Assignment2 레포트

컴퓨터공학과 / 201510621 / 김민혁

<ku_pfred.c 설명>

DATA 구조체, int* frequency함수 , main 으로 구성되어 있다.

먼저 자료파일을 열어 자료의 첫 줄을 읽어 첫 줄의 글자수, 자료의 개수를 알아온다. 이 정보를 포함해 frequency 함수의 계산에 필요한 정보들을 구조체에 담아준다. 그리고 mqd_t mqdes 배열을 프로세스의 개수만큼 mq_open 해주어 프로세스 개수만큼의 메시지 큐를 열어준다.

그 후 반복문안에서 프로세스 넘버를 구조체에 넣어준다. 그리고 fork를 해서 (pid == 0)이면(child 프로세스이면) 반복문에서 빠져나가게 한다. 즉 프로세스가 생성되면서 자식들은 각자 자신의 프로세스를 받아서 다음 코드로 넘어간다.

그 후 모든 자식들이 frequency 함수를 이용하여 도수분포표를 계산하고 각자의 메시지 큐를 이용해 mq_send를 해준다.

부모 프로세스는 자식 프로세스들이 끝날 때까지 wait 하였다가 메시지 큐들에서 정보들을 mq_receive하고 반복문을 돌면서 도수분포표를 업데이트해준다.

이를 반복문을 통해 출력하고 종료한다.

부모는 계산을 안하기 때문에 사실상 프로세스 개수를 n개라고 입력하면 실제로는 n+1개의 프로세스가 돌아가고 있다.

<ku_tfred.c 설명>

DATA 구조체, void* frequency함수 , main 으로 구성되어 있다.

전역변수로는 도수분포표를 표현할 정수형 배열 freq, 쓰레드 번호를 표현할 정수형 변수 k, mutex 가 있다.

먼저 자료파일을 열어 자료의 첫 줄을 읽어 첫 줄의 글자수, 자료의 개수를 알아온다. 이 정보를 포함해 frequency 함수의 도수분포표 계산에 필요한 자료들을 구조체에 담아준다.

그 후 쓰레드의 개수만큼 반복문을 돌면서 pthread_creat해준다. 여기서 frequency 함수가 구조체의 주소를 인자로 받으며 실행된다. IPC와 달리 전역변수에 결과값을 동시에 저장하므로 쓰레드 번호를 받아오고 증가시키는 구간에 mutex lock과 unloc을 걸어주고, freq배열을 증가 시킬 때 lock, unlock을 걸어준다.

그후 모든 쓰레드들을 join 해주고 결과값을 출력한 뒤 종료한다.

<함수 설명>

1. ku_pfred.c

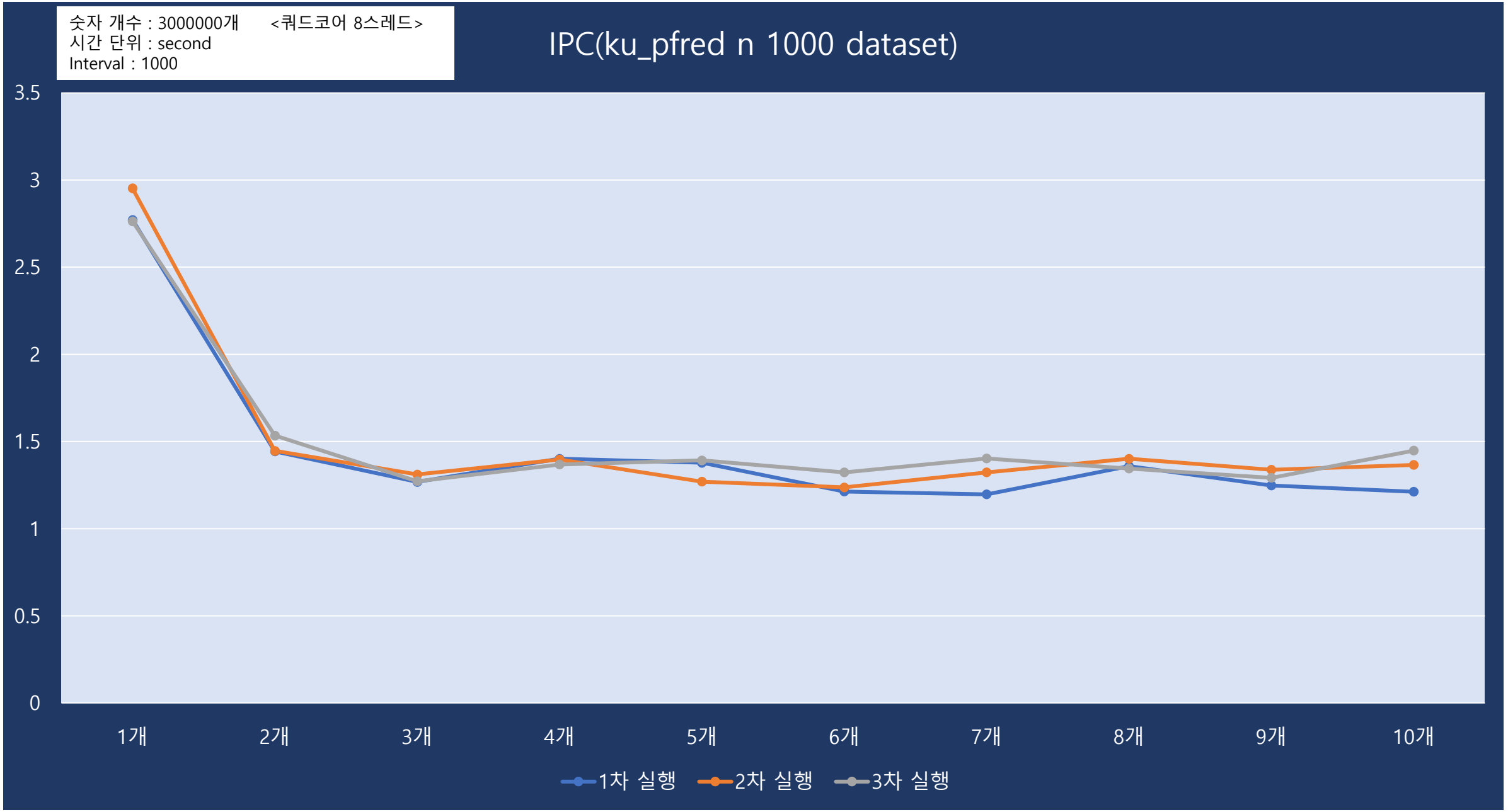
Function Name	Arguments	Description
frequency	(DATA* data)	Data 구조체 포인터를 인자로 받는다. 이 구조체는 {스레드의 개수 숫자(자료)의 개수 첫줄의 글자수 자료 파일의 이름 파일 포인터 프로세스 번호} 를 가지고 있다.
	Return type	
	int* temp	계산 결과를 return해준다.

2. ku_tfred.c

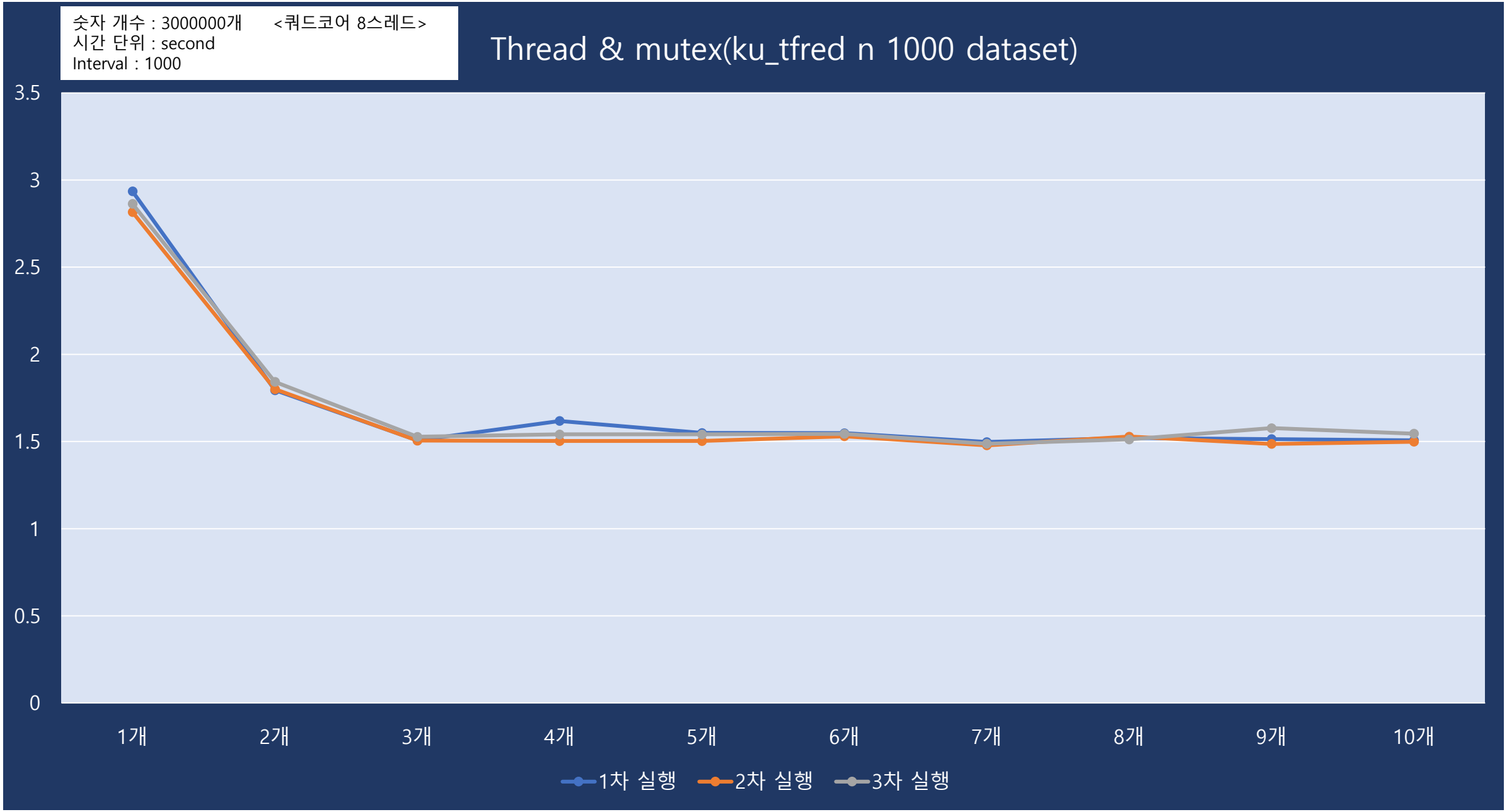
Function Name	Arguments	Description
frequency	(void* data)	Data 구조체 포인터를 인자로 받는다. 이 구조체는 {스레드의 개수 숫자(자료)의 개수 첫줄의 글자수 자료 파일의 이름 파일 포인터} 를 가지고 있다.
	Return type	
	void*	반환을 하지 않는다.

위의 두 함수는 작동방식이 같은 도수분포표를 구하는 함수이다.
자료 파일을 한 줄씩 읽어들여 정수값으로 변환한다. 그리고 이 정수값을 구간의 범위로 나누면 그 정수가 어느 구간에 속하는지 나오는데 그 구간의 freq 정수 배열값에 1을 더해준다. 이 수행을 파일이 끝날때까지 반복하며 프로세스나 스레드의 번호에 따라 파일을 읽을 범위를 나눠가진다.

숫자 개수 : 3000000개 <쿼드코어 8스레드>
시간 단위 : second
Interval : 1000



숫자 개수 : 3000000개 <쿼드코어 8스레드>
시간 단위 : second
Interval : 1000



<Performance evaluation>

쓰레드를 늘렸을 때와 프로세스를 늘렸을 때 둘의 그래프를 보면 비슷한 양상을 보였다.
둘 다 1개에서 2개로 늘렸을 때 가장 많은 performance 차이를 보였다. 그 후 3개로 늘렸을 때는 아주 미미한 차이를 보여주었으며 4개부터 10개까지는 거의 차이가 없었다.

이 실행들이 쿼드코어 컴퓨터에서 수행되었기 때문에 4개에서부터 속도의 변화가 적을 것으로 예상했지만 3개에서부터 속도의 변화가 거의 없어졌다.

IPC에서는 부모 프로세스를 포함해서 총 n+1개의 프로세스가 작동하고 있기 때문으로 보인다.

thread에서는 mutex_lock ,mutex_unlock을 frequenc함수의 두부분에서 하기 때문에 thread가 늘어나도 이 부분에서 딜레이가 더 많이 생긴다. 그래서 두 부분이 서로 상쇄시키며 계속 비슷한 시간이 걸리는 것으로 보인다.