Project #4: Per-Process Paging

Sejun Kwon
(sejun000@snu.ac.kr)
Systems Software &
Architecture Lab.
Seoul National University

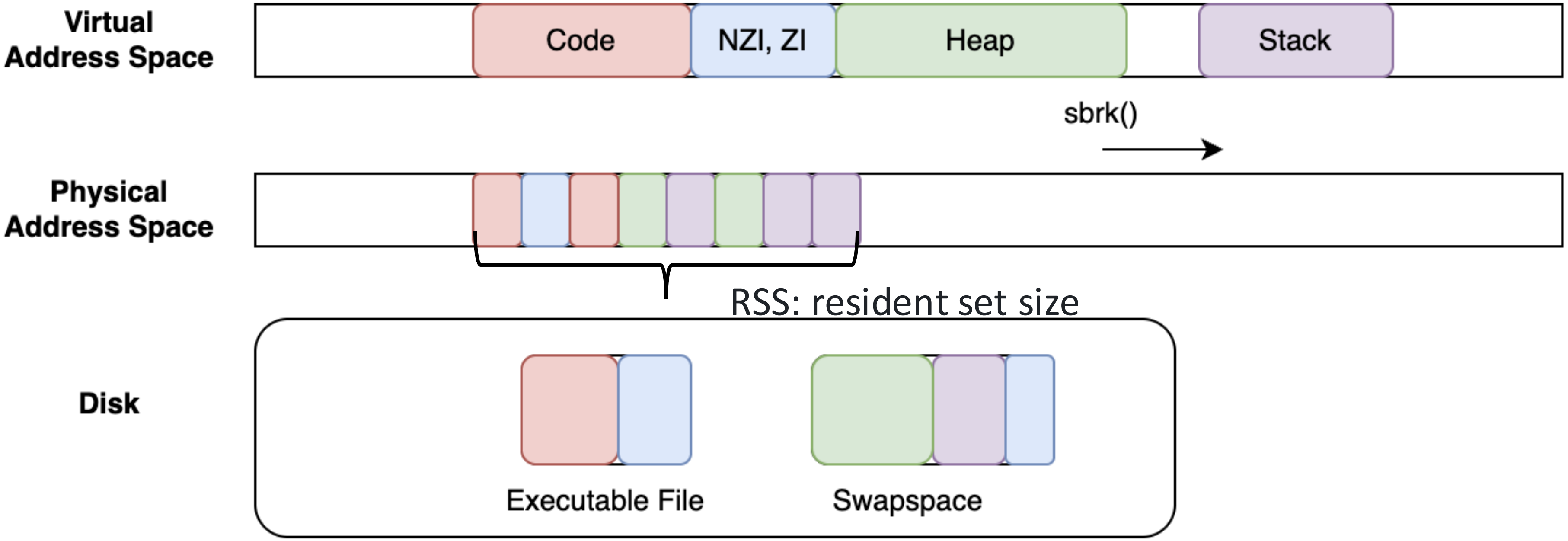
2025.11.13.

Paging

- **Paging** lets each **process** divide its virtual memory into fixed-size **pages**, which map to physical memory **frames**
- Each page can be stored in **any frame**, reducing memory fragmentation
- A **page table** keeps track of where each page is stored, helping the CPU translate virtual addresses to physical ones

Demand Paging (Linux)

- **Demand paging**
  - A page frame is allocated (and loaded from disk if needed) **only when the process actually accesses that page**
- **Swap out:**
  - **evicting a page frame** from memory, typically dirty page

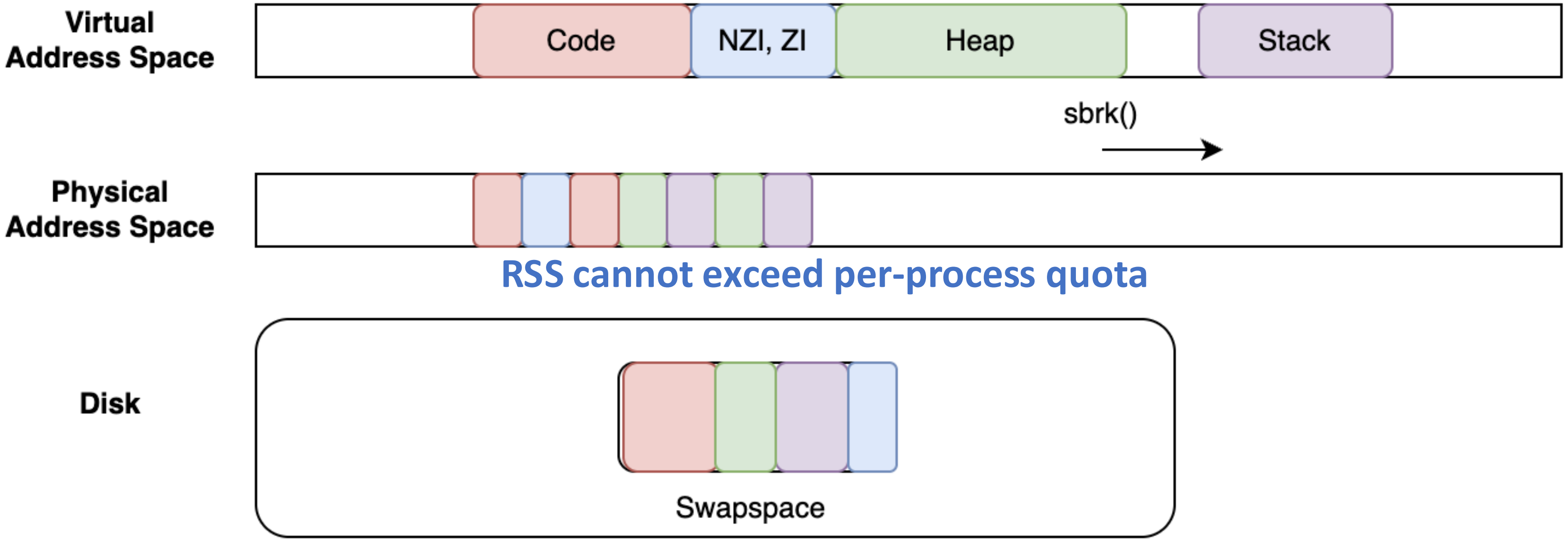


XV6 Baseline

- page faults occur **only for lazily allocated heap pages created via sbrk(n, SBRK_LAZY)**
- all other regions (code, data, stack, and heap with SBRK_EAGER) are already kept resident in physical memory after exec or fork

Per-Process Paging

- **Swap out physical pages that exceed the quota**
- **Swap out any user process page frame, regardless of whether it belongs to code, heap, or stack**
- **Do not swap out memory used by the kernel**



RSS cannot exceed per-process quota

Part 1. System Calls

- rss_set(): Sets the size of the quota for the current process
- rss_get(): Reads the size of the quota for the current process
- rss_stat(): Reads the RSS-related statistics for the current process, including: quota, rss size, nrss, faults, swap-ins, swap-outs
- **Test whether the RSS value is correct after** fork(), exec(), sbrk(SBRK_EAGER), **and** sbrk(SBRK_LAZY)
- **You don’t need to consider swapping**

Part 2. Per-Process Paging with FIFO (1)

- **Eligible for eviction**
  - Only *user-space pages* within the process's virtual address range
  - including the *user stack guard page*
- **Not eligible:**
  - Pages **outside** this range (e.g., the *trampoline* and *trapframe* pages).
  - Pages with **kernel-only mappings** (e.g., the *kernel stack* and its *guard page*).

Part 2. Per-Process Paging with FIFO (2)

- A second **virtio-blk disk (fixed 32 MiB)** is attached to **QEMU** as the **swap device**.
- The **mkswap utility** (mkfs/mkswap.c) formats this device, writing a **swap magic number** and the **number of available swap slots** to its superblock
- During boot, **swapinit(SWAPDEV)** is called to **validate the swap device**
- For data movement, the kernel directly accesses **swap slot s** using **swapread(pa, s)** or **swapwrite(pa, s)**, where **pa** is the physical address of a **4 KiB buffer** (one page)

Part 2. Per-Process Paging with FIFO (3)

- Each process maintains its own **FIFO queue** of resident user page frames
- When a process's **rss exceeds its quota**:
  - The kernel selects a victim page from the process's FIFO queue
  - It allocates a **free swap slot**
  - It writes the page's contents to that slot using swapwrite()
  - It updates the page's **PTE to mark it as non-resident**
- The **oldest eligible resident page** in the queue is selected as the **victim** and swapped out
- Newly swapped-in pages are **appended to the end** of the FIFO queue

Part 2. Per-Process Paging with FIFO (4)

- Part 2 targets a single-hart machine (CPUS := 1 in Makefile)
- usertests -q , vmtest
- Both the **parent and child processes** can have their **quotas set smaller than their actual memory usage**
- The values of **rss, nrss, page faults, swap-ins, and swap-outs** must all be **correct**
  - Only **page faults triggered by traps (usertrap)** shall be counted

Part 3. Multi-hart Support without Memory/Space Leaks

- The implementation must be **correct and free of deadlocks** when running on **multi-hart systems**
- **No Memory Leaks:**
  - Ensure **all page frames and kernel allocations** are properly **freed** on both normal execution paths and aborted termination paths
- Avoid **double frees** in all cases
- **No Swap-Space Leaks:**
  - Every allocated **swap slot** must be **immediately released** after a successful **swap-in**, and also upon **normal or aborted process termination**.

BONUS: Your Own Replacement Policy (1)

- **Bonus Condition:**
  - Designing and integrating a custom **page replacement policy** that reduces **page faults** for workloads with **memory-access locality**
- **Correctness and Stability:**
  - **correct**, **deadlock-free**, and **leak-free** under the same testing conditions as Part 1, 2, 3
- **Allowed Extensions:**
  - You may track **lightweight per-page metadata** or maintain **auxiliary state**, as long as the overhead is **modest** and the design is **clearly explained** in your design document

BONUS: Your Own Replacement Policy (2)

- **Evaluation Criteria:**
  - Submissions will be ranked by the **total number of page faults** across multiple workloads (**lower is better**).
  - The **top 10 submissions** will receive a **20% bonus**, and the **next 10 submissions** will receive a **10% bonus**.
- **There will be a non-tight timeout**

BONUS: Your Own Replacement Policy (3)

- Testcases
  - When **the locality size is similar to the quota**
  - When **the locality size slightly exceeds the quota**
  - When **the locality changes over time**

Restriction

- Do not change the allocation order, access order, or modification order **of user pages** inside every syscall in skeleton code
- When handling a system call, if there are user pages that were swapped out, the kernel **must not incrementally** enforce the quota **during the syscall.** Instead, it must first read in all such pages, and enforce the quota in **FIFO order(part 1, 2, 3)** only after all required **user page** accesses have finished (right before returning from the syscall)
- In the rss_stat() syscall, the swapins and swapouts values returned to the user buffer must not include any swap-in or swap-out operations that occur *inside the execution of the rss_stat() syscall itself* (i.e., swapping caused by accessing the user buffer passed to rss_stat())
- Only **page faults triggered by traps (usertrap)** shall be counted as "rss_stat. faults"

Restriction

- **Code pages** must also be **eligible for swapping**
- After a **swap-in**, the corresponding **swap block on the swap device must be freed immediately**
  - **You must not keep the swap slot while throwing away a clean swapped-in page**
  - **Zero-mapped heaps are not allowed**
- Swap slot is owned by exactly one PTE and must never be referenced by more than one PTE
- Each allocated user page frame is owned by exactly one PTE and must never be shared by more than one PTE

Restriction

- You may modify only the following files
- kernel/defs.h | ++
- kernel/exec.c | ++
- kernel/fs.c | +
- kernel/pipe.c | +
- kernel/proc.h | +
- kernel/proc.c | ++++
- kernel/riscv.h | +
- kernel/swap.h | +
- kernel/swap.c | ++++++++++++++++++++
- kernel/trap.c | +
- kernel/vm.c | ++++

Notice

- All test cases that result in a **timeout** will be scored as zero
- Please upload project related Q&A to https://github.com/snu-csl/os-pa4/issues
- The grading server is scheduled to be released around next Tuesday

Restriction

- "make submit" command to generate a compressed tar file named xv6-{PANUM}-{STUDENTID}.tar.gz in the ../xv6-riscv-snu directory
- You need to submit a report (Design Document) to server
- Up to 30 submissions are permitted
  - **If you submit multiple times to the queue, only your last submission will be graded, but the submission count will be deducted for each attempt**
- You can use up to 3 slip days during this semester
  - You should explicitly declare the number of slip days you want to use on the QnA board of the submission server before the next project assignment is announced
  - Once slip days have been used, they cannot be canceled later
- Only the version marked FINAL will be considered for the project score

Due date

- **Due: 11:59 PM, November 30** (Sunday)
- Only the upload submitted before the deadline will receive the full credit. 25% of the credit will be deducted for every single day delayed.

Thank you!