Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:

“Операционные системы”

Лабораторная работа №2

“MemBomb”

Выполнил:

ст. группы N3246 Цыдыпов А.О.

Проверил:

Ханов А.Р.

Санкт-Петербург

2022 г.

Задание:

1. Написать программу выделения памяти и заполнения ее нулями с шагом, равным размеру страницы памяти (mmap, VirtualAlloc)
2. Составить график свободной памяти
3. Ознакомиться с работой демона OOM Killer в Linux
4. Достичь сообщения о невозможности выделить память в Windows

Ссылка на GitHub-репозиторий со всеми исходниками

Листинг программ:

./src/calloc_linux.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sys/mman.h>

#define PAGESIZE 4096

extern int errno;
const char* LOGPATH = "logs/calloc_log.txt";
const char* MEMINFO = "/proc/meminfo";

struct tm* gettime(){
  time_t rawtime;
  struct tm * timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  return timeinfo;
  //printf ( "Current local time and date: %s", asctime (timeinfo) );

}
```

```c
void logging(unsigned mem, FILE* fd){
	if(!fd) {
		printf("fopen Failed!\n");
		exit(-1);
	}
	struct tm * time = gettime();
	fprintf(fd, "%s %u\n", asctime(time), mem);
}

int main(){
	printf("[calloc()  Edition]  Good  Luck  Have  fun!  You're  being membombed...\n");
	FILE* fd = fopen(LOGPATH, "a+");
	char *container;
	unsigned totalmem = 0;
	printf("Input the correct pwd to get bombed.\n");
	char correctPwd[7] = "qwerty";
	char buf[7];
	scanf("%s", &buf);

	struct tm * time = gettime();
	if(strcmp(buf, correctPwd)){
		printf("Incorrect!\n");
		goto HERE;
	}
	printf("Get ready...\n");
	while(1){
	  container = calloc(1, PAGESIZE);
		if(!container) {
			printf("Malloc Failed\n");
			exit(-1);
		}

	}
		totalmem += PAGESIZE;
		logging(totalmem, fd);

	HERE:
	return 0;
}
```

./src/mmap_linux.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sys/mman.h>

#define PAGESIZE 4096

extern int errno;
const char* LOGPATH = "logs/mmap_log.txt";
const char* MEMINFO = "/proc/meminfo";

struct tm* gettime(){
  time_t rawtime;
  struct tm * timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  return timeinfo;
  //printf ( "Current local time and date: %s", asctime (timeinfo) );

}

void logging(unsigned mem, FILE* fd){

      if(!fd) {
            printf("fopen Failed!\n");
            exit(-1);
      }
      struct tm * time = gettime();
      fprintf(fd, "%s %u\n", asctime(time), mem);
}
```

```
int main(){
	printf("[mmap() Edition] Good Luck Have fun! You're being
membombed...\n");
	FILE* fd = fopen(LOGPATH, "a+");
	char *container;
	unsigned totalmem = 0;
	printf("Input the correct pwd to get bombed.\n");
	char correctPwd[7] = "qwerty";
	char buf[7];
	scanf("%s", &buf);

	struct tm * time = gettime();
	if(strcmp(buf, correctPwd)){
		printf("Incorrect!\n");
		goto HERE;
	}
	printf("Get ready...\n");
	while(1){
		container = (char*)mmap(0, PAGESIZE, PROT_WRITE | PROT_READ,
MAP_PRIVATE | MAP_ANONYMOUS, 0, 0);
		memset(container, 0, PAGESIZE);
		if(container == MAP_FAILED) {
			printf("Malloc Failed\n");
			exit(-1);
		}

		totalmem += PAGESIZE;
		logging(totalmem, fd);
	}
	HERE:
	return 0;
}
```

./src/virtualAlloc_windows.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>

int main()
{
	printf("[VirtualAlloc() Edition] Good Luck Have fun! You're being membombed...\n");
	printf("Input the correct pwd to get bombed.\n");
	char correctPwd[7] = "qwerty";
	char buf[7];
	scanf("%s", &buf);

	if(strcmp(buf, correctPwd)){
		printf("Incorrect!\n");
		return 0;
	}
	printf("Get ready...\n");
	SYSTEM_INFO info;
	GetSystemInfo(&info);
	size_t PageSize = info.dwPageSize;
	while (1) {
		LPVOID container = VirtualAlloc(0, PageSize, MEM_COMMIT, PAGE_READWRITE);
	}
	return 0;
}
```

OOM-killer (out of memory killer) - это такой демон в OS Linux, который помогает операционной системе при нехватке памяти не завершаться аварийно. Вместо этого, он убивает процесс, который потребляет слишком много памяти, тем самым спасая ядро.

Выбор процесса для убийства происходит неслучайным образом. Выбор осуществляется исходя из метрики *oom_score*, умноженной на количество используемой памяти. Процессы, связанные с привилегированным пользователем, имеют более низкую оценку и меньше шансов на принудительное завершение.

Результаты

calloc():

Убийство OOM-killer'ом MemBomb:

```
[mertz@arch 2_lab]$ echo -e "qwerty" | ./calloc
[calloc() Edition] Good Luck Have fun! You're being membombed...
Input the correct pwd to get bombed.
Get ready...
Killed
[mertz@arch 2_lab]$
```

На 3-ем скриншоте видно, как после пика OOM-killer убил процесс и память освободилась.

mmap():

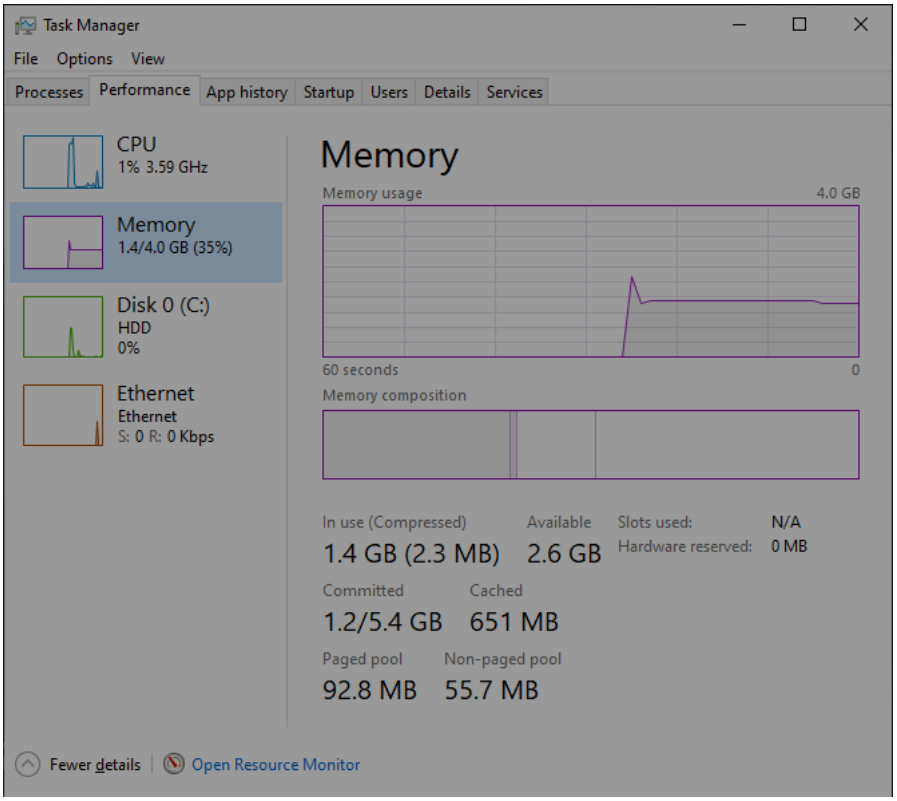
Выделяя память с помощью *mmap()* OOM-killer не сработал. Через какое-то время Linux просто ушел в перезагрузку.

virtualAlloc():

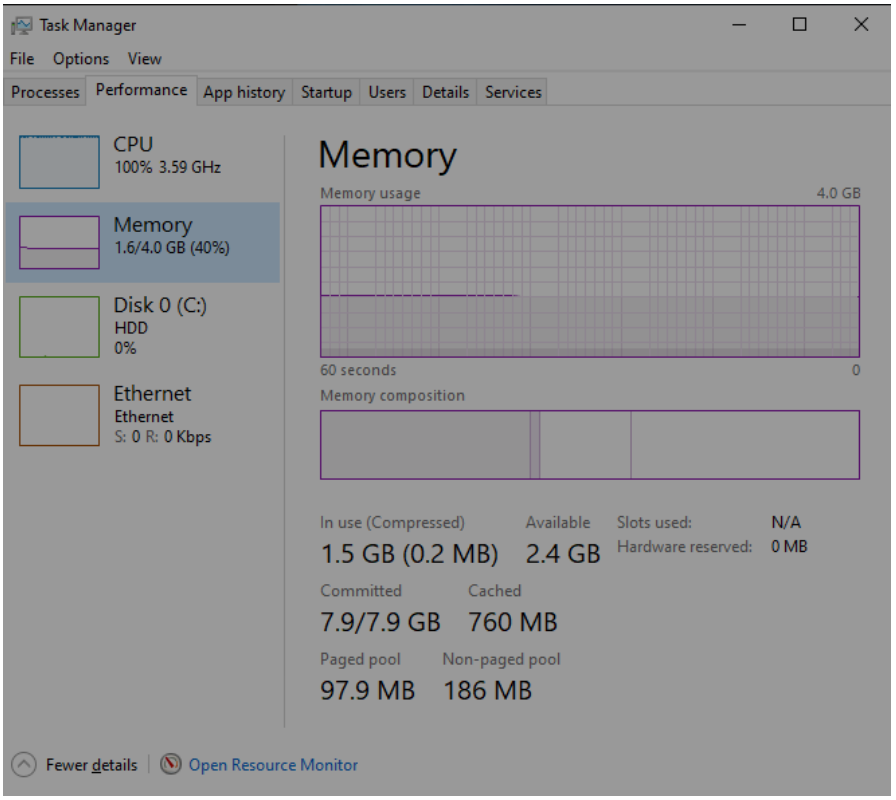
В Windows 10 (4gm RAM allocated) после примерно 10 секунд работы MemBomb ОС тоже уходит в перезагрузку. При большем количестве изначально доступной оперативной

памяти Windows начинает закрывать приложения и в конце, закрыв уже и саму MemBomb, аналогично, уходит в перезагрузку.

До запуска:



После:



Вывод

Linux объективно лучше справился с MemBomb, хоть и его тоже получилось увести в перезагрузку. В Windows пользоваться системой становится невозможно уже после нескольких секунд запущенной MemBomb.