ФГАОУ ВО «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Мегафакультет компьютерных технологий и управления

Факультет программной инженерии и компьютерной техники

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине «Операционные системы»

Выполнил студент:

Киселев Артем Олегович

группа: P33113

Проверил:

Покид А.А

Санкт-Петербург, 2020 г.

1 ВАРИАНТ

Разработать программу на языке C, которая осуществляет следующие действия:

– Создает область памяти размером **A=262** мегабайт, начинающихся с адреса **B=0xACDC45F1** (если возможно) при помощи **C=malloc** заполненную случайными числами /dev/urandom в **D=113** потоков. Используя системные средства мониторинга определите адрес начала в адресном пространстве процесса и характеристики выделенных участков памяти. Замеры виртуальной/физической памяти необходимо снять:

а) До аллокации

б) После аллокации

в) После заполнения участка данными

г) После деаллокации

– Записывает область памяти в файлы одинакового размера **E=26** мегабайт с использованием **F=nocache** обращения к диску. Размер блока ввода-вывода **G=113** байт. Преподаватель выдает в качестве задания последовательность записи/чтения блоков **H=random**

– Генерацию данных и запись осуществлять в бесконечном цикле.

– В отдельных **I=129** потоках осуществлять чтение данных из файлов и подсчитывать агрегированные характеристики данных - **J=sum**.

– Чтение и запись данных в/из файла должна быть защищена примитивами синхронизации **K=cv**.

По заданию преподавателя изменить приоритеты потоков и описать изменения в характеристиках программы.

Измерить значения затраченного процессорного времени на выполнение программы и на операции ввода-вывода используя системные утилиты.

Отследить трассу системных вызовов.

Используя stap построить графики системных характеристик.

2 ВЫПОЛНЕНИЕ

2.1 Листинг программы

```c
#define _XOPEN_SOURCE 500
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <fcntl.h>
```

```c
#include <stdint.h>

#include <search.h>
#include <time.h>

#define checkMalloc(memory, err) if(memory == NULL){ printf("%s is not allocated", err); exit(1); }

#define A 262
// #define B 0xACDC45F1
// #define C malloc
#define D 113
#define E 26
#define G 113
// #define H random
#define I 129
// #define J sum
// #define K cv

#define RANDOM "/dev/urandom"
#define RANDOM_MODE "r"

#define MB_TO_BYTE 1024*1024
#define A_BYTE A*MB_TO_BYTE // 274726912
#define E_BYTE E*MB_TO_BYTE

#define COUNT_BYTE_TO_WRITE A_BYTE/D
#define COUNT_FILE A/E

#define BLOCK_SIZE 4096
#define BLOCK_COUNT E_BYTE / BLOCK_SIZE

#define FILENAME "file_%d.bin"
#define FLAG_TO_CREATE_FILE O_RDWR | O_CREAT | O_FSYNC

typedef struct {
    size_t start;
    size_t stop;
} genDataThread;

typedef struct {
    int number;
} writeFileData;
```

```
49
50 typedef struct {
51     int number;
52     int fileNumber;
53 } readFromFileData;
54
55 typedef struct {
56     int number;
57     size_t start;
58     size_t end;
59 } block;
60
61 pthread_mutex_t mutexs[COUNT_FILE];
62 pthread_cond_t conds[COUNT_FILE];
63 size_t isWrite[COUNT_FILE] = {0};
64
65 block MBlockRead[BLOCK_COUNT];
66 block MBlockWrite[BLOCK_COUNT];
67
68 int *files;
69 int fp;
70 uint8_t *memory;
71
72 void generateRandomData();
73
74 void writeToFile();
75
76 void readFromFile();
77
78 void shuffle(block *arr, int N) {
79     srand(time(NULL));
80     for (int i = N - 1; i >= 1; i--) {
81         int j = rand() % (i + 1);
82         block tmp = arr[j];
83         arr[j] = arr[i];
84         arr[i] = tmp;
85     }
86 }
87
88 int main() {
89     short firstRun = 1;
90
91     for (int i = 0; i < BLOCK_COUNT; i++) {
```

```
92          MBlockWrite[i].number = i;
93          MBlockWrite[i].start = i * BLOCK_SIZE;
94          MBlockWrite[i].end = (i + 1) * BLOCK_SIZE;
95
96          MBlockRead[i].number = i;
97          MBlockRead[i].start = i * BLOCK_SIZE;
98          MBlockRead[i].end = (i + 1) * BLOCK_SIZE;
99      }
100
101     shuffle(MBlockRead, BLOCK_COUNT);
102
103     files = (int*)malloc(COUNT_FILE * sizeof(int));
104     char filename[35];
105     for (int i = 0; i < COUNT_FILE; i++) {
106         snprintf(filename, 35, FILENAME, i);
107         files[i] = open(filename, FLAG_TO_CREATE_FILE);
108         pthread_cond_init(&(conds[i]), NULL);
109         pthread_mutex_init(&(mutexs[i]), NULL);
110     }
111
112     fp = open(RANDOM, O_RDONLY);
113
114     while (1) {
115         memory = (uint8_t *) malloc(A_BYTE);
116         checkMalloc(memory, "Memory")
117
118         generateRandomData();
119
120         shuffle(MBlockWrite, BLOCK_COUNT);
121         writeToFile();
122
123         if (firstRun) {
124             firstRun = 0;
125             readFromFile();
126         }
127
128         free(memory);
129     }
130 //     return 0;
131 }
132
133 void *threadRandomGenerate(void *thread_data) {
134     genDataThread *data = (genDataThread *) thread_data;
```

```
135 //     FILE *fp = fopen(RANDOM, RANDOM_MODE);
136     for (size_t i = data->start; i < data->stop; i++) {
137         uint8_t random = 0;
138         read(fp, &random, 1);
139 //          fread(&random, sizeof(uint8_t), 1, fp);
140         memory[i] = random;
141     }
142 //     fclose(fp);
143     return NULL;
144 }
145
146 void generateRandomData() {
147     pthread_t *threads = (pthread_t *) malloc(D * sizeof(pthread_t));
148     genDataThread *threadData = (genDataThread *) malloc(D * sizeof(genDataThread)
    );
149
150     checkMalloc(threads, "Thread's Generate")
151     checkMalloc(threadData, "ThreadData Generate")
152
153     for (size_t i = 0; i < D - 1; i++) {
154         threadData[i].start = i * COUNT_BYTE_TO_WRITE;
155         threadData[i].stop = (i + 1) * COUNT_BYTE_TO_WRITE;
156     }
157
158     for (int i = 0; i < D; i++)
159         pthread_create(&(threads[i]), NULL, threadRandomGenerate, &threadData[i]);
160
161     for (int i = 0; i < D; i++)
162         pthread_join(threads[i], NULL);
163
164     free(threads);
165 //     free(threadData);
166 }
167
168 void *threadWriteToFile(void *thread_data) {
169     writeFileData *data = (writeFileData *) thread_data;
170     int thread = data->number;
171     pthread_mutex_t *mutex = &(mutexs[thread]);
172     pthread_cond_t *cond = &conds[thread];
173     int toWrite = files[thread];
174
175     if (toWrite == -1) {
176         printf("Creat or write file is have error.\n");
```

```
177         exit(1);
178     }
179
180     int offset = thread * E_BYTE;
181     for (int j = 0; j < BLOCK_COUNT; j++) {
182         size_t start = MBlockWrite[j].start;
183         size_t end = MBlockWrite[j].end;
184
185         pthread_mutex_lock(mutex);
186         isWrite[thread] = 1;
187
188         for (size_t k = start; k < end; k += G) {
189             uint8_t arrToWrite[G];
190             for (int h = 0; h < G; ++h) {
191                 arrToWrite[h] = memory[offset + k + h];
192             }
193             pwrite(toWrite, arrToWrite, G, k);
194             fsync(toWrite);
195         }
196
197         isWrite[thread] = 0;
198         pthread_cond_signal(cond);
199         pthread_mutex_unlock(mutex);
200
201     }
202
203     return NULL;
204 }
205
206 void writeToFile() {
207     pthread_t *threads = (pthread_t *) malloc(COUNT_FILE * sizeof(pthread_t));
208     writeFileData *threadData = (writeFileData *) malloc(COUNT_FILE * sizeof(
    writeFileData));
209
210     checkMalloc(threads, "Thread's Write")
211     checkMalloc(threadData, "ThreadData Write")
212
213     for (int i = 0; i < COUNT_FILE; ++i) {
214         threadData[i].number = i;
215     }
216
217     for (int i = 0; i < COUNT_FILE; ++i) {
218         pthread_create(&(threads[i]), NULL, threadWriteToFile, &threadData[i]);
```

```
219     }
220
221     for (int i = 0; i < COUNT_FILE; i++)
222         pthread_join(threads[i], NULL);
223
224     free(threads);
225     free(threadData);
226 }
227
228 void *readFileThread(void *thread_data) {
229     readFromFileData *data = (readFromFileData *) thread_data;
230
231     int fileNumber = data->fileNumber;
232     pthread_mutex_t *mutex = &(mutexs[fileNumber]);
233     pthread_cond_t *cond = &(conds[fileNumber]);
234     int fp = files[fileNumber];
235
236     while (1) {
237         unsigned long long sum = 0;
238         for (int i = 0; i < BLOCK_COUNT; i++) {
239             size_t start = MBlockRead[i].start;
240             size_t end = MBlockRead[i].end;
241
242             pthread_mutex_lock(mutex);
243             while (isWrite[fileNumber]) {
244                 pthread_cond_wait(cond, mutex);
245             }
246
247             lseek(fp, start, SEEK_SET);
248             for (size_t j = start; j < end; j++) {
249                 uint8_t numb;
250                 read(fp, &numb, 1);
251                 sum += numb;
252             }
253             lseek(fp, 0, SEEK_SET);
254
255             pthread_mutex_unlock(mutex);
256         }
257         printf("Thread %d complete in file %d sum = %llu\n", data->number,
    fileNumber, sum);
258
259     }
260     return NULL;
```

```
261 }
262
263 void readFromFile() {
264     pthread_t *threads = (pthread_t *) malloc(I * sizeof(pthread_t));
265     readFromFileData *threadData = (readFromFileData *) malloc(I * sizeof(
    readFromFileData));
266
267     checkMalloc(threads, "Thread's Read")
268     checkMalloc(threadData, "ThreadData Read")
269
270     for (int i = 0; i < I; ++i) {
271         threadData[i].number = i;
272         threadData[i].fileNumber = i % COUNT_FILE;
273     }
274
275     for (int i = 0; i < I; ++i) {
276         pthread_create(&(threads[i]), NULL, readFileThread, &threadData[i]);
277     }
278 }
```

2.2 Замеры

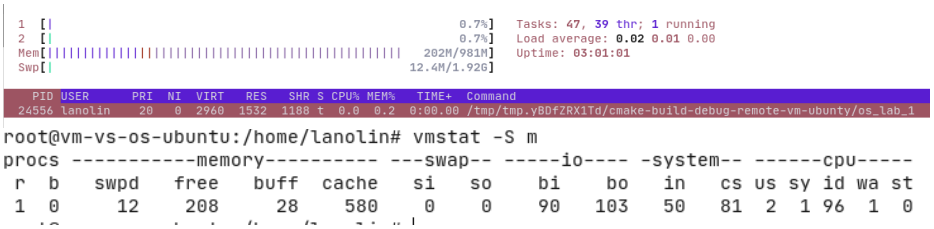

Рис. 2.1 – Замеры памяти до аллокации htop и vmstat

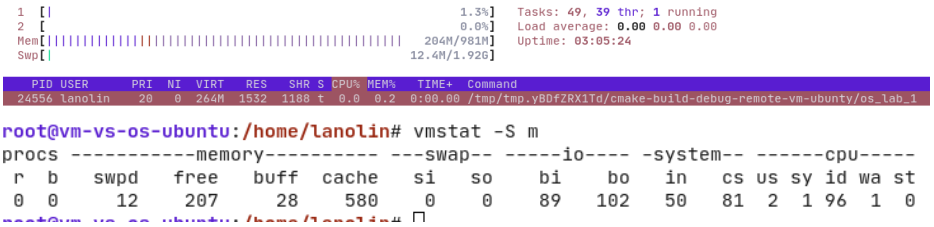

Рис. 2.2 – Замеры памяти после аллокации htop и vmstat

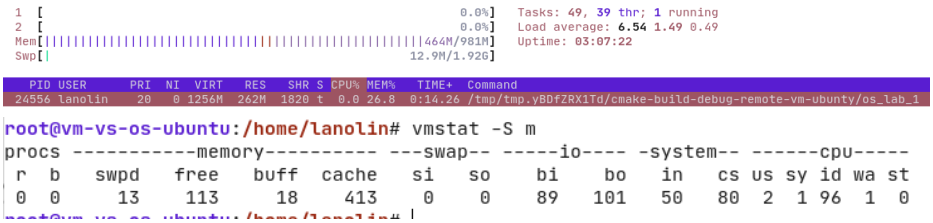

Рис. 2.3 – Замеры памяти после заполения данными htop и vmstat

Рис. 2.4 – Замер после деаллокации памяти htop

Листинг 1: strace программы

```
execve("/tmp/tmp.yBDfZRX1Td/cmake-build-debug-remote-vm-ubunty/os_lab_1", ["/tmp/
    tmp.yBDfZRX1Td/cmake-build-"...], 0x7ffca7ebf060 /* 34 vars */) = 0
brk(NULL)                                = 0x5560ceb9e000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff5fadb390) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=56888, ...}) = 0
mmap(NULL, 56888, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ffa5623c000
close(3)                                 = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 
    832) = 832
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@
    \261\23\327o"..., 68, 824) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    x7ffa5623a000
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@
    \261\23\327o"..., 68, 824) = 68
mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ffa56217000
mmap(0x7ffa5621e000, 69632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_
    DENYWRITE, 3, 0x7000) = 0x7ffa5621e000
mmap(0x7ffa5622f000, 20480, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0
    x18000) = 0x7ffa5622f000
mmap(0x7ffa56234000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_
    DENYWRITE, 3, 0x1c000) = 0x7ffa56234000
mmap(0x7ffa56236000, 13432, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_
    ANONYMOUS, -1, 0) = 0x7ffa56236000
close(3)                                 = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

```
22 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0 >\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832)
     = 832
23 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
    784, 64) = 784
24 pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32,
     848) = 32
25 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y
    \377\t\334"..., 68, 880) = 68
26 fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
27 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
    784, 64) = 784
28 pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32,
     848) = 32
29 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y
    \377\t\334"..., 68, 880) = 68
30 mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ffa56025000
31 mprotect(0x7ffa5604a000, 1847296, PROT_NONE) = 0
32 mmap(0x7ffa5604a000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_
    DENYWRITE, 3, 0x25000) = 0x7ffa5604a000
33 mmap(0x7ffa561c2000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0
    x19d000) = 0x7ffa561c2000
34 mmap(0x7ffa5620d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_
    DENYWRITE, 3, 0x1e7000) = 0x7ffa5620d000
35 mmap(0x7ffa56213000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_
    ANONYMOUS, -1, 0) = 0x7ffa56213000
36 close(3)                                = 0
37 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    x7ffa56022000
38 arch_prctl(ARCH_SET_FS, 0x7ffa56022740) = 0
39 mprotect(0x7ffa5620d000, 12288, PROT_READ) = 0
40 mprotect(0x7ffa56234000, 4096, PROT_READ) = 0
41 mprotect(0x5560cd042000, 4096, PROT_READ) = 0
42 mprotect(0x7ffa56277000, 4096, PROT_READ) = 0
43 munmap(0x7ffa5623c000, 56888)             = 0
44 set_tid_address(0x7ffa56022a10)          = 2303
45 set_robust_list(0x7ffa56022a20, 24)      = 0
46 rt_sigaction(SIGRTMIN, {sa_handler=0x7ffa5621ebf0, sa_mask=[], sa_flags=SA_
    RESTORER|SA_SIGINFO, sa_restorer=0x7ffa5622c3c0}, NULL, 8) = 0
47 rt_sigaction(SIGRT_1, {sa_handler=0x7ffa5621ec90, sa_mask=[], sa_flags=SA_RESTORER
    |SA_RESTART|SA_SIGINFO, sa_restorer=0x7ffa5622c3c0}, NULL, 8) = 0
48 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
49 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) =
     0
```

```
50 brk(NULL)                                    = 0x5560ceb9e000
51 brk(0x5560cebbf000)                          = 0x5560cebbf000
52 openat(AT_FDCWD, "/home/lanolin/out/file_0.bin", O_RDWR|O_CREAT|O_SYNC, 000) = 3
53 openat(AT_FDCWD, "/home/lanolin/out/file_1.bin", O_RDWR|O_CREAT|O_SYNC, 004) = 4
54 openat(AT_FDCWD, "/home/lanolin/out/file_2.bin", O_RDWR|O_CREAT|O_SYNC, 010) = 5
55 openat(AT_FDCWD, "/home/lanolin/out/file_3.bin", O_RDWR|O_CREAT|O_SYNC, 014) = 6
56 openat(AT_FDCWD, "/home/lanolin/out/file_4.bin", O_RDWR|O_CREAT|O_SYNC, 020) = 7
57 openat(AT_FDCWD, "/home/lanolin/out/file_5.bin", O_RDWR|O_CREAT|O_SYNC, 024) = 8
58 openat(AT_FDCWD, "/home/lanolin/out/file_6.bin", O_RDWR|O_CREAT|O_SYNC, 030) = 9
59 openat(AT_FDCWD, "/home/lanolin/out/file_7.bin", O_RDWR|O_CREAT|O_SYNC, 034) = 10
60 openat(AT_FDCWD, "/home/lanolin/out/file_8.bin", O_RDWR|O_CREAT|O_SYNC, 040) = 11
61 openat(AT_FDCWD, "/home/lanolin/out/file_9.bin", O_RDWR|O_CREAT|O_SYNC, 044) = 12
62 mmap(NULL, 274731008, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
       x7ffa45a21000
63 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0
       x7ffa45220000
64 mprotect(0x7ffa45221000, 8388608, PROT_READ|PROT_WRITE) = 0
65 clone(child_stack=0x7ffa45a1ffb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_
       SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD
       _CLEARTID, parent_tid=[2304], tls=0x7ffa45a20700, child_tidptr=0x7ffa45a209d0)
       = 2304
66 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0
       x7ffa44a1f000
67 mprotect(0x7ffa44a20000, 8388608, PROT_READ|PROT_WRITE) = 0
68 clone(child_stack=0x7ffa4521efb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_
       SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD
       _CLEARTID, parent_tid=[2305], tls=0x7ffa4521f700, child_tidptr=0x7ffa4521f9d0)
       = 2305
69 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0
       x7ffa4421e000
70 mprotect(0x7ffa4421f000, 8388608, PROT_READ|PROT_WRITE) = 0
71 clone(child_stack=0x7ffa44a1dfb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_
       SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD
       _CLEARTID, parent_tid=[2306], tls=0x7ffa44a1e700, child_tidptr=0x7ffa44a1e9d0)
       = 2306
72 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0
       x7ffa3f7ff000
73 mprotect(0x7ffa3f800000, 8388608, PROT_READ|PROT_WRITE) = 0
```

Листинг 2: stap

```
1 Incremental reporting (max 20 lines, sorted by count) every 10 s
2 Period time elapsed: 11236 ms, 56686 events, 39175 after filtering.
3 TID                    COUNT (Hz)       EVENT
```

```
--                    -- -- -- -- --       -- --
os_lab_1(60859)       1992 (177.28)   syscall.fsync
os_lab_1(60859)       1992 (177.28)   syscall.pwrite
os_lab_1(60858)       1979 (176.13)   syscall.pwrite
os_lab_1(60858)       1979 (176.13)   syscall.fsync
os_lab_1(60862)       1977 (175.95)   syscall.pwrite
os_lab_1(60862)       1977 (175.95)   syscall.fsync
os_lab_1(60861)       1975 (175.77)   syscall.fsync
os_lab_1(60861)       1975 (175.77)   syscall.pwrite
os_lab_1(60863)       1964 (174.79)   syscall.pwrite
os_lab_1(60863)       1964 (174.79)   syscall.fsync
os_lab_1(60860)       1954 (173.90)   syscall.fsync
os_lab_1(60860)       1954 (173.90)   syscall.pwrite
os_lab_1(60866)       1950 (173.54)   syscall.fsync
os_lab_1(60866)       1950 (173.54)   syscall.pwrite
os_lab_1(60865)       1944 (173.01)   syscall.fsync
os_lab_1(60865)       1944 (173.01)   syscall.pwrite
os_lab_1(60864)       1931 (171.85)   syscall.fsync
os_lab_1(60864)       1931 (171.85)   syscall.pwrite
os_lab_1(60867)       1921 (170.96)   syscall.pwrite
os_lab_1(60867)       1921 (170.96)   syscall.fsync
Period time elapsed: 10000 ms, 117823 events, 83984 after filtering.
TID                   COUNT (Hz)      EVENT
--                    -- -- -- -- --       -- --
os_lab_1(60867)       4266 (426.60)   syscall.pwrite
os_lab_1(60867)       4266 (426.60)   syscall.fsync
os_lab_1(60860)       4235 (423.50)   syscall.fsync
os_lab_1(60860)       4235 (423.50)   syscall.pwrite
os_lab_1(60865)       4233 (423.30)   syscall.fsync
os_lab_1(60865)       4233 (423.30)   syscall.pwrite
os_lab_1(60859)       4213 (421.30)   syscall.fsync
os_lab_1(60859)       4213 (421.30)   syscall.pwrite
os_lab_1(60862)       4198 (419.80)   syscall.pwrite
os_lab_1(60862)       4198 (419.80)   syscall.fsync
os_lab_1(60863)       4190 (419.00)   syscall.pwrite
os_lab_1(60863)       4190 (419.00)   syscall.fsync
os_lab_1(60858)       4184 (418.40)   syscall.pwrite
os_lab_1(60858)       4184 (418.40)   syscall.fsync
os_lab_1(60866)       4183 (418.30)   syscall.fsync
os_lab_1(60866)       4183 (418.30)   syscall.pwrite
os_lab_1(60861)       4152 (415.20)   syscall.fsync
os_lab_1(60861)       4152 (415.20)   syscall.pwrite
os_lab_1(60864)       4138 (413.80)   syscall.fsync
```

```
os_lab_1(60864)          4138 (413.80)    syscall.pwrite
Period time elapsed: 9999 ms, 104893 events, 65330 after filtering.
TID                      COUNT (Hz)       EVENT
------                   ---------- ------------
os_lab_1(60858)          3298 (329.83)    syscall.pwrite
os_lab_1(60858)          3298 (329.83)    syscall.fsync
os_lab_1(60861)          3291 (329.13)    syscall.fsync
os_lab_1(60861)          3291 (329.13)    syscall.pwrite
os_lab_1(60867)          3285 (328.53)    syscall.pwrite
os_lab_1(60867)          3285 (328.53)    syscall.fsync
os_lab_1(60866)          3281 (328.13)    syscall.fsync
os_lab_1(60866)          3281 (328.13)    syscall.pwrite
os_lab_1(60862)          3276 (327.63)    syscall.pwrite
os_lab_1(60862)          3276 (327.63)    syscall.fsync
os_lab_1(60864)          3261 (326.13)    syscall.fsync
os_lab_1(60864)          3261 (326.13)    syscall.pwrite
os_lab_1(60865)          3260 (326.03)    syscall.fsync
os_lab_1(60865)          3260 (326.03)    syscall.pwrite
os_lab_1(60860)          3245 (324.53)    syscall.fsync
os_lab_1(60860)          3245 (324.53)    syscall.pwrite
os_lab_1(60859)          3243 (324.33)    syscall.fsync
os_lab_1(60859)          3243 (324.33)    syscall.pwrite
os_lab_1(60863)          3225 (322.53)    syscall.pwrite
os_lab_1(60863)          3225 (322.53)    syscall.fsync
```

3 ВЫВОД

Понял основы написания многопоточных программ на C. Попробовал некоторые программы мониторинга системы Linux.