Activity 8 Memory Management

วัตถุประสงค์

1. เพื่อให้นิสิตเข้าใจหลักการทำงานของ virtual memory
2. เพื่อให้นิสิตสามารถวัดประสิทธิภาพของโปรแกรมในด้านการใช้หน่วยความจำ
3. เพื่อให้นิสิตสามารถเขียนโปรแกรมที่ใช้หน่วยความจำและ I/O ได้อย่างมีประสิทธิภาพ

ความรู้เบื้องต้น

- ความต้องการใช้หน่วยความจำของทุกโปรเซสรวมกันอาจมีมากกว่าขนาดของ physical memory ระบบปฏิบัติการสามารถสร้างหน่วยความจำเสมือน (virtual memory) ที่มีขนาดใหญ่กว่า physical memory โดยใช้หน่วยความจำสำรอง คือ ฮาร์ดดิสก์ ในการเก็บ page ที่ยังไม่มีความต้องการใช้งาน และนำเข้าสู่ physical memory เมื่อมีการใช้งาน เรียกกลไกนี้ว่า demand paging
- เมื่อโปรเซสต้องการใช้หน่วยความจำ page ที่ยังไม่อยู่ใน physical memory จะเกิด exception เรียกว่า page fault ซึ่งจะทำให้ระบบปฏิบัติการเข้ามาจัดการดึงข้อมูลที่ต้องการมาจากฮาร์ดดิสก์ และอาจจะต้องเลือกบาง page ที่อยู่ใน physical memory ออกไปเก็บไว้ในฮาร์ดดิสก์แทน เรียกว่า page replacement การเกิด page fault จะมี overhead ในการทำงานสูง เนื่องจากการเรียกใช้บริการจากระบบปฏิบัติการและการใช้ฮาร์ดดิสก์
- Page fault มีสองแบบ แบบแรกเรียกว่า major page fault หรือ hard page fault เป็น page fault ที่มีการอ่านข้อมูลจากฮาร์ดดิสก์เข้ามาเก็บในหน่วยความจำด้วย ซึ่งอาจเกิดจากการที่ต้องการใช้ page ที่ถูก swapped out ไปก่อนหน้านี้ หรือมีการทำ memory mapped file แล้วมีอ่านไฟล์นั้น
- แบบที่สองเรียกว่า minor page fault หรือ soft page fault เป็น page fault ที่ไม่มีการใช้ฮาร์ดดิสก์ เป็นเพียงการ allocate หรือ map page ที่ต้องการใช้ที่มีอยู่ในหน่วยความจำอยู่แล้วให้กับโปรเซส
- Memory mapped I/O เป็นเทคนิคในการเพิ่มประสิทธิภาพในการเข้าถึงไฟล์หรืออุปกรณ์อื่นโดยการจัดสรรพื้นที่ในหน่วยความจำที่สัมพันธ์กับไฟล์ แล้วการอ่านเขียนหน่วยความจำบริเวณนั้นจะเท่ากับการอ่านเขียนไฟล์ ใน Linux สามารถทำได้โดยใช้ `mmap()`

เตรียมตัว

1. อ่านหนังสือ Operating System Concepts ของ Silberschatz หัวข้อ 8.5 Paging, 9.1 Virtual Memory Background 9.2 Demand Paging
2. ศึกษาวิธีใช้ฟังก์ชั่น `malloc(), free(), memset(), mmap()`
3. ศึกษาคำสั่ง ps ในส่วน Standard Format Specifiers (https://man7.org/linux/man-pages/man1/ps.1.html#STANDARD_FORMAT_SPECIFIERS) โดยเฉพาะความหมายของ pid, cmd, sz, min_flt, maj_flt, rssize, size, vsize, pmem

กิจกรรม

1. นำโปรแกรมต่อไปนี้ไปคอมไพล์เป็นโปรแกรมชื่อ memusage

```
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>


int main(void)
{
  void *small, *large, *map;
  FILE *fp;  // file pointer
  int fd;    // file descriptor
  int i;
  char x;

  int FILESIZE = (1024*1024);
  char *FILENAME = "mmap.dat";
  char *data = "hello";
  int len;

  printf("Program starts. Press Enter to continue.\n");
  getchar();

  small = malloc(1<<10);
  printf("Allocated memory 1 KB\n");
  getchar();

  memset(small, 1, (1<<10));
  printf("Access memory 1 KB\n");
  getchar();

  large = malloc(1<<20);
  printf("Allocated memory 1 MB\n");
  getchar();
```

```
  memset(large, 1, (1<<20));
  printf("Access memory 1 MB\n");
  getchar();


  // Create a file with specified size
  fp = fopen(FILENAME, "w+");
  if (fp) {
    fseek(fp, FILESIZE-1, SEEK_SET);
    fwrite("", 1, sizeof(char), fp);
    fclose(fp);
  }
  else { perror("fopen"); exit(-1); }
  printf("Create file\n");
  getchar();


  fd = open(FILENAME, O_RDWR);
  if (fd == -1) { perror("open"); exit(-1); }

  printf("Open file\n");
  getchar();

  // Memory map to file
  map = mmap(NULL, FILESIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
  if (map == MAP_FAILED) { perror("mmap"); exit(-1); }
  printf("Create memory map\n");
  getchar();

  // Write to memory map
  len = strlen(data);
  for (i=0; i<FILESIZE/len; i++) {
    strcpy((char*)map+(i*len), data);
  }
  printf("Access Memory map. You can see data in the file.\n");
  getchar();

  free(small);
  free(large);
  munmap(map, FILESIZE);
  printf("Free memory\n");
  getchar();

  printf("End\n");

}
```

- รันโปรแกรม memusage ใน terminal หนึ่ง โปรแกรมนี้จะทำงานเป็นขั้นๆ แต่ละขั้นจะรอให้ผู้ใช้กด Enter เพื่อทำงานต่อ
- รันคำสั่งต่อไปนี้ในอีก terminal หนึ่ง

```
ps -C memusage -o pid,cmd,sz,min_flt,maj_flt,rssize,size,vsize,pmem
```

หรือใช้ร่วมกับคำสั่ง `watch` เช่น

```
watch -n 1 ps -C memusage -o pid,cmd,sz,...
```

คำสั่งนี้ใช้เพื่อดูว่าตัวเลขต่างๆที่เกี่ยวข้องกับหน่วยความจำมีการเปลี่ยนแปลงอย่างไรในแต่ละขั้นตอนการทำงานของโปรแกรม memusage

- รันคำสั่งต่อไปนี้ในอีก terminal หนึ่ง โดยแทน <pid> ด้วย process id ของ memusage

```
cat /proc/<pid>/maps
```

หรือใช้ร่วมกับคำสั่ง `watch` เช่น `watch -n 1 cat /proc/<pid>/maps`
คำสั่งนี้ใช้เพื่อสังเกตความเปลี่ยนแปลงของผลลัพธ์ ว่ามีการเพิ่มหรือลดพื้นที่ในหน่วยความจำในช่วง address ใด (อาจเก็บผลลัพธ์ลงไฟล์แล้วใช้คำสั่ง diff ช่วยเปรียบเทียบ)

- กด Enter ที่ terminal ที่รันโปรแกรม memusage แล้วบันทึกผลลัพธ์ของคำสั่ง ps และ cat ในแต่ละขั้นการทำงานของโปรแกรม memusage ลงในตาราง
- เขียนอธิบายในแต่ละขั้นตอนการทำงานของ memusage ว่ามีค่าตัวแปรใดที่เปลี่ยนแปลงไปบ้าง และเกิดจากอะไร

	SZ	MINFL	MAJFL	RSS	SIZE	VSZ	%MEM
Program Start	623	74	0	588	312	2492	0.0
	What happens: มีการ allocate memory ในส่วน code และตัวแปรต่างๆ จึงมีค่า SZ, RSS, SIZE, VSZ > 0 MINFL เกิดขึ้นจากการที่มีการ include library ต่างๆ เข้ามาซึ่งอาจมีอยู่ใน memory แล้ว และไม่เกิด MAJFL						
Allocate Memory 1 K	623	74	0	588	312	2492	0.0
	Allocated address: 55eb1d056000-55eb1d05b000 Code & Data 55eb1e502000-55eb1e523000 [heap] 7f312ed46000- 7f312ef34000 Library (libc) 7f312ef42000- 7f312ef71000 Library (ld) 7ffc2bfd5000-7ffc2bff6000 [stack] 7ffc2bff6000-7ffc2bffa000 [vvar] 7ffc2bffa000-7ffc2bffb000 [vdso]						
	What happens: มีการ allocate พื้นที่ให้ตัวแปร small ซึ่งขนาดไม่เกิน heap จึงไม่มีการเปลี่ยนแปลงอะไร						
Access Memory 1 K	623	74	0	588	312	2492	0.0
	What happens: มีการ access ตัวแปร small ซึ่งพื้นที่ที่จองไว้ขนาดไม่เกิน heap จึงไม่มีการเปลี่ยนแปลงอะไร						
Allocate Memory 1 M	880	75	0	588	1340	3520	0.0
	Allocated address: 55eb1d056000-55eb1d05b000 Code & Data						

	55eb1e502000-55eb1e523000 [heap] 7f312ec45000-7f312ed46000 7f312ed46000- 7f312ef34000 Library (libc) 7f312ef42000- 7f312ef71000 Library (ld) 7ffc2bfd5000-7ffc2bff6000 [stack] 7ffc2bff6000-7ffc2bffa000 [vvar] 7ffc2bffa000-7ffc2bffb000 [vdso]						
	What happens: มีการ allocate พื้นที่ให้ตัวแปร large ซึ่งเกินพื้นที่ที่มีอยู่ VSZ และ SIZE จึงเพิ่มขึ้นตามที่ allocate คือราว ๆ 1M ส่วน SZ เพิ่มน้อยกว่า 1M เพราะเพิ่มแค่ให้เพียงพอต่อ paging นอกจากนี้ allocated address ยังมีมากขึ้น 4K ด้วย						
Access Memory 1 M	880	331	0	2224	1340	3520	0.0
	What happens: ค่าที่เพิ่มขึ้น คือ RSS กับ MINFL เป็นเพราะข้อมูล 1M ไม่อยู่ใน TLB เลย จึงเกิด page fault						
Allocate Memory Map	1136	334	0	2224	1340	4544	0.0
	Allocated address: 55eb1d056000-55eb1d05b000 Code & Data 55eb1e502000-55eb1e523000 [heap] 7f312eb45000-7f312ec45000 mmap.dat 7f312ec45000-7f312ed46000 7f312ed46000- 7f312ef34000 Library (libc) 7f312ef42000- 7f312ef71000 Library (ld) 7ffc2bfd5000-7ffc2bff6000 [stack] 7ffc2bff6000-7ffc2bffa000 [vvar] 7ffc2bffa000-7ffc2bffb000 [vdso]						
	What happens: VSZ กับ SZ เพิ่มขึ้น เนื่องจากมีการ allocate พื้นที่ให้ memory map ราว ๆ 1M SIZE ไม่เพิ่มเนื่องจากสิ่งที่เกิดขึ้นเป็นไฟล์ใน disk จึงไม่ต้อง swap ขึ้นมา						
Access Memory Map	1136	334	256	3236	1340	4544	0.0
	What happens: เกิด MAJFL และมี RSS เพิ่งขึ้น เนื่องจากต้องไปเอา memory map file ใน disk						

Free	623	334	256	1316	312	2492	0.0
	What happens: ทุกค่ากลับมาเท่าตอน Program Start ยกเว้น Page Fault ซึ่งเป็นค่าที่ accumulate ขึ้น (ไม่ได้เพิ่มขึ้น) และ RSS						