System Programming
Assignment #2: Finding Sequence

202111322 오색빛
컴퓨터공학부
제출일: 2022.12.01.

Design

1. multi-thread program 으로 병렬적으로 주어진 input file 내에서 제시된 substring 을 모두 찾아 그 sequence 의 시작 index 를 output file 에 출력하는 프로그램을 작성한다.

./ku_fs str i input output str: 찾을 문자열 i: 병렬 처리할 thread 의 개수 input: input file output: output file

2. input, output files

- input file
  - 첫 번째 줄은 input substrings 의 수를 나타낸다.
  - 각 substring 은 5bytes 이며, 마지막 줄만 1~5bytes 이다.
  - 각 substring 은 개행문자('₩n')로 구별된다.
- output file
  - 찾은 substring 의 시작 index 를 오름차순으로 출력한다.
  - 각 시작 index 는 개행문자('₩n')로 구별된다.

3. Scenario

1) 인자들을 받아 i 개의 thread 를 생성한다.
2) input, output 파일을 open 한다.
3) 생성된 thread 들이 input file 에서 병렬적으로 sequence 찾기를 수행한다.
4) thread 가 문자를 찾으면 그 substring 의 시작 index 를 linked-list 에 저장한다.(오름차순)
5) thread 종료되면 linked-list 에 저장된 data 를 output file 에 write 한다.
6) input, output 파일을 close 한다.

Implementation

1. Code Description

1) main()로부터 인자받기

```
if(argc != 5){
    exit(1);
}
if(atoi(argv[2]) < 1) {
    exit(1);
}

// arguments of main function
str = argv[1];
thread_num = atoi(argv[2]);
input_file = argv[3];
output_file = argv[4];

pthread_t thread_id[thread_num];
```

- main 함수로부터 들어온 인자의 개수가 다를 경우, 생성할 thread 의 개수가 0 초과가 아닌 경우 종료시켜 예외처리해주었다.
- main 함수로부터 들어온 인자들을 변수에 저장했다.
- thread_num(thread 개수)만큼의 thread 를 생성하여 id 를 저장하기 위해 thread_id 배열을 만들었다.

2) thread 분할 준비

```c
// open the input file
ifd = open(input_file, O_RDONLY, NULL);
if(ifd < 0) {
    perror("input file open");
    exit(1);
}

// read the number of input substrings(first line)
if((ret = read(ifd, buf, 5)) != 0) {
    if(ret == -1){
        exit(1);
    }
    line_num = atoi(buf);
}

// calculate the threads' start line
int line_arr[thread_num];
int quota = line_num / thread_num;
int remainder = line_num % thread_num;
for(int i=0; i<thread_num; i++) {
    line_arr[i] = quota;
}
if(remainder) {
    for(int i=0; i<remainder; i++){
        line_arr[thread_num - 1 - i]++;
    }
}
```

- input file 을 읽기전용으로 open 했다.
- input file 의 첫 번째 줄(5bytes)를 읽어 input substring 의 개수를 저장했다.
- 각 thread 마다 비슷한 양의 일을 처리하도록 처리할 개수를 담는 배열(line_arr)을 생성했다.
- 각 프로세스는 우선 input 배열의 크기를 thread 개수로 나눈 몫을 나눠 갖고, 만약 나머지가 존재한다면 뒤에서부터 1 씩 나눠 갖도록 구현했다.

3) thread 생성 및 종료

```c
// create threads
for(int i=0; i<thread_num; i++) {

    arg = (my_arg*)malloc(sizeof(my_arg));
    arg->input_file = input_file;
    arg->str = str;
    arg->start = 0;
    for(int j=0; j<i; j++) {
        arg->start += line_arr[j];
    }
    arg->num = line_arr[i];

    status = pthread_create(&thread_id[i], NULL, find_sequence, arg);
    if(status != 0) {
        perror("pthread_create");
        exit(1);
    }
}

// join threads (terminate and reap)
for(int i=0; i<thread_num; i++) {
    status = pthread_join(thread_id[i], &thread_result);
    if(status != 0) {
        perror("pthread_join");
        exit(1);
    }
}
```

- my_arg* 타입의 구조체를 각 thread 마다 따로 생성에서 thread 의 인자로 넣어주었다.
- thread 에 들어가는 arg 는 input_file, str 은 모두 동일하고, start(시작 line)와 num(처리할 line 수)는 계산한 결과에 따라 달라진다.
- 반복문과 pthread_create 을 통해 thread_num 개의 thread 를 생성해주었고, 오류가 나면 오류를 출력하고 종료한다.
- 각 thread 의 find_sequence 가 종료되면 reaping 할 수 있도록 pthread_join 을 사용하였으며, 오류가 나면 오류를 출력하고 종료한다..

4) 결과(linked-list)를 파일로 출력

```c
// open the output file
ofd = open(output_file, O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
if(ofd < 0) {
    perror("output file open");
    exit(1);
}

// write results on output file
if(head != NULL) {
    Node* n = head->next;
    char nbuf[BUF_SIZE];
    sprintf(nbuf, "%d\n", head->data);
    write(ofd, nbuf, strlen(nbuf));


    while(n != NULL){
        int num = n->data;
        sprintf(nbuf, "%d\n", num);
        write(ofd, nbuf, strlen(nbuf));
        n = n->next;
    }
}
```

- output 파일을 쓰기전용, 파일이 존재하지 않는다면 새로 생성, 파일이 존재한다면 덮어쓰기하도록 open 하였다.
- linked-list 가 NULL 이 아니라면, while 문을 통해 linked-list 값을 탐색하여 output file 에 write 하도록 작성하였다.
- linked-list 에 저장된 정수를 문자열로 변환하여 파일에 write 했다.

5) 파일닫기

```c
// close files
close(ifd);
close(ofd);
```

- input 및 output file descriptor 를 close 해주었다.

6) find_sequence

```
// open the input file
ifd = open(input_file, O_RDONLY, NULL);
if(ifd < 0) {
    perror("input file open");
    exit(1);
}

// repostition of file offset
start_idx = targ->start*5;
startpos = (targ->start+1)*6;
lseek(ifd, startpos, SEEK_SET);
```

- 각 thread 마다 개별적으로 file descriptor 를 갖도록 구현하였으며, 인자로 전달된 시작 substring 위치와 처리해야할 substring 의 개수를 가지고 시작 인덱스(start_idx)와 읽어야할 파일 위치(startpos)를 계산해주었다.
- lseek 으로 file descriptor 의 위치를 변경해주었다.

```
for(int i=0; i<targ->num;) {
    int found = start_idx;
    if(read(ifd, &buf, 1) == 0) {
        found = -1;
        break; // EOF
    }
    if(buf == '\n') {
        i++;
        continue;
    }
    prepos = lseek(ifd, 0, SEEK_CUR);
    for(int j=0; j<strlen(str); j++) {
        if(buf == str[j]) {
            if(read(ifd, &buf, 1) == 0) {
                found = -1;
                break; // EOF
            }
            continue;
        } else {
            if(buf == '\n') {
                j--;
                if(read(ifd, &buf, 1) == 0) {
                    found = -1;
                    break;
                } // EOF
                continue;
            }
            found = -1;
            break;
        }
    }
    if(found != -1) {
        pthread_mutex_lock(&mutex);
        insert(start_idx); // insert the start index to shared linked-list
        pthread_mutex_unlock(&mutex);
    }
    curpos = lseek(ifd, 0, SEEK_CUR);
    pos = prepos - curpos;
    lseek(ifd, pos, SEEK_CUR);
    start_idx++;
}
```

- 각 thread 는 자신이 맡은 양(num)을 처리한다. input 에서 읽은 '₩n'의 개수를 기준으로 반복문을 돌렸다.
- input 파일에서 '₩n'을 읽었다면 index 값은 증가시키지 않고, 다음 문자를 읽는다.
- input 파일을 읽다가 EOF 에 도달하면 어디서든 break 한다.
- input 파일에서 1bytes 씩 읽으면서 찾으려고 하는 substring 과 맞는지 검사한다. substring 과 다른 부분이 나오면 안쪽 반복문을 break 하고 다음 index 로 넘어가서 다시 검사를 진행한다.
- input 파일에서 substring 을 찾았다면, 시작 index 를 linked-list 에 저장한다.
- linked-list 는 thread 내에서 공유되므로 race condition 이 발생하지 않도록 mutex 를 이용하여 처리해주었다.
- curpos(검사후)와 prepos(검사전)를 이용해서 검사 후에 file descriptor 를 다음 index 로 넘겨 모든 sequence 를 찾을 수 있도록 구현했다.

7) insert() : shared linked list

```
void insert(int data)
{
    Node* ptr;
    Node* N = (Node*)malloc(sizeof(Node));
    N->data = data;
    N->next = NULL;

    if(head == NULL) {
        head = N;
    } else {
        if(head->data > N->data) {
            N->next = head;
            head = N;
            return;
        }
        for(ptr = head; ptr->next; ptr=ptr->next) {
            if(ptr->data < N->data && ptr->next->data > N->data) {
                N->next = ptr->next;
                ptr->next = N;
                return;
            }
        }
        ptr->next = N;
    }
}
```

- linked-list 가 새로운 node 를 insert 할 때, data 값을 기준으로 오름차순으로 정렬하여 저장하도록 코드를 작성하였다.
- 새로 저장할 node 가 적절한 위치에 도착하면, 구조체 node 의 next pointer 의 값을 변경해주었다.

```
typedef struct node
{
    int data;
    struct node *next;
} Node;
```

Function description

Function name	Arguments	Description
main	int argc	메인함수에 전달되는 정보의 개수
	char* argv[]	프로그램을 실행할 때 메인 함수에 전달되는 인자의 문자열들이 저장되는 배열, argv[0]은 프로그램 실행경로로 고정
	Return Value: int	정상적인 종료: 0, 비정상적인 종료: -1

find_sequence	void* arg	각 thread 들의 main 함수처럼 실행될 함수로, find_sequence 함수 실행에 필요한 인자들을 구조체로 선언하여 넣어주었다
		`typedef struct {`
		`  char* input_file;`
		`  char* str;`
		`  int start;`
		`  int num;`
		`} my_arg;`

	Return Value: void*	thread 의 함수 실행 후 반환되는 값을 넘기는 데 사용된다. 이번 과제에서는 의미 없는 값을 반환해줬다.
insert	int data	linked-list 에 저장될 데이터 값(찾은 substring 의 시작 index)
	Return Value: void	