Midterm-VerB

● Graded

Student

PO-YUN) 鄭博允 (CHENG

Total Points

80 / 100 pts

Question 1

1(a) **2** / 2 pts

✔ **– 0 pts** correct

Question 2

1(b) **2** / 2 pts

✔ **– 0 pts** 正確 (key point: 單一整合、無法部分更新)

Question 3

1(c) **2** / 2 pts

✔ **– 0 pts** Correct answer

Question 4

1(d) **2** / 2 pts

✔ **– 0 pts** Correct answer.
Note that the interrupt cannot be blocked, delayed, ignored, and requires immediate handling.

Question 5

2(a) **4** / 4 pts

✔ **+ 4 pts** Correct

Question 6

3(a) **3** / 8 pts

- ✔ **– 2 pts** Miss the terminology (working-set model)
- ✔ **– 2 pts** Wrong or empty (working-set model)
- ✔ **– 1 pt** Incomplete description (page-fault frequency)

Question 7

3(b) **0** / 4 pts

- ✔ **– 2 pts** async wrong
- ✔ **– 2 pts** sync wrong

Question 8

3(c) **2** / 4 pts

- ✔ **– 2 pts** (2) wrong

Question 9

3(d) **4** / 4 pts

✔ **– 0 pts** correct

Question 10

3(e) **2** / 2 pts

✔ **– 0 pts** 正確

Question 11

3(f) **2** / 2 pts

✔ **– 0 pts** Correct

Question 12

4(a)-2 **4** / 4 pts

✔ **– 0 pts** 正確

Question 13

4(b) **2** / 4 pts

✔ **– 2 pts** 錯1個

Question 14

4(a)-1 **4** / 4 pts

✔ **– 0 pts** 正確

Question 15

4(c) **4** / 4 pts

✔ **– 0 pts** Correct answer

Question 16

5(a) **8** / 8 pts

✔ **– 0 pts** correct answer

Question 17

5(b) **4** / 4 pts

✔ **– 0 pts** Correct

Question 18

5(c) **4** / 4 pts

✔ **– 0 pts** 正確

Question 19

6(a) **0** / 6 pts

✔ **– 6 pts** incorrect

Question 20

6(b) **9** / 10 pts

✔ **+ 5 pts** (i) Average time for handling page faults

✔ **+ 5 pts** (ii) Effective access time

✔ **– 1 pt** Minor error

1 p is a rate, shouldn't have a unit

Question 21

6(c) **2** / 2 pts

✔ **– 0 pts** correct

Question 22

6(d) **2** / 2 pts

✔ **– 0 pts** Correct

Question 23

7(a) **2** / 2 pts

✔ **– 0 pts** Correct Answer

Question 24

7(b) **2** / 2 pts

✔ **– 0 pts** Correct Answer

Question 25

7(c) **4** / 4 pts

✔ **– 0 pts** Correct Answer

Question 26

7(d) **4** / 4 pts

✔ **– 0 pts** Correct Answer

CSIE 3310, Spring 2024: Midterm
National Taiwan University

Name	鄭	博	允			SID	b	1	1	9	0	2	0	3	8

Instructions:

- Do NOT use the pencil to answer.
- Write your name and student ID in the above boxes. One character/digit in each box.
- Write your answers in the rectangle below each question or the specific areas.
- Provide sufficient explanations of your answer.
- Ask for permission to use the restroom, and leave your exam sheets and phone on the seat.
- Stay seated in the last 5 minutes before the exam ends: restroom and early turning in are NOT allowed.
- When the exam ends, stay seated until the TAs collect the exam sheets in your seating region.

Wait!!

Do NOT turn the page until you are told to start.

Ver. B

There are 7 questions on 14 pages. Raise your hand NOW if not.

1 Term Definitions (8 pts)

(a) (2 pts) What is a trap (or an exception) in operating systems?

Answer:

an interrupt caused by software , might occur when devided by 0 or other reasons.

(b) (2 pts) Monolithic Operating System

Answer:

由單一 kernel 負責處理所有操作的系統

(c) (2 pts) Real-time systems

Answer:

需要和用戶交互且有反應時間限制的系統

(d) (2 pts) Nonmaskable interrupt

Answer:

不能被 mask 蓋掉(擋住)的 interrupt, 也就是一定會被 process (thread) 給 handle

2 Calculation (4 pts)

(a) (4 pts) Amdahl's Law is a formula that identifies potential performance gains from adding additional computing cores to an application that has both serial (nonparallel) and parallel components. When an application has 75 percent parallel and 25 percent serial and executes on a system having three processing cores, what's the estimated speedup according to Amdahl's Law? (Please provide the process to compute.)

Answer:

$$S \leq \frac{1}{\frac{1}{4} + \frac{\frac{3}{4}}{3}} \Rightarrow S \leq 2$$

3 Short Answers (24 pts)

(a) (8 pts) Please briefly explain two solutions to prevent thrashing (4 pts for each answer). [Hint: A terminology and short description are all needed.]

Answer: thrashing 代表 CPU 花費太多時間在處理 page faults

(1) 在 process 的 page fault rate 太高的時候，釋放其它 process 的 pages 來支援，使得 page fault rate 降低

(2) 在 process 的 page fault rate 太低的時候，釋放多餘的 pages 來保持整體的 pages 使用率

(b) (4 pts) Explain the difference(s) between *asynchronous* and *synchronous* threading.

Answer:

asynchronous threading 代表不同 threads 輪流執行

synchronous threading 代表不同 threads 並行

(c) (4 pts) Explain at least two (out of three) methods to pass parameters to system calls from a user space to the kernel space.

Answer: 將 parameters

(1) 放在記憶體 (memory) 內，因此 user 可以寫入 memory，再由 kernel 去讀取

(2) 透過 process 本身的 heaps，一樣的概念

(d) (4 pts) Regarding how a processor gives commands and data to a controller, what is memory-mapped I/O? Compared with special (direct) I/O instructions, what is the main advantage of memory-mapped I/O?

Answer:

memory mapped I/O 代表 processor 直接對特定的 address 做 I/O，

(e) (2 pts) What is the purpose of separating the kernel model from the user mode?

Answer:

為了不讓 user 可以對 kernel 做任意的改動(操作)

(f) (2 pts) What is the purpose of systems calls?

Answer: 讓在 user mode 的 program 可以
執行在 kernel mode 下才能做的操作，像是 I/O。

4 Threads (16 pts)

The following program creates a thread pool in a process. Read the program shown below and answer the questions.

```
struct threadpool_t {
  pthread_mutex_t lock;
  pthread_cond_t notify;
  pthread_t *threads;
  threadpool_task_t *queue;
  int threadcount;
    /* skipped to save space */
};

static void *threadpool_thread(void *threadpool);
int threadpool_free(threadpool_t *pool);

threadpool_t *threadpool_create(int threadcount, int queue_size, int flags)
{
    if(threadcount <= 0 || threadcount > MAX_THREADS || queue_size <= 0 || queue_size > MAX_QUEUE) {
        return NULL; }

    threadpool_t *pool;
    int i;

    if((pool = (threadpool_t *)malloc(sizeof(threadpool_t))) == NULL) {
        goto err; }

    /* Initialize */
    pool->threadcount = 0;
    pool->queue_size = queue_size;
    pool->head = pool->tail = pool->count = 0;
    pool->shutdown = pool->started = 0;

    /* Allocate thread and task queue */
    pool->threads = (pthread_t *)malloc(sizeof(pthread_t) * threadcount);
    pool->queue = (threadpool_task_t *)malloc(sizeof(threadpool_task_t) * queue_size);

    /* Initialize mutex and conditional variable first */
    if((pthread_mutex_init(&(pool->lock), NULL) != 0 ) ||
       (pthread_cond_init(&(pool->notify), NULL) != 0 ) ||
       (pool->threads ) == NULL||
       (pool->queue == NULL) {
        goto err;
    }

    /* Start worker threads */
    for(i = 0; i < threadcount; i++) {
        if(pthread_create(&(pool->threads[i]), NULL,
                          threadpool_thread, (void*)pool) != 0) {
            threadpool_destroy(pool, 0);
            return NULL;
        }
        pool->threadcount++;
        pool->started++;
    }
    return pool;

err:
    if(pool) {
        threadpool_free(pool);
    }
    return NULL;
}
```

```
static void *threadpool_thread(void *threadpool)
{
    threadpool_t *pool = (threadpool_t *)threadpool;
    threadpool_task_t task;

    for(;;) {
        pthread_mutex_lock(&(pool->lock));

        while((pool->count == 0) && (!pool->shutdown)) {
            pthread_cond_wait(&(pool->notify), &(pool->lock));
        }

        if((pool->shutdown == immediate_shutdown) ||
           ((pool->shutdown == graceful_shutdown) &&
            (pool->count == 0))) {
            break;
        }

        /* Grab our task */
        task.function = pool->queue[pool->head].function;
        task.argument = pool->queue[pool->head].argument;
        pool->head += 1;
        pool->head = (pool->head == pool->queue_size) ? 0 : pool->head;
        pool->count -= 1;

        pthread_mutex_unlock(&(pool->lock));

        /* Get to work */
        (*(task.function))(task.argument);
    }

    pool->started--;

    pthread_mutex_unlock(&(pool->lock));
    pthread_exit(NULL);
    return(NULL);
}
```

Listing 1: Thread Pool

(a) (8 pts) Figure 1 shows memory layout for single-thread process on the left and multiple-thread process on the right. Please complete the memory layout on the right of Figure 1 (4 pts) and elaborate the changes (4 pts) after the threadpool is created.

Answer: (The first part should be answered on Figure 1. Don't redraw the figure here.)

threads 會共享 process 的 (1), (3), (4), (5), (6) 區塊，
當 thread 接到不同的 task，會使用各自的 stack 來
做處理。

(b) (4 pts) Please indicate which segment shown Figure 1 will variable `pool` defined at Line ~~17~~ 18 and variable `pool->queue` defined at Line ~~27~~ 33 be stored.

Answer:

pool : segment (4) uninitialized data

pool → queue : segment (3) heap

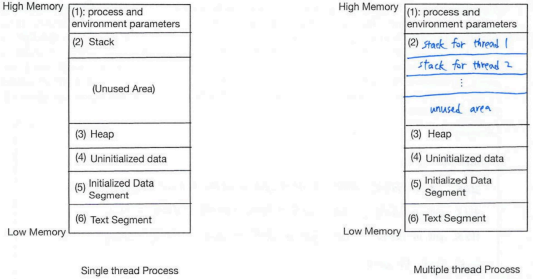

Figure 1: Memory Layout

(c) (4 pts) The threads created on Line 43-44 by function `pthread_create()` shown below are assigned to call function `threadpool_thread`, listed from Line 61-97, as the start routine.

```
1  #include <pthread.h>
2
3  int pthread_create( pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*
     start_routine)(void *), void *restrict arg);
```

(i) What does the created thread do immediately after creation?

(ii) Does the `for` loop between Line 66 and 90 keeps the thread execute forever? Yes or No, and elaborate your answer.

Answer:

(i) 執行 threadpool_thread()

(ii) No. 每當 thread 執行完工作且，會檢查 pool → shutdown 的狀態
一旦 pool → shutdown 被設定立即關閉，或是當前沒有工作而且為 graceful shutdown 時，就會跳出 loop.

5 Main Memory (16 pts)

(a) (8 pts) Assume that an initial list of available memory partitions (i.e., free holes) as follows: 120K → 550K → 250K → 400K → 650K. Four processes are requesting for memory size of 200K, 410K, 115K, and 500K (in order). Considering a variable-size partition scheme, please perform the memory allocation using the following algorithms: (1) first-fit algorithm and (2) best-fit

algorithm. For each algorithm, please indicate **(i) which memory partition is allocated to which request and (ii) the updated list of free available memory partitions.** If no partition can satisfy the request, please also indicate it. [Hint: for example, when the first-fit algorithm is applied, "*200K is put in 550K partition, and the updated list: 120K → 350K → 250K → 400K → 650K.*" is expected. If a request of xxx K needs to wait, "*xxxK must wait since no available free partition can be allocated.*" is expected].]. [1 point for each allocation in each algorithm. No partial points are given. Full points are given only when the answer is completely correct.]

Answer:

First-fit:

200K put in 550K, list: 120K → 350K → 250K → 400K → 650K

410K put in 650K, list: 120K → 350K → 250K → 400K → 240K

115K put in 120K, list: 5K → 350K → 250K → 400K → 240K

500K needs to wait

Best-fit:

200K put in 250K, list: 120K → 550K → 50K → 400K → 650K

410K put in 550K, list: 120K → 140K → 50K → 400K → 650K

115K put in 120K, list: 5K → 140K → 50K → 400K → 650K

500K put in 650K, list: 5K → 140K → 50K → 400K → 150K

(b) (4 pts) What type of fragmentation is present in the above cases of memory allocation?

Answer:

external fragmentation

(c) (4 pts) Considering a 16-bit system, a logical address contains 16 bits, and a physical address contains 16 bits. The starting address in the physical memory is 0000000000000000. A paging model is used in the system. The page size is 4KB ($=2^{12}$). Given the following page table of a process in Figure 2 and a logical address 0100000000100001 referenced by CPU,

(i) what is the page number of the process referenced by CPU, and

(ii) what is the frame number in the physical memory referenced by CPU?

Please briefly explain procedure of address translation in your answers. (2 pts for page number; 2 pts for frame number).

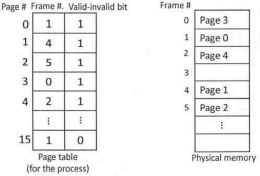

Figure 2: The page table vs. the physical memory.

Answer:

(i) first, the logical address should be split into 0|00, 0000001|0000|

which means the page number is

0|00 ⇒ 4

(ii) by the page table, page number 4 is refer to frame number 2

6 Virtual memory (20 pts)

(a) (6 pts) Assume that we have a paged memory system with special fast-lookup hardware cache called "associative memory" or "translation look-aside buffers (TLBs)". In this system, page tables are assumed in memory. The following Figure 3 shows the paging hardware with TLBs. The TLB is used to enhance the paging model as follows.

- When a logical address is generated by the CPU, the MMU first checks if its page number is present in the TLB.
- On a TLB ~~bit~~ hit, its frame number is used to access memory. In this case, page table will not be checked.
- On a TLB miss, a memory reference to the page table must be made. The corresponding frame number can be found in the page table to access physical memory. Assume that the hit ratio of the page table is 100% (i.e., no page fault needs to be handled).

Assume that memory access time is 750 ns. The TLBs' hit ratio is 80%. The time for searching for entries in TLBs is 50 ns. What is the effective memory access time? [No partial points are given. Full points are given only when the answer is completely correct.]

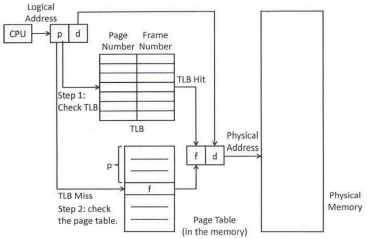

Figure 3: Paging with translation look-aside buffers (TLBs).

Answer:

time = (750 + 50) × 0.8 + (750 + 750) × 0.2

= 640 + 300

= 940 ns

(b) (10 pts) Assume that we have a demand paging virtual memory system. The page table is held in the memory. When a page fault happens, we consider the following paging in/out strategy:

(i) If there a free frame in the physical memory, page in the desired page.

(ii) If the selected victim page is not dirty, overwrite the victim page without paging out.

(iii) If the selected victim page is dirty, the victim page is written to the disk (i.e., paging out) and page in the desired page.

The page fault handling time takes 8,000 microseconds (μs) if either the case (b)(i) or the case (b)(ii) happens. If the case (b)(iii) happens, the page fault handling time takes 20,000 μs. Note that the case (b)(iii) takes longer time to handle a page fault due to a dirty page. The probability that the replaced page is dirty is 60%. The memory access time is 1 μs.

(i) (5 pts) What is the average time for handling page faults?

(ii) (5 pts) What is the maximum acceptable page fault rate (denoted by p) for an effective access time of no more than 4 μs?

[Hint: The effective access time = (1-p)× (memory-access time) + p × (page fault time), where p is the page fault rate. There is no need to compute the final calculation result. Your answer can be a fraction. The time unit is in μs.]

Answer:

(i) time = 8000 × 0.4 + 20000 × 0.6

= 3200 + 12000

= 15200 μs

(ii) (1-p) × 1 + p × 15200 ≤ 4

15199 p ≤ 3

p ≤ 3/15199

(c) (2 pts) What is the Belady's Anomaly?

Answer: 即使可用的 frames (pages) 增加了，page fault rate 也沒有減少

(d) (2 pts) A system has 4 frames in the physical memory. The snapshot of the pages in the memory is shown in the following table. The reference bits and dirty bits of these pages are also shown in the following Table 1. Considering the enhanced second-chance replacement algorithm, when a page replacement is necessary, which page will be replaced?

Frame No.	Page	Reference bit	Dirty bit
0	Page 0	0	1
1	Page 1	1	1
2	Page 4	0	0
3	Page 2	1	0

Table 1: A snapshot of pages in the memory.

Answer:

page 4 , cause the reference bit and dirty bit are both zero

7 Process Management (12 pts)

Context switches in operating systems enable time-sharing multi-programming in computing systems. *Voluntary context switches* are triggered by the user applications when the applications have to wait for certain events and yield the right to use processing times; *involuntary context switches* are enforced by the scheduler in the operating system to suspend a process from using processors. Both of them are available in modern operating systems. Read the program shown below and answer the questions.

```
1 /* Skip headers to save space */
2 #define TRUE 1
3 #define FALSE 0
4
5 const int NUM_ITERATIONS = 1000;
6
```

```
/* Return the time in nano-second */
static inline unsigned long long time_ns();

int main(int argc, char **argv) {
  int opt, sflag=FALSE, iflag=FALSE, nswitches = 0;
  char str[128];
  struct rusage usageBefore, usageAfter;
  struct timespec deadline;

  while ( -1 != (opt = getopt(argc, argv, "is")) ){
    switch (opt) {
      case 'i': /* io */ {
        iflag = TRUE;
        break; }
      case 's': /* sleep */ {
        sflag = TRUE;
        break; }
    } /* end of switch */
  } /* end while */

  getrusage( RUSAGE_SELF, &usageBefore );
  const long long unsigned start_ns = time_ns();
  deadline.tv_sec = 0;
  deadline.tv_nsec = 10;

  for (int i = 0; i < NUM_ITERATIONS; i++) {
    memset(str, 0, sizeof(str));
    strcpy(str, "Hello World, OS Spring 2024.");
    if (iflag == TRUE) printf("%s\n", str);
    else if (sflag == TRUE)  nanosleep(&deadline, NULL);
  }
  const long long unsigned elapsed = time_ns() - start_ns;
  getrusage( RUSAGE_SELF, &usageAfter );
  printf("Voluntary Context Switch: %ld\n", usageAfter.ru_nvcsw-usageBefore.ru_nvcsw);
  printf("Involuntary Context Switch: %ld\n", usageAfter.ru_nivcsw-usageBefore.ru_nivcsw);

  nswitches = (int) (usageAfter.ru_nvcsw-usageBefore.ru_nvcsw) + (int) (usageAfter.ru_nivcsw-
      usageBefore.ru_nivcsw);
  printf("%i context switches in %llu ns (%.1ffns / switch)\n", nswitches, elapsed, (elapsed / (float)
      nswitches));
}
```

(a) (2 pts) How many lines of ''Hello World, OS Spring 2024.'' will the output show when `iflag` is set to `true`?

Answer:

1000

(b) (2 pts) Under what values of the flag `iflag` and `sflag`, defined at Line 11, the process will more likely be involuntarily context switched?

Answer:

iflag == FALSE

sflag == TRUE

(c) (4 pts) Please identify which line may lead to `Voluntary Context Switch`? 列出不只一個

Answer:

line 35, 40, 41, 44

(d) (4 pts) Which scheduling queues will the process be inserted when it encounters (a) voluntary context switch and (b) involuntary context switch? You may use the queueing diagram representation to illustrate your answer. However, it is not required.

Answer:

(a) the process will be insert to waiting queue

(b) the process will be insert to ready queue

Below is intentionally left blank for scratch and will NOT be graded.

130
50
250
400
650

200
410
115
500

$(1-p)\times 1$

$1-p+1520p \le 4$

$1+1519p \le 4$

$\frac{1}{12}+\frac{9}{12}$ $1519p \le 3$

$\frac{12}{10}$