ITP 30002 Operating System

Final Exam

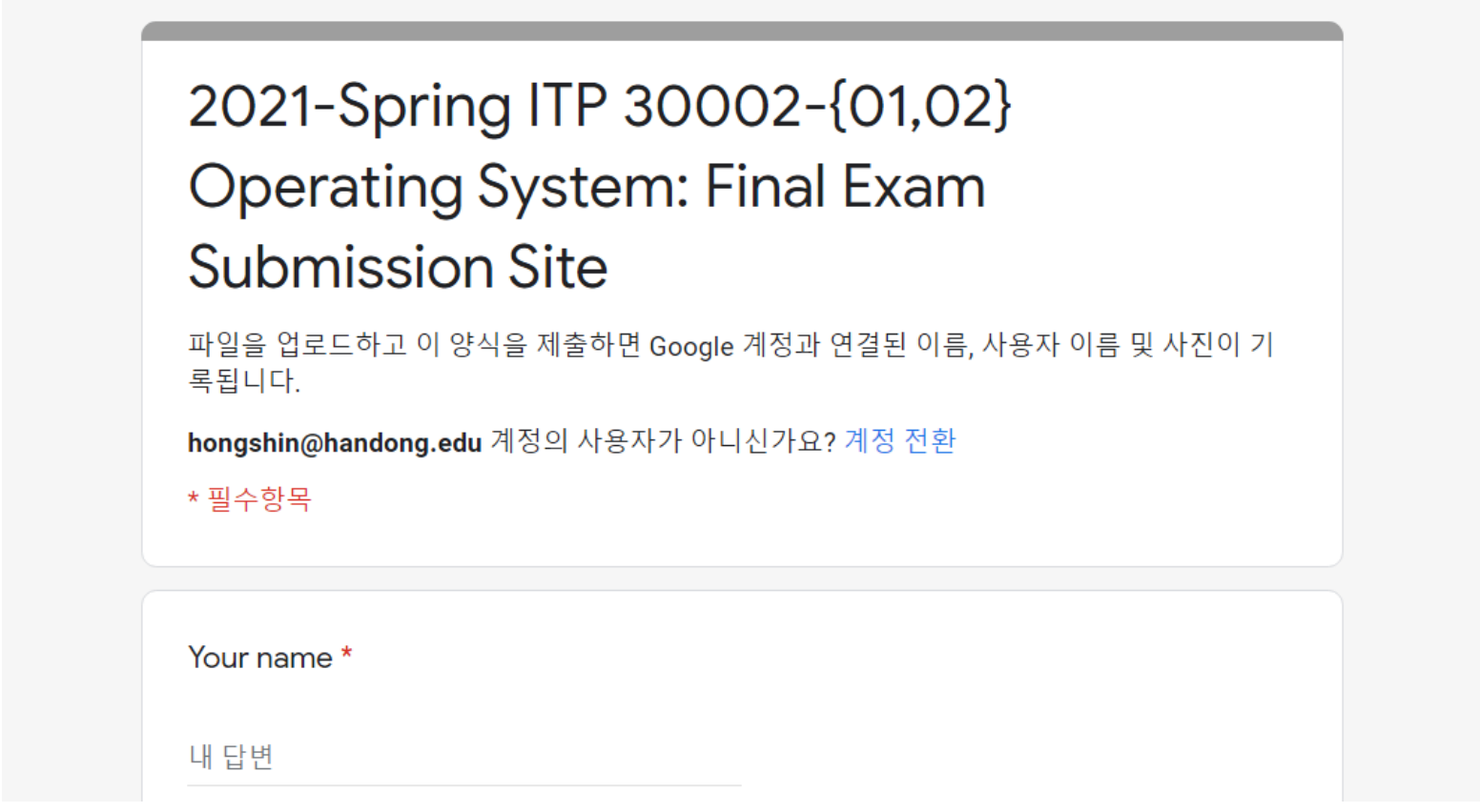
2021-06-17

Problems Overview

Problem 1	15 points	**Problem 6**	8 points
Problem 2	6 points	**Problem 7**	12 points
Problem 3	8 points	**Problem 8**	10 points
Problem 4	10 points	**Problem 9**	9 points
Problem 5	10 points	**Problem 10**	12 points

Submission Site

- https://forms.gle/TyacF4ZTKzwud2e77
  - this form requires your Handong sign-in
  - you can submit back-up files at the Hisnet homework repository



Problem 1 (15 points)

- Write in your own words to declare that you will uphold the honor code in taking this exam. Exceptionally, this problem can be answered in Korean.

c.f. the Handong CSEE Standard on Examination

1. Examination is an educational act necessary for evaluation of the students' achievement and for encouraging the students to absorb the material in the process of preparation.
2. Student should do their best to prepare for exams in order to improve her/his own knowledge and skill and should fully engage in the test during examination hour.
3. Accessing or providing unauthorized information, including other students' answer sheets, is regarded as cheating. The use of electronic devices, including cell phones and computers, without permission is strictly prohibited.
4. Entering or leaving the classroom during the examination before the finish time without permission is regarded as cheating.

Problem 2 (6 points)

Suppose that you just created a regular file in a file system and wrote 100 MB data to it. Explain the amount of the data that was actually written on the disk device.

Problem 3 (8 points)

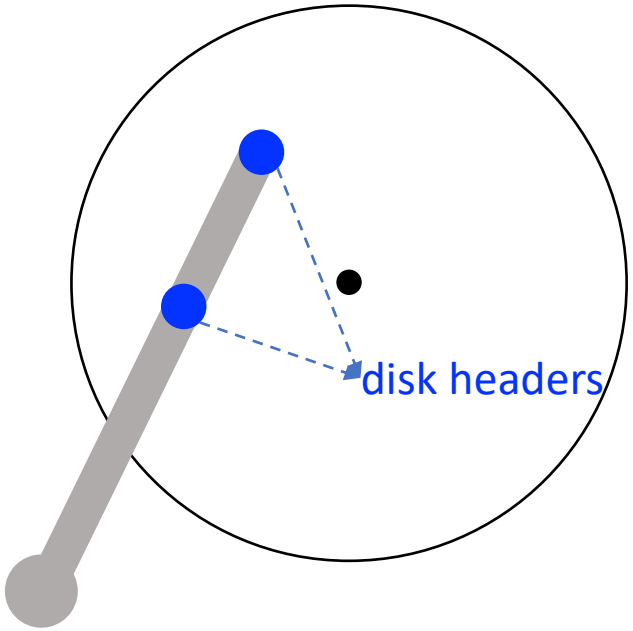
You are asked to implement a Fast File System to allow concurrent executions of file-related system calls. For high concurrency, you decide to allocate a mutex for each inode such that an operation on an inode must acquire the corresponding mutex prior to reading or writing the inode data.

Suppose that you are to implement a system call for file move (e.g., `mv`).

Discuss the possible deadlock issue regarding this operation (e.g., in which case a deadlock may occur and why), and suggest a valid solution to resolve this issue.

Problem 4 (10 points)

Suppose that you are given a brand new HDD where each disk arm has two headers to read two sectors simultaneously (the top view in section of this HDD is on the right side).



Discuss which aspects of Fast File System should be changed, and how these should be modified to make Fast File System to utilize HDD more efficiently.

Problem 5 (10 points)

Complete the three missing functions in the source code of a conditional variable implementation (see the next slide).

Note that this program uses a mutex and a semaphore to implement a conditional variable.

For simplicity, use the following shortend API names instead of the real ones:

- `sem_post()`
- `sem_wait()`
- `mutex_lock()`
- `mutex_unlock()`

Problem 5 (Con't)


```
typedef struct {
     int n_waiting ; // the number of waiting threads
     mutex l ; // to avoid race among cv operations
     semaphore sem ;
} my_cv ;


void my_cv_init (my_cond_var * cv)
{
     cv->blocked = 0 ;
     mutex_init(&(cv->lock)) ;
     sem_init(&(cv->sem), 0, 0) ; //initialize semaphore with counter 0
}


void my_cv_wait (mutex * m, my_cv * cv) {
     // TODO
}

void my_cv_notify (my_cv * cv) {
     // TODO
}

void my_cv_notify_all (my_cv * cv) {
     // TODO
}
```

Problem 6 (8 points)

You are given a HDD of the following specification:

- capacity: 1 TB
- RPM: 7200
- seek time: 6 ms to 12 ms
- max transfer rate: 105 MB/sec
- platters: 4
- the number sectors per track: 63

Suppose that you are reading from this disk a series of 10 sectors (8KB) that are consecutively numbered. How much time do you expect this driver to take to serve this reading request?

Problem 7 (12 points)

You can find in the next slide an implementation of a concurrent linked list which uses a single mutex to provide mutual exclusion of all operations.

Suppose that you want to (yes, indeed) revise this program to use a per-node mutext so that operations may run concurrently.

Complete the source code (fill in /*TODO*/'s)in the slide after the next slide.

(the changed lines in the given baseline code is highlighted in yellow)

Problem 7 – First Version


```
// basic node structure
typedef struct __node_t {
    int                 key;
    struct __node_t         *next;
} node_t;

// basic list structure (one used per list)
typedef struct __list_t {
    node_t                *head;
    pthread_mutex_t     lock;
} list_t;

void List_Init(list_t *L) {
    L->head = NULL;
    pthread_mutex_init(&L->lock, NULL);
}
```

```
void List_Insert(list_t *L, int key) {
    // synchronization not needed
    node_t *new = malloc(sizeof(node_t));
    if (new == NULL) {
        perror("malloc");
        return;
    }
    new->key = key;

    // just lock critical section
    pthread_mutex_lock(&L->lock);
    new->next = L->head;
    L->head   = new;
    pthread_mutex_unlock(&L->lock);
}
```

```
int List_Lookup(list_t *L, int key) {
    pthread_mutex_lock(&L->lock);
    node_t *curr = L->head;
    while (curr) {
        if (curr->key == key) {
            pthread_mutex_unlock(&L->lock);
            return 0; // success
        }
        curr = curr->next;
    }
    pthread_mutex_unlock(&L->lock);
    return -1; // failure
```

Problem 7 – Revised Version


```
typedef struct __node_t {
  int key ;
  struct __node_t * next ;
  pthread_mutex_t m ;
} node_t ;

typedef struct __list_t {
  node_t * head ;
  pthread_mutex_t m_head ;
}

void List_init (list_t * L) {
  L->head = NULL ;
  pthread_mutex_init(&(L->m_head), NULL);
}
```

```
void List_Insert (list_t *L, int key) {
  /*TODO*/
}

int List_Lookup (list_t *L, int key) {
  /*TODO*/
}
```

Problem 8 (10 points)

- Explain why security matters for operating system and what kinds of mechanisms that kernels have in order to provide security protection (within the scope of our classes so far)

Problem 9 (9 points)

Explain why LRU is preferable to FIFO as a page swapping strategy. Write your answer with an example with 3 frames and 5 pages where LRU performs better than FIFO.

Problem 10 (12 points)


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

int main () {
  char * s = malloc(100) ;
  scanf("%s", s) ;
  printf("%s", s) ;
}
```

<hello.c>

```
$ gcc hello.c
$ ls
a.out   hello.c   input.txt
$ ./a.out < input.txt
```

<bash>

Suppose that you had built `hello.c` as `a.out`, and now you are commanding bash to execute `./a.out < input.txt`.

Explain all possible system calls and traps that the operating system services for handling this command.