**ОБЩАЯ ТЕОРИЯ**

**Что такое /proc и зачем он нужен?**

**/proc** — это виртуальная файловая система:

В Linux есть специальная директория /proc, которая не хранится на жёстком диске, как обычные файлы. Это "виртуальная" система, созданная ядром Linux (мозгом операционной системы), чтобы давать информацию о том, что происходит в системе.

Что внутри /proc:

В /proc есть файлы и папки, которые дают информацию о системе и запущенных процессах. Например:

/proc/cpuinfo — информация о процессоре.

/proc/meminfo — данные о памяти.

/proc/<PID> — папка для каждого процесса, где <PID> — это уникальный номер процесса (Process ID).

Папка /proc/<PID>:

Для каждого запущенного процесса Linux создаёт папку /proc/<PID>.

В этой папке есть файлы, которые рассказывают о процессе: сколько памяти он использует, какие файлы открыл, какие ресурсы занимает и т.д.

Пример файлов в /proc/<PID>:

/proc/<PID>/status — общая информация о процессе (имя, память, состояние).

/proc/<PID>/maps — карта памяти процесса (о ней позже).

/proc/<PID>/pagemap — наш герой, который даёт информацию о страницах памяти

**Что такое /proc/<PID>/pagemap?**

/proc/<PID>/pagemap — это виртуальный файл, созданный ядром Linux, чтобы показать, как виртуальная память процесса связана с физической памятью или swap (подкачкой).

Для чего нужен:

Процессы в Linux используют виртуальную память — это как личное пространство, где программа думает, что у неё есть вся память компьютера. Но на самом деле ядро Linux управляет, какая часть этой виртуальной памяти хранится в физической RAM (оперативной памяти) или в swap (на диске).

/proc/<PID>/pagemap позволяет узнать:

- Какие страницы виртуальной памяти процесса реально находятся в физической памяти.

- Какие страницы выгружены в swap.

- Какие страницы вообще не используются.

- Дополнительные характеристики страниц (например, "грязные" страницы или файловая память).

**Как выглядит файл /proc/<PID>/pagemap?**

Это бинарный файл:

/proc/<PID>/pagemap не текстовый, а бинарный. Ты не можешь открыть его в редакторе (например, nano) и увидеть понятный текст. Если попробовать cat /proc/<PID>/pagemap, ты увидишь "мусор" (странные символы), потому что это набор чисел в машинном формате.

Каждая запись в файле — это 64-битное число (8 байт), которое описывает одну виртуальную страницу процесса.

Структура файла:

Файл содержит записи для всех возможных виртуальных страниц в адресном пространстве процесса, начиная с адреса 0x0 и до максимума (например, 0xFFFFFFFFFFFFFFFF для 64-битных систем).

Каждая страница соответствует виртуальному адресу, кратному размеру страницы (4 КБ). Например:

Адрес 0x0 → страница 0.

Адрес 0x1000 (4096) → страница 1.

Адрес 0x2000 (8192) → страница 2.

Размер файла огромен, потому что он покрывает всё адресное пространство. Например, для 64-битного процесса это 2^64 / 4096 = 2^52 записей, каждая по 8 байт.

Формат каждой записи:

Каждая 64-битная запись (8 байт) содержит информацию о странице. Согласно документации, биты в записи означают следующее:

Бит 63: Страница присутствует (present). Если 1, страница в физической памяти. Если 0, страница либо в swap, либо не замаплена.

Бит 62: Страница в swap (swapped). Если 1, страница выгружена в swap.

Бит 61: Страница файловая или shared-anon (file\_page). Если 1, это страница, связанная с файлом или общей анонимной памятью.

Бит 56: Страница эксклюзивно замаплена (exclusive). Если 1, страница используется только этим процессом (с Linux 4.2).

Бит 55: Страница soft-dirty. Если 1, страница была изменена с последнего сброса (используется для отслеживания изменений).

Биты 57–60: Зарезервированы (всегда 0).

Биты 0–54:

Если страница присутствует (present = 1), это Page Frame Number (PFN) — номер физической страницы в RAM.

Если страница в swap (swapped = 1):

Биты 0–4: тип swap (какой файл подкачки).

Биты 5–54: смещение в файле подкачки.

Если страница не замаплена (present = 0 и swapped = 0), эти биты равны 0.

1. **Оперативная память (RAM, Random Access Memory)**

Оперативная память — это физическая память в компьютере, где хранятся данные, с которыми процессор работает прямо сейчас. Это быстрая память, но она временная — данные теряются при выключении компьютера.

В Linux RAM делится на страницы (обычно по 4 КБ, или 4096 байт), и каждая страница имеет номер, называемый Page Frame Number (PFN).

Как связано с твоей программой:

В /proc/<PID>/pagemap бит 63 (present) показывает, находится ли страница в оперативной памяти:

Если present = 1, страница в RAM, и биты 0–54 дают PFN (номер физической страницы).

2. **Swap (Подкачка)**

Swap — это область на жёстком диске (или SSD), которую Linux использует как "резервную память", когда RAM заполняется. Если в оперативной памяти не хватает места, Linux может временно переместить некоторые страницы из RAM в swap.

Swap медленнее, чем RAM, потому что диск работает медленнее памяти.

Как связано с твоей программой:

В /proc/<PID>/pagemap бит 62 (swapped) показывает, находится ли страница в swap:

Если swapped = 1, страница не в RAM, а на диске, и биты 0–54 содержат:

Биты 0–4: Тип swap (какой файл или раздел подкачки).

Биты 5–54: Смещение в swap (где на диске хранится страница).

if (entry & (1ULL << 63)) {1ULL << 63: Создаёт число с единицей в 63-м бите (0x8000000000000000).

entry & ...: Проверяет, есть ли 1 в 63-м бите записи.

Согласно документации, бит 63 = 1 означает, что страница находится в физической памяти.

uint64\_t pfn = entry & ((1ULL << 55) - 1);(1ULL << 55) - 1: Создаёт маску 0x007FFFFFFFFFFFFF (55 единиц в младших битах).

entry & ...: Выделяет биты 0–54, обнуляя старшие биты.

PFN — это номер физической страницы в RAM (например, 74565).

if (entry & (1ULL << 62)) {1ULL << 62: Создаёт число с единицей в 62-м бите (0x4000000000000000).

Бит 62 = 1 означает, что страница выгружена в swap.

uint64\_t offset = (vaddr / page\_size) \* 8;vaddr / page\_size: Делит виртуальный адрес на размер страницы, чтобы получить индекс страницы (например, 0x555555554000 / 4096).

\* 8: Умножает на 8, так как каждая запись — 8 байт.

offset: Смещение в байтах от начала файла.

if (lseek(fd, offset, SEEK\_SET) == (off\_t)-1) {lseek(fd, offset, SEEK\_SET): Устанавливает позицию в файле на offset байт от начала.

(off\_t)-1: Если lseek возвращает -1, произошла ошибка (например, неверное смещение).

if (sscanf(line, "%lx-%lx %4s %\*s %\*s %\*s %[^\n]", &start, &end, perms, path) < 3) {Парсит строку из /proc/<PID>/maps, извлекая адреса и права.

%lx-%lx: Читает начальный и конечный адреса в шестнадцатеричном формате.

%4s: Читает 4 символа прав (например, r-xp).

%\*s: Пропускает три поля (смещение, устройство, инод), которые нам не нужны.

%[^\n]: Читает остаток строки (путь) до новой строки.

< 3: Если прочитано меньше 3 полей (адреса и права), строка некорректна.

Зачем: Извлекает данные о регионе памяти.

if (strchr(perms, 'r') && perms[3] == 'p') {strchr(perms, 'r'): Ищет символ r в строке прав (чтение).

perms[3] == 'p': Проверяет, что 4-й символ — p (приватный регион).

**Почему выбраны права r и p?**

1. Чтение (r):

Код проверяет наличие r с помощью strchr(perms, 'r'), чтобы убедиться, что регион доступен для чтения.

Почему:

Регионы с правом чтения содержат данные, которые процесс активно использует (например, код, данные, библиотеки).

Если регион не читаемый (например, -wxp), он может быть неинтересен для анализа, так как процесс не может читать данные из него.

В контексте pagemap мы хотим анализировать страницы, которые реально используются, а читаемые регионы — это те, где есть данные.

2. Приватный (p):

Код проверяет perms[3] == 'p', чтобы выбрать только приватные регионы.

Почему:

Приватные регионы (p) — это память, которая принадлежит только этому процессу и не разделяется с другими. Это включает стек, кучу, анонимную память и приватные копии файлов (copy-on-write).

Общие регионы (s) — это память, разделяемая между процессами (например, разделяемые библиотеки или замапленные файлы). Их страницы могут быть сложнее анализировать, так как они зависят от других процессов.

Программа использует **оба файла**:

- /proc/<PID>/maps — чтобы найти замапленные регионы памяти процесса и взять их начальные адреса.

- /proc/<PID>/pagemap — чтобы получить информацию о странице памяти для этих адресов (например, присутствует ли страница в RAM, её PFN).

Она перебирает все регионы из maps с правами чтения и приватности (r--p), а затем для начального адреса каждого региона читает запись из pagemap и выводит её содержимое.

Использование maps в текущем коде нужно, чтобы выбрать реально замапленные адреса, потому что /proc/<PID>/pagemap содержит записи для всех возможных виртуальных страниц (миллиарды), и большинство из них не используются (нули). Без maps пришлось бы проверять всё адресное пространство, что неэффективно.

**/proc/<PID>/maps**

Что это: Текстовый файл, который показывает, какие части виртуальной памяти процесса реально используются (замаплены).

Содержимое: Каждая строка описывает регион памяти:

Диапазон адресов (например, 555555554000-555555555000).

Права доступа (например, r-xp — чтение, выполнение, приватный).

Связанный файл или объект (например, /bin/bash, [stack], [heap]).

Пример:

555555554000-555555555000 r-xp 00000000 08:01 12345 /bin/bash

7ffff7dda000-7ffff7dfd000 r-xp 00000000 08:01 67890 /lib/x86\_64-linux-gnu/libc.so.6

7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0 [stack]

**/proc/<PID>/pagemap**

Что это: Бинарный файл, который содержит информацию о каждой возможной виртуальной странице процесса.

Содержимое: Состоит из 64-битных записей (по 8 байт), каждая из которых описывает одну страницу (4 КБ):

Бит 63: Страница присутствует в физической памяти (present).

Бит 62: Страница в swap (swapped).

Биты 0–54: Номер физической страницы (PFN) или информация о swap.

Особенность:

Записи есть для всех страниц в адресном пространстве (2^64 / 4096 для 64-битных систем), но большинство страниц не замаплены (их записи — нули).

Чтобы найти запись для адреса 0x555555554000, нужно вычислить индекс страницы (vaddr / 4096) и переместиться на смещение index \* 8 байт.

**Как текущая программа использует оба файла?**

1. Читает /proc/<PID>/maps:

Открывает файл с помощью fopen и читает его построчно с помощью fgets.

Парсит строки с помощью sscanf, чтобы получить:

Начальный и конечный адреса региона (start, end).

Права доступа (perms, например, r-xp).

Путь или имя региона (path, например, /bin/bash).

Проверяет, есть ли у региона права на чтение (r) и приватность (p), то есть r--p (например, rwxp тоже подойдёт, если есть r и p).

2. Для каждого подходящего региона:

Берёт начальный адрес региона (start) как виртуальный адрес (vaddr).

Вычисляет смещение в /proc/<PID>/pagemap для этого адреса ((vaddr / page\_size) \* 8).

Читает одну 64-битную запись из /proc/<PID>/pagemap с помощью lseek и read.

3. Декодирует запись, выводя:

- Виртуальный адрес.

- Сырую запись (raw entry).

- - Присутствует ли страница в памяти (present).

- PFN, если страница в памяти.

- Находится ли страница в swap (swapped).

4. Выводит информацию:

Для каждого региона с правами r--p программа показывает информацию о первой странице региона.

