

# VE482 Homework 6

Lan Wang 519370910084

## Ex. 1

1. Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB, and 15 KB. Assuming first fit is used, which hole is taken for successive segment requests of: (i) 12 KB, (ii) 10 KB and (iii) 9KB. Repeat for best fit and quick fit.

first fit: 20 KB, 10KB, 18 KB

best fit: 12 KB, 10KB, 9KB

quick fit: 12 KB, 10KB, 9KB

2. If an instruction takes 10 nsec and a page fault takes an additional  $n$  nsec, give a formula for the effective instruction time if page faults occur every  $k$  instructions.

$$\text{effective instruction time} = \frac{10k+n}{k}$$

3. A small computer has four page frames. At the first clock tick, the  $R$  bits are 0111. At  $t$  subsequent clock ticks, the values are 1011, 1010, 1101, 0010, 1010, 1100 and 0001. Assuming the aging algorithm is used with an 8-bit counter what is the value of the four counters after the last tick.

Page	Counter
page 0	01101110
page 1	01001001
page 2	00110111
page 3	10001011

## Ex. 2

### Inverted page table (IPT)

An inverted page table (IPT) is best thought of as an off-chip extension of the TLB which uses normal system RAM. Unlike a true page table, it is not necessarily able to hold all current mappings. The operating system must be prepared to handle misses.

The IPT combines a page table and a frame table into one data structure. At its core is a fixed-size table with the number of rows equal to the number of frames in memory. If there are 4,000 frames, the inverted page table has 4,000 rows. For each row there is an entry for the virtual page number (VPN), the physical page number, and some other data and a means for creating a collision chain.

A hash table may be used to map virtual addresses (and address space/PID information if need be) to an index in the IPT - this is where the collision chain is used. This hash table is known as a hash anchor table. The hashing function is not generally optimized for coverage - raw speed is more desirable. Due to this chosen hashing function, we may experience a lot of collisions in usage, so for each entry in the table the VPN is provided to check if it is the searched entry or a collision.

In searching for a mapping, the hash anchor table is used. If no entry exists, a page fault occurs. Otherwise, the entry is found. Depending on the architecture, the entry may be placed in the TLB again and the memory reference is restarted, or the collision chain may be followed until it has been exhausted and a page fault occurs.

A virtual address in this schema could be split into two, the first half being a virtual page number and the second half being the offset in that page.

A major problem with this design is poor cache locality caused by the hash function. Tree-based designs avoid this by placing the page table entries for adjacent pages in adjacent locations, but an inverted page table destroys spatial locality of reference by scattering entries all over. An operating system may minimize the size of the hash table to reduce this problem, with the trade-off being an increased miss rate.

There is normally one hash table, contiguous in physical memory, shared by all processes. A per-process identifier is used to disambiguate the pages of different processes from each other. It is somewhat slow to remove the page table entries of a given process; the OS may avoid reusing per-process identifier values to delay facing this. Alternatively, per-process hash tables may be used, but they are impractical because of memory fragmentation, which requires the tables to be pre-allocated.

Inverted page tables are used for example on the PowerPC, the UltraSPARC and the IA-64 architecture.

## Multilevel page table

A page table structure that contains mappings for virtual pages. It is done by keeping several page tables that cover a certain block of virtual memory. For example, we can create smaller 1024-entry 4K pages that cover 4M of virtual memory.

This is useful since often the top-most parts and bottom-most parts of virtual memory are used in running a process - the top is often used for text and data segments while the bottom for stack, with free memory in between. The multilevel page table may keep a few of the smaller page tables to cover just the top and bottom parts of memory and create new ones only when strictly necessary.

Now, each of these smaller page tables are linked together by a master page table, effectively creating a tree data structure. There need not be only two levels, but possibly multiple ones. For example, a virtual address in this schema could be split into three parts: the index in the root page table, the index in the sub-page table, and the offset in that page.

## Ex. 3

---

### Security holes [\[1\]](#) [\[2\]](#) [\[3\]](#)

Buffer overflow is ranked #1 on the CWE Top 25 2019 list and is most prevalent in C and C++ programming languages. It is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.

Buffer overflows can often be triggered by malformed inputs. If an anomalous transaction that produces more data than pre-defined buffer size could cause it to write past the end of the buffer. This may overwrite adjacent data or executable code, which results in erratic program behavior. So a rogue program can make use of this security hole by sending an anomalous transaction to cause the erratic program behavior.

A simple example is shown below:

```
# include <stdio.h>
int main(){
    char str[5] = "";
    strcpy(str, "testtesttest"); // buffer overflow
    return 0;
}
```

The string has 16 characters (17 if plus '\0'), but `str` can only hold five characters. This will lead to a buffer overflow, though it does not mean that there will be a segmentation fault. However, if your code is complex enough, the undefined behavior may cause serious consequences. To avoid this, a basic rule is to minimize the complexity in code that performs memory operations.

## Meltdown and Spectre [\[1\]](#) [\[2\]](#) [\[3\]](#) [\[4\]](#) [\[5\]](#)

Meltdown and Spectre exploit critical hardware vulnerabilities, which allow programs to steal data which is currently processed on the computer. A malicious program can exploit Meltdown and Spectre to get data stored in the memory of other running programs, including passwords, photos, emails, instant messages and even business-critical documents.

**Meltdown** breaks the mechanism that keeps applications from accessing arbitrary system memory, so that applications can access system memory. It also uses side channels to obtain the information from the accessed memory location. To minimize the impact of meltdown, isolation of kernel memory from user-mode processes should be increased, which is known as kernel page-table isolation (KPTI). KPTI is based on KAISER (Kernel Address Isolation to have Side-channels Efficiently Removed), which prevents the data from leaking.

**Spectre** tricks other applications into accessing arbitrary locations in their memory. Spectre is harder to exploit than Meltdown, but it is also harder to mitigate. However, it is possible to prevent specific known exploits based on Spectre through software patches. Some general approaches are also proposed, including context-sensitive fencing (CSF). It is a microcode-level defense against multiple variants of Spectre. CSF leverages the ability to dynamically alter the decoding of the instruction stream, to seamlessly inject new micro-ops, including fences, only when dynamic conditions indicate they are needed. This enables the processor to protect against the attack, but with minimal impact on the efficacy of key performance features such as speculative execution.

## Dirty COW [\[1\]](#)

Dirty COW (Dirty copy-on-write) is a computer security vulnerability for the Linux kernel that affected all Linux-based operating systems. It is a local privilege escalation bug that exploits a race condition in the implementation of the copy-on-write mechanism in the kernel's memory-management subsystem. Because of the race condition, with the right timing, a local attacker can exploit the copy-on-write mechanism to turn a read-only mapping of a file into a writable mapping. Although it is a local privilege escalation, remote attackers can use it in conjunction with other exploits that allow remote execution of non-privileged code to achieve remote root access on a computer.

## Ex. 4

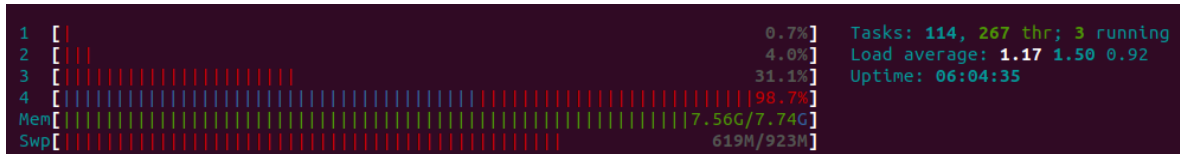
```
#include <limits.h>

int main() {
    static int test[INT_MAX];

    for (int i = 0; i < INT_MAX; ++i) {
        if (i == 0 || i == 1){
            test[i] = 10;
        }else{
            test[i] = test[i-2] * test[i-1];
        }
    }

    return 0;
}
```

Use `htop` to monitor the memory usage:



The screenshot shows the htop utility interface. On the left, a list of processes is shown with their IDs (1, 2, 3, 4) and names ([], [], [], []). The CPU usage bar is visible, showing 0.7% for process 1, 4.0% for process 2, 31.1% for process 3, and 98.7% for process 4. The memory usage bar is also visible, showing 7.56G/7.74G for process 4. The swap usage bar is visible, showing 619M/923M for process 4. On the right, system statistics are displayed: Tasks: 114, 267 thr; 3 running; Load average: 1.17 1.50 0.92; Uptime: 06:04:35.

```
1  [] 0.7% Tasks: 114, 267 thr; 3 running
2  [] 4.0% Load average: 1.17 1.50 0.92
3  [] 31.1% Uptime: 06:04:35
4  [] 98.7%
Mem 7.56G/7.74G
Swp 619M/923M
```