Assignment-3

Shivansh Garg (2024MCS2450), Rohit Patidar (2024JCS2042)

April 2025

Introduction

In this assignment, we add the following capabilities to xv6:

- Memory Printer: This prints the number of pages allocated to a user process.
- Page Swapping: Pages are periodically moved to the disk using an adaptive page replacement policy.

Modified Disk Layout

The current disk layout of `xv6` is:

```
[ boot block | sb block | log | inode blocks | free bitmap
| data blocks ]
```

To transfer memory pages to disk, a new partition must be introduced. Add a **swap blocks partition** between the `sb block` and the `log` of the disk layout (refer to `mkfs.c`).

Each swap block should be divided into **swap slots**, where each slot represents eight consecutive disk blocks (since a page is 4096 bytes and one disk block is 512 bytes).
Initialize the swap slot array at boot time. You can allocate 800 slots for this purpose and accordingly adjust `FSSIZE` in `param.h`.

When saving a memory page to disk, store the contents into 8 disk blocks directly. Unlike file system writes, **swap block writes bypass the log layer** (refer to `fs.c`) as they handle volatile memory.

```
sb: size 7400 nblocks 942 ninodes 200 nlog 30 logstart 6402 inodestart 6432 bmap start 8
init: starting sh
$ Ctrl+I is detected by xv6
PID NUM_PAGES
1 3
2 4
```

Figure 1:

Memory Organization

The function `kalloc()` is used to allocate memory pages in `xv6`. To restrict physical memory to 4MB, use:
(placed in `memlayout.h`).

To handle memory overflows, implement page swapping logic in a new file like `pageswap.c`.

Page Replacement

To identify a process and its memory page for swapping (victim process and page):

- Select the process with the highest number of user pages in RAM.
- Add a new field `rss` in `struct proc` to track resident set size.
- If two processes have equal `rss`, pick the one with the lower PID.

When a process exits, clean up its associated swap slots.
To select a victim page:

- Choose a page with the `PTE_P` flag set, and `PTE_A` flag unset.
- The `PTE_P` indicates the page is present in memory.
- The `PTE_A` (accessed flag) is set by the hardware if the page is accessed.

Swapping Mechanism (Algorithm)

The adaptive swapping mechanism works as follows:

1. **Monitor Free Pages:** Continuously check the number of free pages using the kernel's free space list (e.g., via `get_free_pages()` in kernel space).
2. **Trigger Condition:** If the number of free pages falls below Th, trigger the following steps:

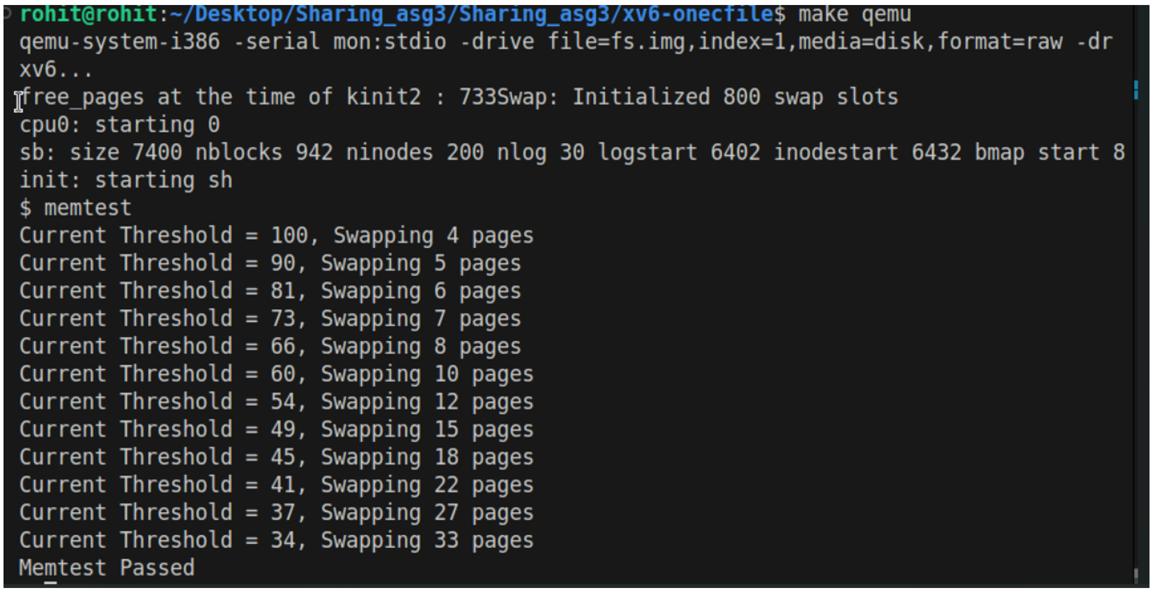
   ```
   Print "Current Threshold = Th, Swapping Npg pages"
   ```

3. **Swap-Out Operation:** Swap out N_{pg} pages to the disk. Typically, these are selected using LRU or similar page replacement policies.
4. **Update Threshold (Th):**

$$Th = \left\lfloor Th \cdot \left(1 - \frac{\beta}{100}\right) \right\rfloor$$

5. **Update Swapping Pages (N_{pg}):**

$$N_{pg} = \left\lfloor \min(LIMIT, N_{pg} \cdot \left(1 + \frac{\alpha}{100}\right)) \right\rfloor$$



```
rohit@rohit:~/Desktop/Sharing_asg3/Sharing_asg3/xv6-onecfile$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -dr
xv6...
free_pages at the time of kinit2 : 733Swap: Initialized 800 swap slots
cpu0: starting 0
sb: size 7400 nblocks 942 ninodes 200 nlog 30 logstart 6402 inodestart 6432 bmap start 8
init: starting sh
$ memtest
Current Threshold = 100, Swapping 4 pages
Current Threshold = 90, Swapping 5 pages
Current Threshold = 81, Swapping 6 pages
Current Threshold = 73, Swapping 7 pages
Current Threshold = 66, Swapping 8 pages
Current Threshold = 60, Swapping 10 pages
Current Threshold = 54, Swapping 12 pages
Current Threshold = 49, Swapping 15 pages
Current Threshold = 45, Swapping 18 pages
Current Threshold = 41, Swapping 22 pages
Current Threshold = 37, Swapping 27 pages
Current Threshold = 34, Swapping 33 pages
Memtest Passed
```

Figure 2:

Role of α and β in System Efficiency

α (Alpha) – Growth Factor for N_{pg}

- Higher α leads to a rapid increase in the number of pages swapped out.
- Helps quickly free up large memory chunks under high memory pressure.
- Risk: May cause excessive swapping, increasing I/O and latency.

β (Beta) – Decay Factor for Th

- Higher β leads to a faster reduction in the threshold Th.
- Reduces frequency of future swaps by making the trigger less sensitive.
- Risk: Might delay swap triggers, risking memory exhaustion.

Combined Effect

Let:

- $\alpha \in [0, 100]$ (growth rate for number of pages to be swapped out)
- $\beta \in [0, 100]$ (decay rate for threshold)
- $Th = 100$ (initial threshold for free pages)
- $N_{pg} = 2$ (initial number of pages to swap out)
- $LIMIT = 100$ (maximum value for N_{pg})

We monitor the number of free pages in memory. If the available free pages fall below the current threshold (Th), we trigger the swapping mechanism. The goal is to manage memory efficiently without affecting system responsiveness.

- **Low α, Low β:** Conservative behavior; small swaps, frequent triggers.
- **High α, Low β:** Aggressive memory freeing with conservative triggering.
- **Low α, High β:** Conservative swapping but progressively rarer.
- **High α, High β:** Aggressive and adaptive—best for dynamic systems.

Example Configuration:

- Initial values: $\alpha = 25$, $\beta = 10$
- N_{pg} increases by 25% after each swap
- Th decreases by 10% after each swap

This configuration makes the system increasingly aggressive in freeing memory, while reducing the chance of repeated swaps.

Makefile Integration

The values of α and β can be set at compile time using the Makefile:

```
CFLAGS += -DALPHA=25 -DBETA=10
```

In the code, use:

```
int alpha = ALPHA;
int beta = BETA;
```

This approach enables easy tuning of the parameters without changing the source code.