFT         12      // log2(PGSIZE)
126 #define PTXSHIFT        12      // offset of PTX in a linear address
127 #define PDXSHIFT        22      // offset of PDX in a linear address
128
129 #define PGROUNDUP(sz)  (((sz)+PGSIZE-1) & ~(PGSIZE-1))
130 #define PGROUNDDOWN(a) (((a)) & ~(PGSIZE-1))
131
132 // Page table/directory entry flags.
133 #define PTE_P           0x001   // Present
134 #define PTE_W           0x002   // Writeable
135 #define PTE_U           0x004   // User
136 #define PTE_PWT         0x008   // Write-Through
137 #define PTE_PCD         0x010   // Cache-Disable
138 #define PTE_A           0x020   // Accessed
139 #define PTE_D           0x040   // Dirty
140 #define PTE_PS          0x080   // Page Size
141 #define PTE_MBZ         0x180   // Bits must be zero
142
143 // Address in page table or page directory entry
144 #define PTE_ADDR(pte)   ((uint)(pte) & ~0xFFF)
145 #define PTE_FLAGS(pte)  ((uint)(pte) &  0xFFF)
146
```

Page Fault Exception in Intel x86

- Conditions
  - There is no translation for the linear address
  - There is a translation for the linear address, but its access rights do not permit the access

- CR2 stores the linear address that caused a page fault

- Processor triggers interrupt 14 (page fault)

Page fault handler in xv6

traps.h

```
// x86 trap and interrupt constants.

// Processor-defined:
#define T_DIVIDE         0      // divide error
#define T_DEBUG          1      // debug exception
#define T_NMI            2      // non-maskable interrupt
#define T_BRKPT          3      // breakpoint
#define T_OFLOW          4      // overflow
#define T_BOUND          5      // bounds check
#define T_ILLOP          6      // illegal opcode
#define T_DEVICE         7      // device not available
#define T_DBLFLT         8      // double fault
// #define T_COPROC      9      // reserved (not used since 486)
#define T_TSS           10      // invalid task switch segment
#define T_SEGNP         11      // segment not present
#define T_STACK         12      // stack exception
#define T_GPFLT         13      // general protection fault
#define T_PGFLT         14      // page fault
// #define T_RES        15      // reserved
#define T_FPERR         16      // floating point error
#define T_ALIGN         17      // aligment check
#define T_MCHK          18      // machine check
#define T_SIMDERR       19      // SIMD floating point error

// These are arbitrarily chosen, but with care not to overlap
// processor defined exceptions or interrupt vectors.
#define T_SYSCALL       64      // system call
#define T_DEFAULT      500      // catchall

#define T_IRQ0          32      // IRQ 0 corresponds to int T_IRQ

#define IRQ_TIMER        0
#define IRQ_KBD          1
#define IRQ_COM1         4
#define IRQ_IDE         14
#define IRQ_ERROR       19
#define IRQ_SPURIOUS    31
```

- If page fault occurs, “trapno” of trapframe automatically filled with T_PGFLT and call trap function in trap.c
- You have to make your “own” page fault handler
- Currently, implemented as below…
  - rcr2() -> page fault address

trap.c

```
//PAGEBREAK: 13
default:
  if(myproc() == 0 || (tf->cs&3) == 0){
    // In kernel, it must be our mistake.
    cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",
            tf->trapno, cpuid(), tf->eip, rcr2());
    panic("trap");
  }
  // In user space, assume process misbehaved.
  cprintf("pid %d %s: trap %d err %d on cpu %d "
          "eip 0x%x addr 0x%x--kill proc\n",
          myproc()->pid, myproc()->name, tf->trapno,
          tf->err, cpuid(), tf->eip, rcr2());
  myproc()->killed = 1;
}
```

User Address Space in xv6

- 1 stack page & 1 guard page

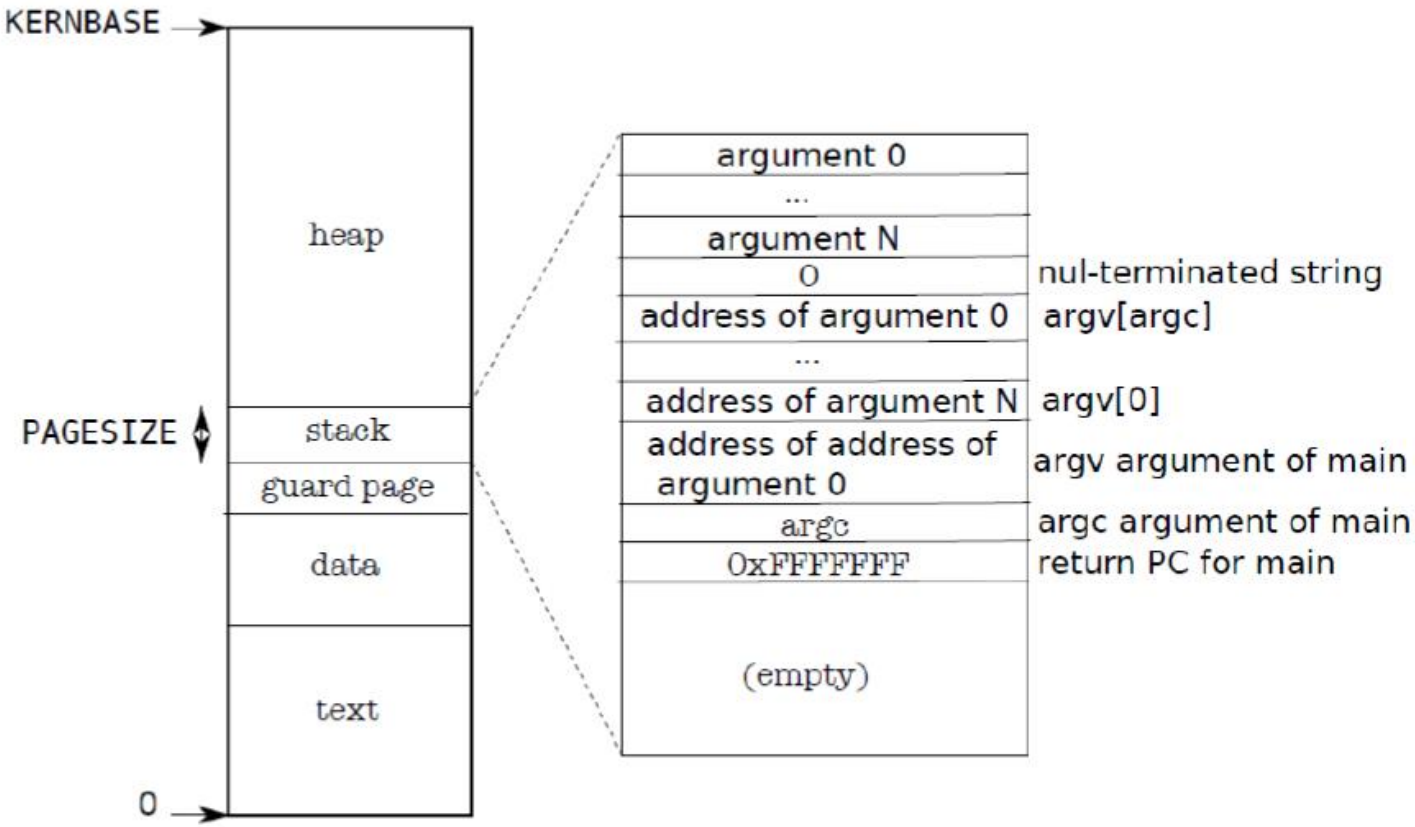


Figure 2-3. Memory layout of a user process with its initial stack.

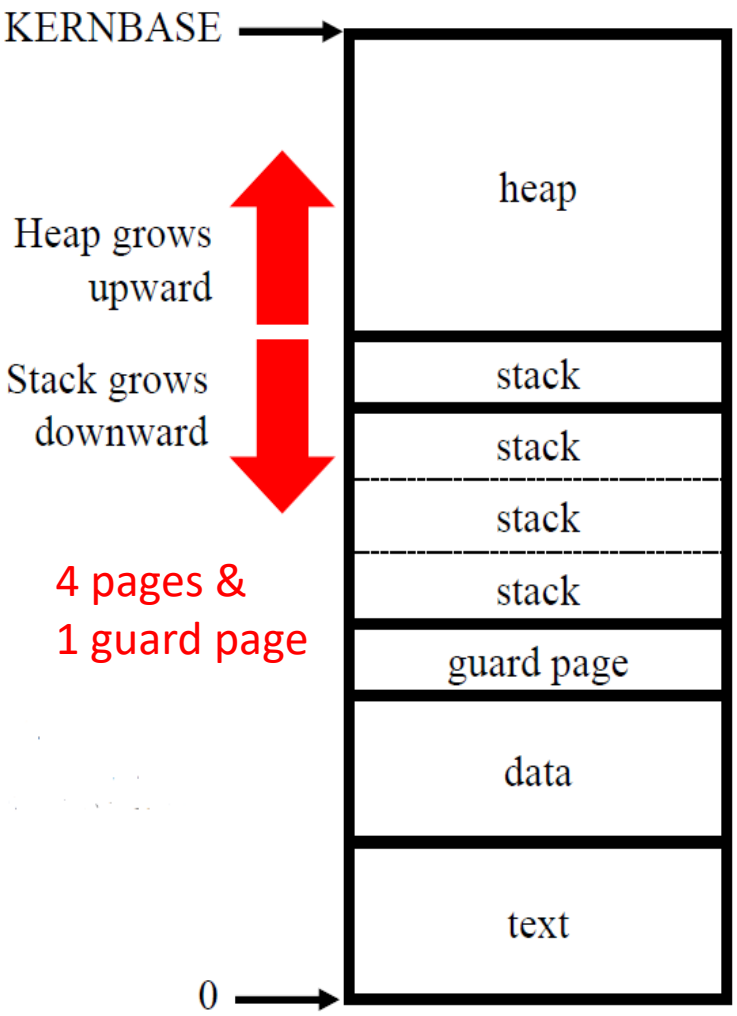
Project 5. Stack growth

- Initial size of stack
  - Prepare 1 page initially
  - Can be grow up to 4 pages
- Growth of stack
  - tf->esp can move upto 32bytes
  - New page should be allocated

  to this process if current stack is full
- When stack pointer reaches guard page

  or a process accesses invalid address,

  kill that process



Project 5. Stack growth

- Print out “Invalid access” when approaching abnormal address
- Print out “Allocate page” if page-fault handler is executed normally

Template Code

- wget http://csl.skku.edu/uploads/SWE3004S19/xv6-skku.tar.gz
- Modifications
  - halt system call
  - Halt xv6 program
  - make tarball
    - Compress your source codes into one .tar.gz file for submission
    - You should enter your ID & project no. on Makefile
  - CPUS=1

Test Case

- Erase –Werror in Makefile

```
CC = $(TOOLPREFIX)gcc
AS = $(TOOLPREFIX)gas
LD = $(TOOLPREFIX)ld
OBJCOPY = $(TOOLPREFIX)objcopy
OBJDUMP = $(TOOLPREFIX)objdump
CFLAGS = -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer
#CFLAGS = -fno-pic -static -fno-builtin -fno-strict-aliasing -fvar-tracking -fvar-tracking-assignments -O0 -g -Wall -MD -gdwarf-2 -m32 -Werror -fno-omit-frame-pointer
CFLAGS += $(shell $(CC) -fno-stack-protector -E -x c /dev/null >/dev/null 2>&1 && echo -fno-stack-protector)
ASFLAGS = -m32 -gdwarf-2 -Wa,-divide
# FreeBSD ld wants ``elf_i386_fbsd''
LDFLAGS += -m $(shell $(LD) -V | grep elf_i386 2>/dev/null | head -n 1)
```

- Correct Result

```
================================ Result=================================
Allocate page
Allocate page
Allocate page
Invalid access
========================================================================
```

Submission

- You need to submit a document.
- Just write how you implemented your code.
- You can use English or Korean.

Submission

- Send your code file (xv6-project-5-studentID.tar.gz) and document file to ks77sj@gmail.com
- Please send a mail with tittle including [SWE3004-P5]
  - Ex) [SWE3004-P5] 2014111111-project5
- PLEASE DO NOT COPY
  - YOU WILL GET F GRADE IF YOU COPIED
- Due date: 5/22(Wed.), 23:59:59 PM
  - Delays are allowed only one week from the deadline. And there will be up to -40% penalty.

Questions

- If you have questions, please email to TA
  - You can't ask questions on deadline day
- You can also visit #85533. Please email TA before visiting