Ex. 1 - Basic Memory

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: 20KB -> 10KB -> 18KB
    - Best Fit: 12KB -> 10KB -> 9KB
    - Quick Fit: 12KB -> 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.**

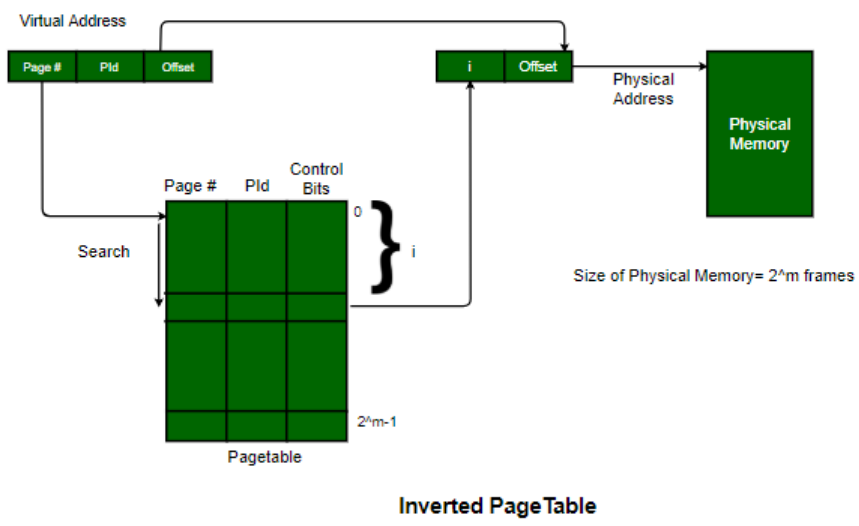
    $10 + n/k(nsec)$
3. **A small computer has four page frames. At the first clock tick, the R bits are 0111. At t subsequent clock tics, the values are 1011, 1010, 1101, 0010, 1010, 1100 and 0001. Assuming the aging algorithm is used with an 8-bit counter what is the value of the four counters after the last tick.**

    Page 0: 01101110
    Page 1: 01001001
    Page 2: 00110111
    Page 3: 10001011

Ex. 2 - Page Tables

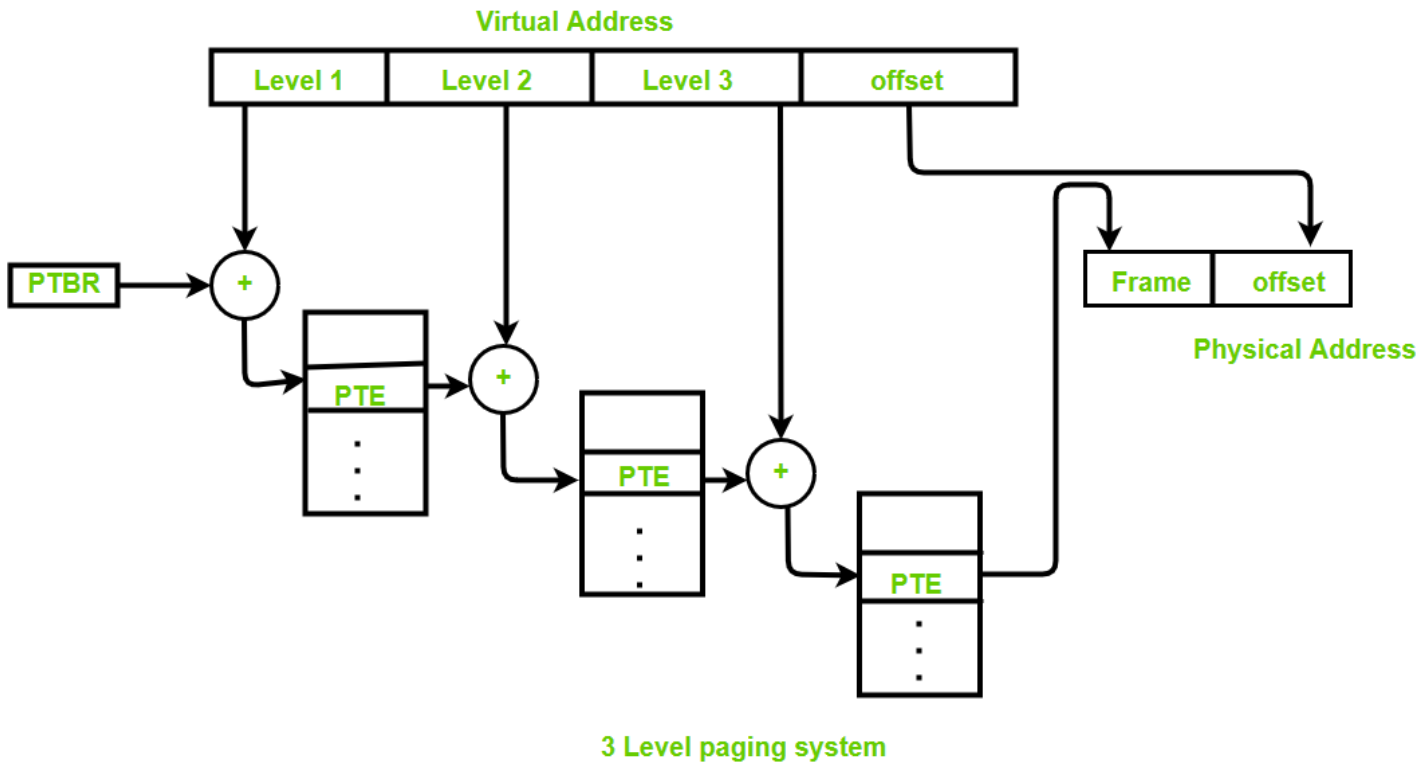
Inverted Page Tables

An alternate approach is to use the **Inverted Page Table** structure that consists of one-page table entry for every frame of the main memory. So the number of page table entries in the Inverted Page Table reduces to the number of frames in physical memory and a single page table is used to represent the paging information of all the processes.



Multilevel Page Tables

Multilevel Paging is a paging scheme which consist of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table are stores actual frame information. Level 1 contain single page table and address of that table is stored in PTBR (Page Table Base Register).



3 Level paging system

Ex. 3 - Research

1. Most vulnerabilities [1] in C are related to buffer overflows and string manipulation. In most cases, this would result in a segmentation fault, but specially crafted malicious input values, adapted to the architecture and environment could yield to arbitrary code execution. You will find below a list of the most common errors and suggested fixes/solutions.

   The first example is `get()` function. The stdio `gets()` function does not check for buffer length and always results in a vulnerability.

```c
#include <stdio.h>
int main () {
    char username[8];
    int allow = 0;
    printf external link("Enter your username, please: ");
    gets(username); // user inputs "malicious"
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) { // has been overwritten by the overflow of the username.
        privilegedAction();
    }
    return 0;
}
```

   If the input size is larger than the buffer size, it may result in an application crash, or in a worse case, information leak.

   The second example is File opening. It is a good idea to check whether a file exists or not before creating it. However, a malicious user might create a file (or worse, a symbolic link to a critical system file) between your check and the moment you actually use the file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MY_TMP_FILE "/tmp/file.tmp"

int main(int argc, char* argv[])
{
    FILE * f;
    if (!access(MY_TMP_FILE, F_OK)) {
        printf external link("File exists!\n");
        return EXIT_FAILURE;
    }
    /* At this point the attacker creates a symlink from /tmp/file.tmp to /etc/passwd */
```

```
    tmpFile = fopen(MY_TMP_FILE, "w");
    if (tmpFile == NULL) {
        return EXIT_FAILURE;
    }
    fputs("Some text ...\n", tmpFile);
    fclose(tmpFile);
    /* You successfully overwrote /etc/passwd (at least if you ran this as root) */
    return EXIT_SUCCESS;
}
```

2. Meltdown and Spectre exploit critical vulnerabilities in modern processors. These hardware vulnerabilities allow programs to steal data which is currently processed on the computer. While programs are typically not permitted to read data from other programs, a malicious program can exploit Meltdown and Spectre to get hold of secrets stored in the memory of other running programs. This might include your passwords stored in a password manager or browser, your personal photos, emails, instant messages and even business-critical documents [2].

   Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system. If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with sensitive information without the chance of leaking the information. This applies both to personal computers as well as cloud infrastructure.

   Spectre breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre. Spectre is harder to exploit than Meltdown, but it is also harder to mitigate.

   There are several different approaches to fix this. There are patches against Meltdown for Linux (KPTI (formerly KAISER)), Windows, and OS X. There is also work to harden software against future exploitation of Spectre, respectively to patch software after exploitation through Spectre (LLVM patch, MSVC, ARM speculation barrier header).

3. **Dirty COW** (Dirty copy-on-write) is a computer security vulnerability for the Linux kernel that affected all Linux-based operating systems, including Android devices, that used older versions of the Linux kernel created before 2018. 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. Computers and devices that still use the older kernels remain vulnerable.

Ex. 4 - Linux

Check `ex4.cpp` and `README.md`.

Reference

---

1. Security Hole ↩

2. Meltdown and Spectre (meltdownattack.com) ↩