

بسم تعالی



سیستم‌های عامل پیشرفته

تمرین اول

استاد:

دکتر حسین اسدی

نویسنده :

محمد هومان کشوری

شماره دانشجویی :

99105667

تمرینات تئوری

سوال 1.

در سیستم داده شده ۱۰۰۰۰۰۰۰ چانک مموری به سایز ۱۰۲۴KB تخصیص داده می‌شود که یعنی عملاً ۱۰۰۰GB مموری اما به این خاطر که کلاً سیستم ما دارای 256GB حافظه است سربرار خیلی زیادی به حافظه وارد می‌شود.

حال زمانی TLB Shutdown داریم که نیاز باشد قسمتی از حافظه را Deallocate کنیم و عملاً نیاز به invalid کردن محتویات TLB در هسته‌های متفاوت باشد.

حال اطلاعات و اندازه page سیستم در TLB Shutdown تاثیر دارد. حال در صورتی که هر پیج را ۱ مگابایت در نظر بگیریم، در هر free نیاز به فلاش ۲ page table entry داریم. حال می‌دانیم هر Shutdown جداگانه در هر cpu انجام می‌شود پس یعنی به ازای هر فلاش نیاز به 4 TLB Shutdown جداگانه داریم.

$$1000000 * 1000 * 4$$

البته احتمالاً تعدادی ترد در الوکیت کردن به مشکل بخورند و نیاز باشد از قسمت‌های الوکیت شده قبل از اینکه free شوند، حافظه را بگیرند چون به صورت protected نیز از آن استفاده نمی‌کنیم.

این بستگی به موازی بودن و زمان دقیق اجرای ریسه‌ها دارد.

حال به راهکارهای بهبود می‌رسیم.

۱. یکی از راه‌های کاهش TLB Shutdown کاهش تعداد allocate , deallocate است چرا که این سیستم کال‌ها نیاز به مدیریت حافظه بالایی دارند و عملاً با هر allocate/deallocate محتوای آن قسمت حافظه invalid می‌شود و نیاز به TLB Shutdown داریم.

۲. استفاده از memory pool یعنی گرفتن یک مقدار حافظه بزرگ به صورت کلی و مدیریت آن به صورت درون‌برنامه‌ای نیاز به مدیریت حافظه سیستم‌عامل را کم می‌کند پس می‌توانیم از همان اول مقدار کلی حافظه را به جای چندین مرتبه تخصیص دهیم.

۳. می‌توان با تغییر سیستم‌عامل در هر چند free یکبار TLB Shutdown انجام داد و سیگنال اینتراپت را فرستاد البته این کار توصیه نمی‌شود چرا که از نظر امنیتی مشکلاتی ایجاد می‌کند. با این کار به ازای هر ۱۰۰۰ تا free کلی می‌توانیم یک TLB Shutdown داشته باشیم. حال این قسمت‌ها را می‌توانیم صرفاً علامت‌گذاری کنیم و در نهایت یکجا به سیستم‌عامل برگردانیم (مانند عملکرد Garbage Collector در جاوا).

تمرینات عملی

سوال 1.1 .

برای حل این سوال ابتدا یک بش اسکرپت می‌نویسیم تا برنامه مورد نظر ما را کامپایل کند و ایشن pg- را به آن اضافه می‌کنیم که به perf اجازه مانیتور کردن خروجی را می‌دهد.

```
gcc ./protect.c -o myprogram -pg  
  
sudo perf stat -e  
faults,page-faults,dTLB-loads,dTLB-load-misses,iTLB-loads,iTLB-load-misses  
,context-switches ./myprogram
```

همانطور که مشاهده می‌کنید در اسکرپت نوشته شده، تعداد کل page fault ها و ایونت‌های مربوط به TLB آورده شده است.

ابتدا برنامه اصلی را بررسی می‌کنیم و سپس نتایج بدست آمده را تحلیل می‌کنیم.

در خطوط ابتدایی به واسطه آرگومان‌های ورودی مشخص می‌کنیم هر کدام از ریشه‌های C , D در حالت خواندن قرار بگیرند یا نوشتن، در صورتی که هیچ آرگومانی به برنامه داده نشود، هر دو در ابتدا در حالت خواندن قرار می‌گیرند. سپس برنامه ۴ ریشه ساخته، که توابع آنها را توضیح می‌دهیم.

```

65  int main(int argc, char** argv) {
66      if (argv[1] != NULL)
67          if (strcmp(argv[1], "C_write"))
68              C_read = 0;
69          else if (strcmp(argv[1], "D_write"))
70              D_read = 0;
71
72      if (argv[2] != NULL)
73          if (strcmp(argv[2], "C_write"))
74              C_read = 0;
75          else if (strcmp(argv[2], "D_write"))
76              D_read = 0;
77
78      // allocate a page size of 4kB with malloc
79      printf("this is the process : %d", getpid());
80      pthread_t A;
81      pthread_t B;
82      pthread_t C;
83      pthread_t D;
84
85      pthread_create(&A, NULL, thread_A, NULL);
86      pthread_create(&B, NULL, thread_B, NULL);
87      sleep(1);
88      pthread_create(&C, NULL, thread_C, NULL);
89      pthread_create(&D, NULL, thread_D, NULL);
90
91      pthread_join(A, NULL);
92      pthread_join(B, NULL);
93      pthread_join(C, NULL);
94      pthread_join(D, NULL);
95      free(ptr);
96
97      return 0;
98  }

```

در قطعه کد زیر تابع ریسه A آورده شده که با استفاده از تابع `posix_memalign` یک فضا به اندازه `PAGE_SIZE` می‌گیرد. این تابع همانند تابع `malloc` عمل می‌کند با این

تفاوت که مطمئن می‌شود اندازه داده شده با اندازه صفحه در سیستم align می‌شود پس مشکلات ناتوانی در رزرو حافظه را نخواهیم داشت.

```
15 void* thread_A() {
16     int ret = posix_memalign(&ptr, PAGE_SIZE, PAGE_SIZE);
17     if (ret != 0 || ptr == NULL) {
18         printf("Error: failed to allocate memory\n");
19         exit(1);
20     }
21 }
```

در قطعه کد زیر که مطعلق به ترد B است، ابتدا قسمت مورد نظر از نوشتن و خواندن محافظت شده و سپس در آن قسمت مقداری نوشته شده و محافظت از نوشتن برداشته می‌شود (دیگر نمی‌توان در این قسمت از حافظه نوشت).

```
23 void* thread_B() {
24     if (mprotect(ptr, PAGE_SIZE, PROT_READ | PROT_WRITE) != 0) {
25         printf("Error: failed to set memory protection\n");
26         exit(1);
27     }
28
29     // use the memory here
30     int_ptr = (int*)ptr;
31     *int_ptr = 12345;
32
33     // set the protection of the memory to read only
34     if (mprotect(ptr, PAGE_SIZE, PROT_READ) != 0) {
35         printf("Error: failed to set memory protection\n");
36         exit(1);
37     }
38 }
```

در قسمت های زیر نیز کدهای ریسه‌های C, D را بررسی می‌کنیم. اگر هر کدام از ورودی‌های C_read, D_read برابر ۱ باشند، در مود خواندن قرار می‌گیرد و مقدار آن ۱۰۰ بار چاپ می‌شود در غیر این صورت در آن فضا نوشته می‌شود.

```

40 void* thread_C() {
41     if (C_read)
42         for (int i; i < 100; i++) {
43             printf("C : %d", *int_ptr);
44         }
45     else {
46         for (int i; i < 100; i++) {
47             *int_ptr = i;
48         }
49     }
50 }
51
52 void* thread_D() {
53     if (D_read)
54         for (int i; i < 100; i++) {
55             printf("D : %d", *int_ptr);
56         }
57     else {
58         for (int i; i < 100; i++) {
59             *int_ptr = i;
60         }
61     }
62 }

```

در نهایت کد اسکریپت اصلی :

```

gcc ./protect.c -o myprogram -pg
sudo perf stat -e faults,page-faults,dTLB-loads,dTLB-load-misses,iTLB-loads,iTLB-load-misses,context-switches ./myprogram
sleep 1
sudo perf stat -e faults,page-faults,dTLB-loads,dTLB-load-misses,iTLB-loads,iTLB-load-misses,context-switches ./myprogram D_write
sleep 1
sudo perf stat -e faults,page-faults,dTLB-loads,dTLB-load-misses,iTLB-loads,iTLB-load-misses,context-switches ./myprogram C_write D_write

```

بعد از اجرا :

در شکل صفحه بعد هر سه برنامه با پارامترهای خواسته شده اجرا شده‌اند.

همانطور که مشاهده می‌شود بیشترین تعداد TLB miss در هنگام نوشتن است و نیز کمترین درصد TLB miss نیز هنگام خواندن است.

کمترین تعداد page fault نیز هنگام خواندن است.

Segmentation Fault هنگامی که عملیات نوشتن نیز داشته باشیم برنامه دچار

می‌شود که منطقی است چرا که از آن قسمت **محافظت شده** و حق نوشتن در آن

قسمت را نداریم !!!

[illegible]

عکس بزرگ‌تر در صفحه بعد

Performance counter stats for './myprogram':

76	faults		
76	page-faults		
466,151	dTLB-loads		
1,230	dTLB-load-misses	#	0.26% of all dTLB cache accesses
472	iTLB-loads		
649	iTLB-load-misses	#	137.50% of all iTLB cache accesses
19	context-switches		

2.005488630 seconds time elapsed

0.000000000 seconds user

0.004309000 seconds sys

this is C_write : 1, and D_write : 0

./myprogram: Segmentation fault

Performance counter stats for './myprogram D_write':

78	faults		
78	page-faults		
452,909	dTLB-loads		
1,236	dTLB-load-misses	#	0.27% of all dTLB cache accesses
371	iTLB-loads		
613	iTLB-load-misses	#	165.23% of all iTLB cache accesses
9	context-switches		

2.071054901 seconds time elapsed

0.002386000 seconds user

0.000000000 seconds sys

this is C_write : 0, and D_write : 0

./myprogram: Segmentation fault

Performance counter stats for './myprogram C_write D_write':

78	faults		
78	page-faults		
450,557	dTLB-loads		
2,001	dTLB-load-misses	#	0.44% of all dTLB cache accesses
410	iTLB-loads		
590	iTLB-load-misses	#	143.90% of all iTLB cache accesses
9	context-switches		

2.068281328 seconds time elapsed

0.000000000 seconds user

0.002509000 seconds sys

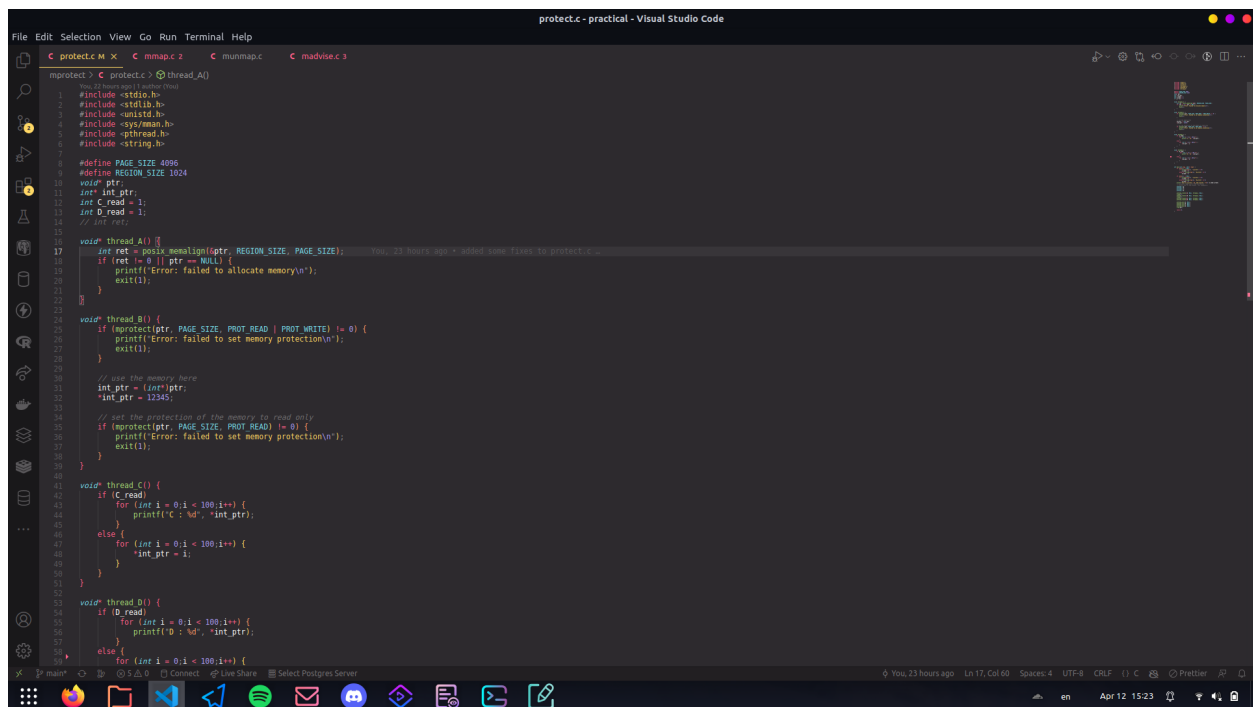
حال اندازه صفحه را به ۱ کیلوبایت تغییر می‌دهیم و دوباره تست‌ها را انجام می‌دهیم.

در صورت نیاز به دیدن قسمت REGION SIZE را تغییر دهید.

```
8 #define PAGE_SIZE 4096
9 #define REGION_SIZE 1024
```

حال در صورتی که اندازه سایز محافظت شده ۱ کیلوبایت باشد طبق عکس زیر به دلیل این که حتما mprotect از کل یک صفحه محافظت می‌کند و نمی‌تواند فقط از قسمتی از صفحه محافظت کند، هیچ محافظی صورت نمی‌گیرد، طبق این [منبع](#) نیز محافظت فقط از تعدادی صفحه انجام می‌گیرد نه قسمتی از یک صفحه. (تصویر بزرگ‌تر در صفحه بعد)

با این حال مشاهده می‌شود که کمترین درصد TLB miss و page fault در هنگام خواندن است.



```
File Edit Selection View Go Run Terminal Help
protect.c - practical - Visual Studio Code

protect.c M X C mmap.c 2 munmap.c madvise.c 3

mprotect > C protect.c > thread A()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/mman.h>
5 #include <pthread.h>
6 #include <string.h>
7
8 #define PAGE_SIZE 4096
9 #define REGION_SIZE 1024
10
11 void* ptr;
12 int C_read = 1;
13 int D_read = 1;
14
15 // thread A
16 void* thread A() {
17     int ret = posix_memalign(&ptr, REGION_SIZE, PAGE_SIZE);
18     if (ret != 0 || ptr == NULL) {
19         printf("Error: failed to allocate memory\n");
20         exit(1);
21     }
22 }
23
24 void* thread B() {
25     if (mprotect(ptr, PAGE_SIZE, PROT_READ | PROT_WRITE) != 0) {
26         printf("Error: failed to set memory protection\n");
27         exit(1);
28     }
29
30     // use the memory here
31     int_ptr = (int*)ptr;
32     *int_ptr = 12345;
33
34     // set the protection of the memory to read only
35     if (mprotect(ptr, PAGE_SIZE, PROT_READ) != 0) {
36         printf("Error: failed to set memory protection\n");
37         exit(1);
38     }
39 }
40
41 void* thread C() {
42     if (C_read) {
43         for (int i = 0; i < 100; i++) {
44             printf("C: %d\n", *int_ptr);
45         }
46     } else {
47         for (int i = 0; i < 100; i++) {
48             *int_ptr = i;
49         }
50     }
51 }
52
53 void* thread D() {
54     if (D_read) {
55         for (int i = 0; i < 100; i++) {
56             printf("D: %d\n", *int_ptr);
57         }
58     } else {
59         for (int i = 0; i < 100; i++) {
60             *int_ptr = i;
61         }
62     }
63 }
64
65 int main() {
66     pthread_t tA, tB, tC, tD;
67     pthread_create(&tA, NULL, thread A, NULL);
68     pthread_create(&tB, NULL, thread B, NULL);
69     pthread_create(&tC, NULL, thread C, NULL);
70     pthread_create(&tD, NULL, thread D, NULL);
71     pthread_join(tA, NULL);
72     pthread_join(tB, NULL);
73     pthread_join(tC, NULL);
74     pthread_join(tD, NULL);
75     return 0;
76 }
```

```

Hooman_Keshvari_99105667@bigwhoman-pp /m/b/N/S/t/O/A/H/p/mprotect> ./perf1.sh
[sudo] password for Hooman_Keshvari_99105667:
this is C_write : 1, and D_write : 1
Error: failed to set memory protection

Performance counter stats for './myprogram':

   Document 72      faults
           72      page-faults
   Doc 361,875      dTLB-loads
           1,015      dTLB-load-misses          #    0.28% of all dTLB cache accesses
   Music  378      iTLB-loads
           508      iTLB-load-misses          #  134.39% of all iTLB cache accesses
   Pictures 13      context-switches

   1.003479340 seconds time elapsed

   0.002191000 seconds user
   0.000000000 seconds sys

hooman_keshvari@bigwhoman-pp:~/m/b/N/S/t/O/A/H/p/mprotect$ ./perf1.sh
this is C_write : 1, and D_write : 0
Error: failed to set memory protection

Performance counter stats for './myprogram D_write':

           72      faults
           72      page-faults
   365,097      dTLB-loads
           1,131      dTLB-load-misses          #    0.31% of all dTLB cache accesses
           355      iTLB-loads
           409      iTLB-load-misses          #  115.21% of all iTLB cache accesses
           14      context-switches

   1.003655146 seconds time elapsed

   0.000000000 seconds user
   0.002015000 seconds sys

hooman_keshvari@bigwhoman-pp:~/m/b/N/S/t/O/A/H/p/mprotect$ ./perf1.sh
this is C_write : 0, and D_write : 0
Error: failed to set memory protection

Performance counter stats for './myprogram C_write D_write':

           73      faults
           73      page-faults
   360,236      dTLB-loads
           1,127      dTLB-load-misses          #    0.31% of all dTLB cache accesses
           356      iTLB-loads
           436      iTLB-load-misses          #  122.47% of all iTLB cache accesses
           14      context-switches

   1.003333057 seconds time elapsed

   0.002030000 seconds user
   0.000000000 seconds sys

```

سوال 2.1 .

تردهای A , B را بررسی می‌کنیم، ابتدا از فضای dev/zero/ مقداری به exchanged مپ می‌کنیم به این معنی که انگار داریم یک قسمت حافظه را به آن تخصیص می‌دهیم (مشابه malloc).

سپس در ترد B این قسمت از حافظه به یک قسمت ناشناخته دیگر مپ می‌شود.

```
21 void* thread_A() {  
22     if ((fd = open("/dev/zero", O_RDWR, 0)) == -1)  
23         err(1, "open");  
24     exchanged = (char*)mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_FILE | MAP_SHARED, fd, 0);  
25 }  
26  
27 void* thread_B() {  
28     char* exchanged = (char*)mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_ANON | MAP_SHARED, -1, 0);  
29 }  
30
```

در شکل صفحه بعد نتایج برنامه مشخص است.

تعداد پیچ فالت ها به ترتیب از زیاد به کم : خواندن و نوشتن - نوشتن - خواندن

درصد TLB miss به ترتیب از زیاد به کم : خواندن - نوشتن - خواندن و نوشتن

زمان صرف شده از زیاد به کم : خواندن - خواندن و نوشتن - نوشتن

با تقریب خوبی برنامه دوم بهترین عملکرد را داشته

تصویر کل، تصویر بزرگتر در صفحه بعدی آمده است.

```
fish /media/bigwhoman/New Volume/Sharif/term6/OS_Advanced/Advanced_Operating_Systems_Spring_2023/HW1/practical/mmap
Hooman_Keshvari_99105667@bigwhoman-pp /w/b/N/S/t/O/A/H/p/mmap> ./perf_mmap.sh
this is C_write : 1, and D_write : 1

Performance counter stats for './mmap':
 73 faults
 73 page-faults
430,497 dTLB-loads
1,111 dTLB-load-misses # 0.26% of all dTLB cache accesses
452 iTLB-loads
595 iTLB-load-misses # 131.64% of all iTLB cache accesses
16 context-switches

1.002861241 seconds time elapsed
0.000000000 seconds user
0.001778000 seconds sys

this is C_write : 1, and D_write : 0

Performance counter stats for './mmap D_write':
 75 faults
 75 page-faults
410,749 dTLB-loads
1,023 dTLB-load-misses # 0.24% of all dTLB cache accesses
460 iTLB-loads
588 iTLB-load-misses # 127.83% of all iTLB cache accesses
15 context-switches

1.002534235 seconds time elapsed
0.000000000 seconds user
0.001644000 seconds sys

this is C_write : 0, and D_write : 0

Performance counter stats for './mmap C_write D_write':
 72 faults
 72 page-faults
404,405 dTLB-loads
1,028 dTLB-load-misses # 0.25% of all dTLB cache accesses
457 iTLB-loads
581 iTLB-load-misses # 127.19% of all iTLB cache accesses
16 context-switches

1.002394989 seconds time elapsed
0.000000000 seconds user
0.001415000 seconds sys

Hooman_Keshvari_99105667@bigwhoman-pp /w/b/N/S/t/O/A/H/p/mmap>
```

```
Hooman_Keshvari_99105667@bigwhoman-pp /m/b/N/S/t/O/A/H/p/mmap> ./perf_mmap.sh
this is C_write : 1, and D_write : 1
```

```
Performance counter stats for './mmap':
```

File Edit View Insert Format Tools Extensions Help			
73	faults		
73	page-faults		
430,497	dTLB-loads	Normal text	Vazir... 14 + B I U
1,111	dTLB-load-misses	#	0.26% of all dTLB cache accesses
452	iTLB-loads		
595	iTLB-load-misses	#	131.64% of all iTLB cache accesses
16	context-switches		

```
1.002861241 seconds time elapsed
```

```
0.000000000 seconds user
```

```
0.001778000 seconds sys
```

```
this is C_write : 1, and D_write : 0
```

```
Performance counter stats for './mmap D_write':
```

75	faults			this is C_write
75	page-faults			Performance c
418,748	dTLB-loads			
1,023	dTLB-load-misses	#	0.24% of all dTLB cache accesses	
460	iTLB-loads			431
588	iTLB-load-misses	#	127.83% of all iTLB cache accesses	1
15	context-switches			

```
1.002534235 seconds time elapsed
```

```
0.000000000 seconds user
```

```
0.001644000 seconds sys
```

```
this is C_write : 0, and D_write : 0
```

```
Performance counter stats for './mmap C_write D_write':
```

72	faults			this is C_write
72	page-faults			Performance c
404,405	dTLB-loads			429
1,028	dTLB-load-misses	#	0.25% of all dTLB cache accesses	1
457	iTLB-loads			
581	iTLB-load-misses	#	127.13% of all iTLB cache accesses	2.00333
16	context-switches			

```
1.002394980 seconds time elapsed
```

```
0.000000000 seconds user
```

```
0.001415000 seconds sys
```

```
this is C_write
```

```
Performance c
```

حال برنامه را عوض کرده و در قسمت B از munmap استفاده می‌کنیم تا که قسمت مپ شده آنمپ کند و آن را به دیسک برگرداند.

```
21 void* thread_A() {
22     if ((fd = open("/dev/zero", O_RDWR, 0)) == -1)
23         err(1, "open");
24     exchanged = (char*)mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_FILE | MAP_SHARED, fd, 0);
25 }
26
27 void* thread_B() {
28     munmap(exchanged, 4096);
29 }
```

حال صفحه بعد را بررسی می‌کنیم.

در برنامه زیر همانطور که مشاهده می‌شود، در برنامه‌ای که فقط خواندن دارد بیشترین تعداد لود و کمترین میس در dTLB را داریم. زمان طی شده اما از دو برنامه دیگر بیشتر است. به صورت عجیبی پیچ فالت در تمامی آنها یکسان است احتمالاً به این دلیل که یک قطعه از حافظه را چون کامل مپ می‌کند و در همان نقطه می‌نویسد و از همان نقطه می‌خواند، پیچ فالت کمی داریم.

همچنین در هنگام نوشتن و خواندن بیشترین درصد TLB miss را شاهد هستیم.

```
Fish /media/bigwhoman/New Volume/Sharif/terms/OS_Advanced/Advanced_Operating_Systems_Spring_2023/HW1/practical/mmap
bigwhoman_Keshvari_99105667@bigwhoman-pp /w/b/N/S/L/O/A/H/p/mmap> ./perf_munmap.sh
this is C_write : 1, and D_write : 1

Performance counter stats for './munmap':
 73 faults
 73 page-faults
 423,631 dTLB-load
 979 dTLB-load-misses # 0.23% of all dTLB cache accesses
 433 iTLB-load
 580 iTLB-load-misses # 133.05% of all iTLB cache accesses
 16 context-switches

1.003191906 seconds time elapsed
0.001854000 seconds user
0.000000000 seconds sys

this is C_write : 1, and D_write : 0

Performance counter stats for './munmap D_write':
 73 faults
 73 page-faults
 414,289 dTLB-load
 1,054 dTLB-load-misses # 0.25% of all dTLB cache accesses
 444 iTLB-load
 612 iTLB-load-misses # 137.64% of all iTLB cache accesses
 16 context-switches

1.002913768 seconds time elapsed
0.001619000 seconds user
0.000000000 seconds sys

this is C_write : 0, and D_write : 0

Performance counter stats for './munmap C_write D_write':
 73 faults
 73 page-faults
 396,653 dTLB-load
 1,030 dTLB-load-misses # 0.26% of all dTLB cache accesses
 430 iTLB-load
 598 iTLB-load-misses # 139.07% of all iTLB cache accesses
 16 context-switches

1.002878598 seconds time elapsed
0.001673000 seconds user
0.000000000 seconds sys

bigwhoman_Keshvari_99105667@bigwhoman-pp /w/b/N/S/L/O/A/H/p/mmap>
```

تصویر بزرگ‌تر در صفحه بعدی.

this is C_write : 1, and D_write : 1

Performance counter stats for './munmap':

```
File Edit View Insert Format Tools Extensions Help
73 faults
73 page-faults
423,631 dTLB-loads
979 dTLB-load-misses # 0.23% of all dTLB cache accesses
433 iTLB-loads
580 iTLB-load-misses # 133.95% of all iTLB cache accesses
16 context-switches

1.003191906 seconds time elapsed

0.001854000 seconds user
0.000000000 seconds sys
```

this is C_write : 1, and D_write : 0

Performance counter stats for './munmap D_write':

```
73 faults
73 page-faults
414,289 dTLB-loads
1,054 dTLB-load-misses # 0.25% of all dTLB cache accesses
444 iTLB-loads
612 iTLB-load-misses # 137.84% of all iTLB cache accesses
16 context-switches

1.002913768 seconds time elapsed

0.001619000 seconds user
0.000000000 seconds sys
```

this is C_write : 0, and D_write : 0

Performance counter stats for './munmap C_write D_write':

```
73 faults
73 page-faults
396,653 dTLB-loads
1,030 dTLB-load-misses # 0.26% of all dTLB cache accesses
430 iTLB-loads
598 iTLB-load-misses # 139.07% of all iTLB cache accesses
16 context-switches

1.002878598 seconds time elapsed

0.001673000 seconds user
0.000000000 seconds sys
```

سوال 3.1 .

ابتدا قطعه کد نوشته شده را بررسی می‌کنیم.

```
48 int main(int argc, char** argv) {
49     int number_of_threads = 1;
50
51     for (int i = 0; i < argc; i++) {
52         if (strcmp(argv[i], "--number_of_threads") == 0 && i < argc - 1) {
53             // Extract the number of threads from the next argument
54             number_of_threads = atoi(argv[i + 1]);
55             printf("number of threads : %d\n", number_of_threads); // Output: the number of threads
56             if (number_of_threads > get_nprocs_conf()) {
57                 printf("You have chosen more threads than what you have \n exiting ...\n");
58                 exit(EXIT_FAILURE);
59             }
60             break;
61         }
62     }
63     if (pthread_barrier_init(&barrier, NULL, number_of_threads) != 0) {
64         perror("pthread_barrier_init failed");
65         exit(EXIT_FAILURE);
66     }
67     pthread_t threads[number_of_threads];
68     for (int i = 0; i < number_of_threads; i++) {
69         if (pthread_create(&threads[i], NULL, thread_function, NULL) != 0) {
70             perror("pthread_create failed");
71             exit(EXIT_FAILURE);
72         }
73     }
74     if (pthread_join(threads[0], NULL) != 0) {
75         perror("pthread_join failed");
76         exit(EXIT_FAILURE);
77     }
78     pthread_barrier_destroy(&barrier);
79     return (EXIT_SUCCESS);
80 }
```

در ابتدای برنامه، ورودی به صورت آرگومان ورودی بررسی می‌شود و `number of threads` آن مشخص کننده تعداد ریشه‌های برنامه ما است.

سپس مطمئن می‌شویم تعداد ریشه‌های گفته شده از تعداد پردازنده‌های ما بیشتر نباشد (خط ۵۵)

سپس تردها را می‌سازیم که در قطعه کد زیر آنها را بررسی می‌کنیم.

```

18 void* thread_function() {
19     for (int i = 0; i < NUMBER_OF_ITERATIONS; i++) {
20         void* ptr;
21         size_t size = 2 * PAGE_SIZE; // allocate two pages of memory
22         int ret;
23
24         // Allocate memory using mmap()
25         ptr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANON, -1, 0);
26         if (ptr == MAP_FAILED) {
27             perror("mmap failed");
28             exit(EXIT_FAILURE);
29         }
30
31         // Use madvise() to advise the kernel about memory usage
32         ret = madvise(ptr, size, MADV_SEQUENTIAL);
33         if (ret == -1) {
34             perror("madvise failed");
35             exit(EXIT_FAILURE);
36         }
37
38         // Deallocate the memory using munmap()
39         if (munmap(ptr, size) == -1) {
40             perror("munmap failed");
41             exit(EXIT_FAILURE);
42         }
43     }
44     pthread_barrier_wait(&barrier);
45 }

```

در این قطعه کد، در هر لوپ برنامه، با mmap یک فضا اختصاص داده می‌شود و سپس از madvise استفاده می‌شود تا به کرنل درباره استفاده از حافظه اصلی توصیه بدهد. در نهایت این قسمت از حافظه داده شده به حافظه جانبی بازمی‌گردد.

در خط ۴۴ هم همگی تردها منتظر می‌مانند تا کارها به اتمام برسد.

حال اسکریپت زیر را برای بررسی برنامه می‌نویسیم.

```

madvise > madvise_perf.sh
1 gcc ./madvise.c -o madvise -pg
2 for VAR in {1..10}
3 do
4     sudo perf stat -e cache-references,cache-misses,faults,page-faults,dTLB-load-misses,iTLB-load-misses,context-switches ./madvise --number_of_threads $VAR
5     sleep 1
6 done

```

و در نهایت نیز اسکریپت را اجرا می‌کنیم.

```
fish /media/bigwhoman/New Volume/Sharif/term6/OS_Advanced/Advanced_Operating_Systems_Spring_2023/HW1/practical/madvise

Hooman_Keshvari_99105667@bigwhoman-pp /m/b/N/S/t/G/A/H/p/madvise> bash ./madvise_perf.sh
[sudo] password for Hooman_Keshvari_99105667:
number of threads : 1

Performance counter stats for './madvise --number_of_threads 1':

    113,167      cache-references          # 74.655 % of all cache refs      (65.11%)
    84,485      cache-misses           # 74.655 % of all cache refs      (65.34%)
        70      faults
        70      page-faults
    206,095,841  dTLB-load-misses        # 0.00% of all dTLB cache accesses (68.37%)
    2,886      dTLB-load-misses        # 0.00% of all dTLB cache accesses (68.37%)
    630,974      dTLB-load-misses        # 38240.03% of all dTLB cache accesses (69.14%)
        25      dTLB-load-misses
        25      context-switches

    0.094409846 seconds time elapsed

    0.008129000 seconds user
    0.085356000 seconds sys

number of threads : 2

Performance counter stats for './madvise --number_of_threads 2':

    91,416,067  cache-references          # 0.098 % of all cache refs      (65.40%)
    89,688      cache-misses           # 0.098 % of all cache refs      (67.12%)
        74      faults
        74      page-faults
    525,255,101  dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.56%)
    6,308      dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.54%)
    10,163      dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.49%)
    1,632,442  dTLB-load-misses        # 10002.00% of all dTLB cache accesses (66.24%)
        36      dTLB-load-misses
        36      context-switches

    0.298191793 seconds time elapsed

    0.012815000 seconds user
    0.580749000 seconds sys

number of threads : 3

Performance counter stats for './madvise --number_of_threads 3':

    188,111,094  cache-references          # 0.086 % of all cache refs      (66.33%)
    162,315      cache-misses           # 0.086 % of all cache refs      (66.44%)
        75      faults
        75      page-faults
    883,346,409  dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.83%)
    13,993      dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.25%)
```

```
fish /media/bigwhoman/New Volume/Sharif/term6/OS_Advanced/Advanced_Operating_Systems_Spring_2023/HW1/practical/madvise

number of threads : 4

Performance counter stats for './madvise --number_of_threads 4':

    272,243,266  cache-references          # 0.061 % of all cache refs      (66.38%)
    166,229      cache-misses           # 0.061 % of all cache refs      (66.31%)
        77      faults
        77      page-faults
    1,150,570,901  dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.04%)
    20,208      dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.04%)
    20,207      dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.04%)
    3,691,304      dTLB-load-misses        # 10000.00% of all dTLB cache accesses (66.06%)
    1,713      dTLB-load-misses
    1,713      context-switches

    0.507177622 seconds time elapsed

    0.007060000 seconds user
    2.0010000 seconds sys

number of threads : 5

Performance counter stats for './madvise --number_of_threads 5':

    349,424,115  cache-references          # 0.051 % of all cache refs      (66.42%)
    179,393      cache-misses           # 0.051 % of all cache refs      (66.33%)
        77      faults
        77      page-faults
    1,580,845,816  dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.04%)
    20,208      dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.04%)
    20,207      dTLB-load-misses        # 0.00% of all dTLB cache accesses (67.04%)
    4,454,454      dTLB-load-misses        # 10000.00% of all dTLB cache accesses (66.70%)
    2,613      dTLB-load-misses
    2,613      context-switches

    0.861457202 seconds time elapsed

    0.012000000 seconds user
    4.1700000 seconds sys

number of threads : 6

Performance counter stats for './madvise --number_of_threads 6':

    489,911,436  cache-references          # 0.039 % of all cache refs      (66.53%)
    161,165      cache-misses           # 0.039 % of all cache refs      (66.50%)
        80      faults
        80      page-faults
    2,824,239,189  dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    20,208      dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    20,207      dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    5,292,570      dTLB-load-misses        # 10000.00% of all dTLB cache accesses (66.53%)
    2,440      dTLB-load-misses
    2,440      context-switches

    1.888873192 seconds time elapsed

    0.017000000 seconds user
    6.2527000 seconds sys

number of threads : 7

Performance counter stats for './madvise --number_of_threads 7':

    489,587,000  cache-references          # 0.039 % of all cache refs      (66.60%)
    161,165      cache-misses           # 0.039 % of all cache refs      (66.54%)
        80      faults
        80      page-faults
    2,824,070,803  dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    20,208      dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    20,207      dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    5,292,570      dTLB-load-misses        # 10000.00% of all dTLB cache accesses (66.53%)
    2,440      dTLB-load-misses
    2,440      context-switches

    1.377042498 seconds time elapsed

    0.002700000 seconds user
    9.3077000 seconds sys

number of threads : 8

Performance counter stats for './madvise --number_of_threads 8':

    489,587,788  cache-references          # 0.039 % of all cache refs      (66.60%)
    161,165      cache-misses           # 0.039 % of all cache refs      (66.54%)
        80      faults
        80      page-faults
    2,824,070,803  dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    20,208      dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    20,207      dTLB-load-misses        # 0.00% of all dTLB cache accesses (66.82%)
    5,292,570      dTLB-load-misses        # 10000.00% of all dTLB cache accesses (66.53%)
    2,440      dTLB-load-misses
    2,440      context-switches
```

```
fish /media/bigwhoman/New Volume/Sharif/terms/OS_Advanced/Advanced_Operating_Systems_Spring_2023/HW1/practical/madvise

8,845 context-switches
1.687658883 seconds time elapsed
8.871838888 seconds user
17.491318888 seconds sys

Number of threads : 8
Performance counter stats for './madvise --number_of_threads 8':
456,524,111 cache-references (66.82%)
315,782 cache-misses # 0.069 % of all cache refs (66.80%)
92 faults (66.82%)
92 page-faults (66.82%)
3,709,004,285 DTLB-load (66.82%)
28,176 DTLB-load-misses # 0.00% of all DTLB cache accesses (66.79%)
30,347 ITLB-load (66.82%)
4,281,527 ITLB-load-misses # 139.0 % of all ITLB cache accesses (66.78%)
8,275 context-switches

1.648838888 seconds time elapsed
8.887688888 seconds user
16.186218888 seconds sys

Number of threads : 10
Performance counter stats for './madvise --number_of_threads 10':
478,889,888 cache-references (66.80%)
389,388 cache-misses # 0.082 % of all cache refs (66.87%)
92 faults (66.87%)
92 page-faults (66.82%)
4,613,354,931 DTLB-load (66.82%)
52,228 DTLB-load-misses # 0.00% of all DTLB cache accesses (66.78%)
32,383 ITLB-load (66.79%)
6,721,800 ITLB-load-misses # 209.6 % of all ITLB cache accesses (66.78%)
29,942 context-switches

2.011372500 seconds time elapsed
8.187888888 seconds user
21.323918888 seconds sys

Number of threads : 11
Performance counter stats for './madvise --number_of_threads 11':
529,514,889 cache-references (66.80%)
418,748 cache-misses # 0.078 % of all cache refs (66.84%)
92 faults (66.84%)
92 page-faults (66.82%)
5,492,286,742 DTLB-load (66.82%)
58,528 DTLB-load-misses # 0.00% of all DTLB cache accesses (66.81%)
34,842 ITLB-load (66.80%)
7,196,888 ITLB-load-misses # 209.1 % of all ITLB cache accesses (66.79%)
40,849 context-switches

2.018881250 seconds time elapsed
8.118478888 seconds user
26.383278888 seconds sys

Number of threads : 12
Performance counter stats for './madvise --number_of_threads 12':
456,239,844 cache-references (66.80%)
448,374 cache-misses # 0.098 % of all cache refs (66.89%)
162 faults (66.89%)
162 page-faults (66.82%)
5,267,141,788 DTLB-load (66.82%)
163,888 DTLB-load-misses # 0.00% of all DTLB cache accesses (66.81%)
30,416 ITLB-load (66.80%)
7,385,340 ITLB-load-misses # 242.6 % of all ITLB cache accesses (66.87%)
131,377 context-switches

2.893208417 seconds time elapsed
8.118118888 seconds user
38.853358888 seconds sys

hosman_Keshvari_99195667@bigwhoman-pp /n/b/N/S/E/O/A/N/p/madvise>
```

حال که نتایج را نشان دادیم کمی اسکرپت را عوض می‌کنیم تا بتوانیم خروجی‌ها را در فایل بریزیم و آنها را تحلیل کنیم.

```
fish /media/bigwhoman/New Volume/Sharif/terms/OS_Advanced/Advanced_Operating_Systems_Spring_2023/HW1/practical/madvise

hosman_Keshvari_99195667@bigwhoman-pp /n/b/N/S/E/O/A/N/p/madvise> bash ./madvise_perf_capture_output.sh
hosman_Keshvari_99195667@bigwhoman-pp /n/b/N/S/E/O/A/N/p/madvise> cat ./madvise_perf_capture_output.sh
gcc ./madvise.c -o madvise -pg
for VAR in {1..12}
do
    sudo perf stat -e cache-references,cache-misses,faults,page-faults,DTLB-load,DTLB-load-misses,ITLB-load,ITLB-load-misses,context-switches ./madvise --number_of_threads $VAR && ./outputs/"thread_count_$VAR".txt
done
sleep 1
done
hosman_Keshvari_99195667@bigwhoman-pp /n/b/N/S/E/O/A/N/p/madvise>
```

حال که نتایج را نشان دادیم کمی اسکرپت را عوض می‌کنیم تا بتوانیم خروجی‌ها را در فایل بریزیم و آنها را تحلیل کنیم.

حال تمامی خروجی‌های ما در فولدر outputs ذخیره شدند که تحلیل‌ها را بر اساس این فایل‌ها انجام می‌دهیم.

حال با برنامه نوشته شده پایتون زیر، داده‌ها در قسمت outputs را ابتدا پارس می‌کنیم و سپس آنها را تحلیل می‌کنیم.

```
madvise > extract_and_draw.py > ...
1 import re
2 import matplotlib.pyplot as plt
3 # Prompt user for input file path
4
5 def draw(x_list, y_list, test_parameter, directory):
6     plt.plot(x_list, y_list)
7     plt.ylabel(test_parameter)
8     plt.xlabel('number of threads')
9     # plt.show()
10    plt.savefig(f'{directory}/{test_parameter}.png')
11    plt.clf()
12
13 tests = []
14 num_of_threads = [i for i in range(1,13)]
15 for i in range(1,13):
16     input_file = f'./outputs/thread_count_{i}.txt'
17     # Read input from file
18     with open(input_file, "r") as f:
19         input_str = f.read()
20
21     # Extract performance metrics using regular expressions
22     cache_misses = int(re.search(r'([\d,]+) "cache-misses"', input_str).group(1).replace(",", ""))
23     page_faults = int(re.search(r'([\d,]+) "page-faults"', input_str).group(1).replace(",", ""))
24     dtlb_load_misses = int(re.search(r'([\d,]+) "dtlb-load-misses"', input_str).group(1).replace(",", ""))
25     itlb_load_misses = int(re.search(r'([\d,]+) "itlb-load-misses"', input_str).group(1).replace(",", ""))
26     elapsed_time = float(re.search(r'([\d,]+) seconds time elapsed"', input_str).group(1))
27
28     metrics = {'cache_misses':cache_misses, 'page_faults':page_faults, 'dtlb_load_misses':dtlb_load_misses, 'itlb_load_misses':itlb_load_misses, 'elapsed_time':elapsed_time, 'number_of_threads':i}
29     tests.append(metrics)
30
31 cache_misses = [test['cache_misses'] for test in tests]
32 page_faults = [test['page_faults'] for test in tests]
33 dtlb_load_misses = [test['dtlb_load_misses'] for test in tests]
34 itlb_load_misses = [test['itlb_load_misses'] for test in tests]
35 elapsed_time = [test['elapsed_time'] for test in tests]
36
37 draw(num_of_threads, cache_misses, 'cache misses', './test_results')
38 draw(num_of_threads, page_faults, 'page faults', './test_results')
39 draw(num_of_threads, dtlb_load_misses, 'dtlb misses', './test_results')
40 draw(num_of_threads, itlb_load_misses, 'itlb misses', './test_results')
41 draw(num_of_threads, elapsed_time, 'elapsed time', './test_results')
42
```

این برنامه نتایج و عکس‌های تولید شده را در پوشه test_results ذخیره می‌کند.

نتایج را در صفحه بعد مشاهده می‌کنیم.

