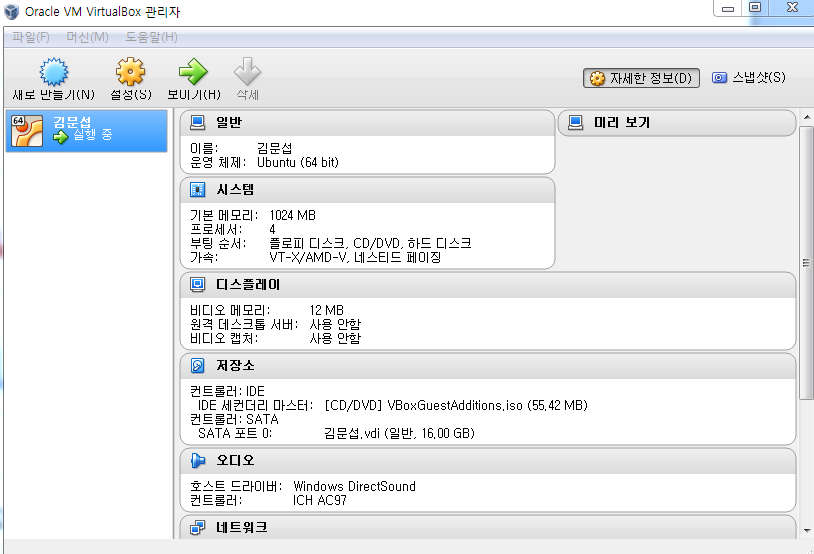
**운영체제 과제#4 20110181김문섭**

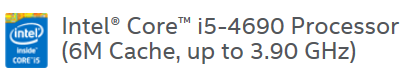
**문제분석**

Batch.txt에 들어있는 파일의 개수와 파일명, 파일 각각의 우선순위 내용을 읽어들어서,

POSIX pthread.h에 구현되어 있는 함수들을 사용하여 1~16개의 쓰레드를 가지는 쓰레드풀을 생성하고, 각각의 쓰레드들이 semaphore로 동기화 된 전역변수인 histogram[]에 동시 접근하면서 각각의 파일들은 우선순위(0~4)값을 가지는데, 가장 낮은 우선순위를 가진 데이터파일을 우선적으로 처리해주는 방식으로 스레드를 스케줄하여 가장 빠른시간에 데이터 파일의 히스토그램을 구할 수 있는 프로그램을 작성한다.

**컴퓨터 사양**

****

****

CPU : intel core i5-4690m 3.90Ghz (쿼드코어) 하이퍼스레딩X

메모리 : 1024MB

운영체제 : 우분투 12.04 (멀티코어 개수 4개로 설정)

컴파일러 : gcc-4.6 version

라이브러리 : 사용않함

**설계**

**전체 프로그램의 구조**

-쓰레드풀을 생성하는 모듈.

-사이즈 N인 우선순위 큐에 값을 전달하는 모듈

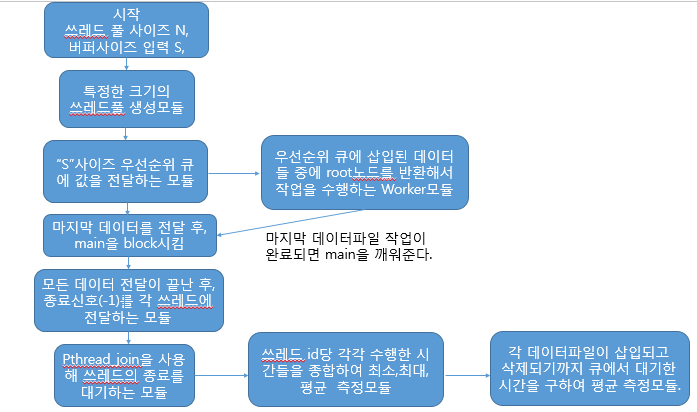
-모든 작업이 끝나고나서 우선순위 큐에 -1을 전달해서 쓰레드종료를 유도하는 모듈

-쓰레드 풀에서 생성된 쓰레드 개수만큼 쓰레드의 종료를 기다려주는 모듈

-생성된 쓰레드풀이 작업하게되는 worker()함수

-각 스레드풀의 시간정보를 정렬시키고 원하는 시간데이터를 출력하는 모듈.

-각 우선순위당 누적 대기시간을 종합해서 평균 대기시간을 측정하고 출력하는 모듈.



**자료구조**

**-**data 구조체 : data파일 이름, data파일 우선순위, data파일을 수행시 시작,종료시간을 멤버로 가짐.

-min\_heap 구조체 : data구조체의 포인터, 힙 사이즈를 멤버로 가짐.

-hissum[256] : 배열에 데이터파일의 히스토그램을 저장한다.

-histogram[256] : hissum의 값들을 누적해서 모든 데이터파일의 히스토그램을 구한다.

-struct timespec starttime,endtime : 프로그램 수행시간을 구하기 위해서 timespec구조체 사용.

-sem\_t sem,empty,full,sig,create,end,enter : 다중쓰레드프로그래밍도중에 레이스컨디션을 막기위해서 변수들을 동기화해주기 위한 각각은 세마포어 사용.

(sem : histogram[] 전역배열에 접근하기 위해서 사용.

Priority\_wating\_time[5] : 각 우선순위마다 큐에서 기다린 시간을 누적하는 값을 저장하는 배열

Priority\_size[5] : batch.txt에 1~128데이터파일이 0~4우선순위를 가지는데 각 우선순위가 몇 개씩 파일을 가지는지 개수를 counting하는 배열

Buffer : 버퍼사이즈만큼 데이터를 우선순위 큐에 삽입하기 위해 사용.

full : 스레드풀에서 사용중인 개수를 표현하기 위해 사용.

sig : 버퍼에 변수를 하나씩 전달하기 위해서 사용

create : 각 스레드별로 ID를 부여하기 위해서 사용.

end : 모든 작업을 스레드들에게 전달해주었다는 것을 알리기 위해 사용.

enter : 모든 스레드들이 종료값을 전달 받았는지 확인하기 위해서 사용.)

h\_sem : 우선순위 큐에 접근할 때 삽입과 삭제연산이 동시에 진행되지 않게하기 위해 사용.

Pthread\_t\* threads : 스레드를 생성하고 각 스레드들을 기다리기 위해서 사용.

fp1, fp2 : data파일들을 읽기 위해서, histogram.bin파일을 생성하기 위해서 사용.

버퍼에 있는 데이터를 다른 쓰레드에서 사용해서 버퍼가 비어있는 경우, 쓰레드풀의 모든 쓰레드들이 작업을 하고있거나 모든 쓰레드들에게 종료신호를 모두 보내기 전까지 block이 걸려서 지연시간이 발생하게 된다.

**알고리즘**

1. 시작과 동시에 프로그램 전체 수행시간 출력을 위한 시작시간을 측정한다.

2. 입력인자 값에 알맞은 개수의 쓰레드 풀을 생성한다

For(i=0;i<pool\_size;i++)

Pthread(thread[i],NULL,worker,NULL);

3. 우선순위 큐에 각 값을 삽입하고, 우선순위 큐의 값을 각 쓰레드들이 delete하면서 값을 반환 받아서 작업을 수행한다

[작업 : 각 데이터 파일의 값들을 hissum[]배열에 카운팅 해주고 histogram[]배열에 그 값을 누적 시키는 작업

4. 3번 과정을 우선순위 큐 사이즈만큼 반복해주고, 우선순위 큐의의 모든 공간이 사용상태이면 Block()상태가 된다

--------------------------------------------------------------------------------------------------------------

5. 모든 인자들의 전달이 끝나고 모든 파일들의 데이터 histogram 누적이 끝나게 되면 histogram.bin파일에 해당 데이터를 write해준다

[histogram[]배열의 값을 histogram.bin파일에 write해준다.] 과제 1~2와 동일

--------------------------------------------------------------------------------------------------------------

6. 쓰레드풀 개수만큼 종료조건 -> “-1”의 값을 우선순위 큐로 전달해서 모든 쓰레드가 -1값을 전달받을 수 있도록 한다

sem\_init(&sem, 0, 1);

data t\_data;

t\_data.priority = -1;

for (i = 0; i < pool\_size; i++)

{

sem\_wait(&h\_buffer);

sem\_wait(&h\_sem);

insert\_min\_heap(&heap, t\_data); //최소힙 삽입

sem\_post(&h\_sem);

sem\_post(&full);

}

7. pthread\_join()를 쓰레드 풀 개수만큼 실행시켜서 우선순위가 -1인 값을 전달받은 쓰레드들이 종료될때까지 기다려준다.

for (k = 0; k < pool\_size; k++)

pthread\_join(threads[k], (void\*)&status);

8. 정상적으로 pthread\_join()이 끝나서 Block이 풀리게되면 쓰레드풀의 각 쓰레드들의 실행누적시간을 구해서 최소,최대,평균 시간을 구해주고 각 우선순위들의 대기시간의 평균을 종합하고, 총 프로그램 수행시간을 출력시켜준다

**세부 구현사항**

**가.** char name[20];

int priority;

data f\_data;

for(i=1;i<=f\_count;i++)

{

fscanf(fp, "%s %d", name, &priority);

strcpy(f\_data.f\_name, name); f\_data.priority = priority;

sem\_wait(&h\_buffer);

sem\_wait(&h\_sem);

clock\_gettime(CLOCK\_REALTIME, &f\_data.data\_start); //삽입될 때 시간 측정(waiting\_time)

insert\_min\_heap(&heap, f\_data); //최소힙 삽입

sem\_post(&h\_sem);

sem\_post(&full);

}

sem\_wait(&end);

batch.txt파일의 데이터파일, 우선순위 형식의 데이터들을 fscnaf() 함수를 사용해서 name, priority변수에 데이터 값을 저장하고 해당 값을 f\_data변수에 초기화시킨 후, heap에 삽입하게 된다. 삽입할때는 h\_buffer 세마포어를 사용해서 버퍼사이즈 개수만큼만 삽입되고 그 이상이 삽입되고자 할때는 블럭상태에 빠지게 된다. H\_sem 세마포어는 main의 삽입과 worker의 삭제가 같은 heap에 대한 연산이기 때문에 동시에 동작하게 될 경우 문제가 생길 수 있어서 따로 세마포어를 두어 관리를 하고 있습니다.

삽입 직전에 해당 데이터에 시작시간을 측정하고

……이전코드……

while(1)

{

Sem\_wait(&full);

//////////////////////////////////////////////////

sem\_wait(&h\_sem);

h\_data = delete\_min\_heap(&heap); //힙에서 값을 반환

priority = h\_data.priority;

strcpy(filename, h\_data.f\_name);

clock\_gettime(CLOCK\_REALTIME, &starttime);

clock\_gettime(CLOCK\_REALTIME, &h\_data.data\_end); //반환되고나서 waiting\_time측정을 위해 시간 측정

sem\_post(&h\_sem);

if (priority == -1)

break; //데이터가 모두 전달되고 -1전달을 받앗으면 쓰레드 종료

data\_time = (double)((h\_data.data\_end).tv\_sec - (h\_data.data\_start).tv\_sec) \* 1000 + (double)((h\_data.data\_end).tv\_nsec - (h\_data.data\_start).tv\_nsec) / 1000000;

printf("@@@@@@@@@@@@@%s %lf\n",filename,data\_time);

priority\_size[h\_data.priority]++;//우선순위에 따라 카운트 증가

priority\_wating\_time[h\_data.priority] += data\_time; //각 우선순위들의 waiting\_time 누적

……………코드 계속…………

힙에서 값을 반환할 때 종료시간을 측정해서 두 개의 시간으로 데이터가 큐에서 waiting\_time을 측정하게 됩니다.

**다**.worker thread들의 작업이 모두 끝난 것을 main thread는 어떻게 아는가?

data t\_data;

t\_data.priority = -1; //데이터 전달이 끝나고나서 “-1”의 우선순위를 가지는 데이터 삽입.

for (i = 0; i < pool\_size; i++)

{

sem\_wait(&h\_buffer);

sem\_wait(&h\_sem);

insert\_min\_heap(&heap, t\_data); //최소힙 삽입, 우선순위 “-1”이다.

sem\_post(&h\_sem);

sem\_post(&full);

}

과제 3과 동일한 방식인데, 쓰레드 풀의 개수만큼 우순선위 값을 “-1”로 가지는 data값을 heap으로 삽입하게 된다. 삽입하게 되면 worker의 함수 코드 부분

if (priority == -1)

break;

이런 코드를 삽입해서 해당 스레드가 while(1)루프를 빠져나와서 종료되게 되고 **라** 과정을 실행하게 된다.

**라**.작업이 모두 끝나면 main thread는 어떻게 worker thread들을 종료 시킬 것인가?

특별한 작업을 보낸 후 pthread\_join()을 수행한다.

for (i = 0; i < pool\_size; i++)

pthread\_join(threads[i], (void\*)&status);

에서 쓰레드가 break; 되기를 기다리고 있다가 break된 쓰레드 풀의 쓰레드들을 개수만큼 정상종료되도록 해준다.

**마.** 마지막으로 측정한 시간들의 종합한 데이터를 출력해준다.

double sum=0;

for(i=0;i<pool\_size;i++)

sum+=threads\_time[i];

printf("MAX TIME : %lf\n",threads\_time[pool\_size-1]);

printf("MIN TIME : %lf\n",threads\_time[0]);

printf("AVG TIME : %lf\n",sum/pool\_size);

clock\_gettime(CLOCK\_REALTIME, &total\_end);

double total\_time = (double)(total\_end.tv\_sec - total\_start.tv\_sec) \* 1000 + (double)(total\_end.tv\_nsec - total\_start.tv\_nsec) / 1000000;

printf("total\_time : %f\n\n", total\_time);

////////////////////////////////////waiting time 출력////////////////////////////////////

for (i = 0; i < 5; i++)

{

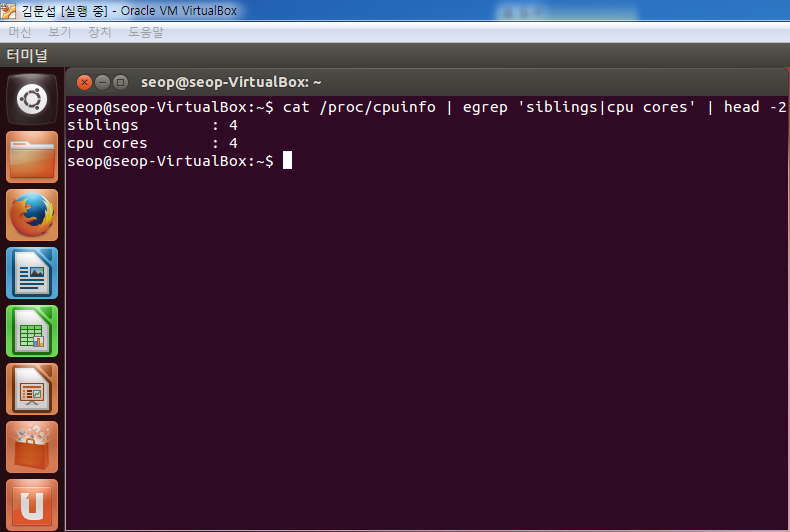
priority\_wating\_time[i] /= priority\_size[i];

printf("\"%d\" priority\_wating\_time : %lf\n", i,priority\_wating\_time[i]);

}

////////////////////////////////////출력종료 프로그램////////////////////////////////////

**성능 평가**

****

Hyper thread 사용않함.

(siblings 값이 cpu cores값의 2배를 가지면 하이퍼 스레딩을 사용하는 환경)

컴퓨터 사양

-문제분석 단락에 내용 첨부

**실행 데이터**

**Buffer\_Size 10 고정, pool\_size만 변경**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 쓰레드 개수(개)  (data1~128.bin 파일 대상) | 쓰레드 수행시간(ms) | | | 프로그램 수행시간  (ms) |
| 최소 | 평균 | 최대 |
| 1 | 77.336139 | 77.336139 | 77.336139 | 85.100188 |
| 2 | 50.904405 | 50.952202 | 51.000000 | 58.741248 |
| 3 | 42.000000 | 42.541972 | 43.625916 | 50.135105 |
| 4 | 33.161512 | 35.790378 | 38.000000 | 49.046530 |
| 5 | 29.236866 | 31.114607 | 35.000000 | 42.844132 |
| 6 | 34.000000 | 40.911123 | 44.620323 | 52.866796 |
| 7 | 71.076659 | 71.953749 | 73.106420 | 82.119474 |
| 8 | 21.461016 | 26.841450 | 39.000000 | 51.042842 |
| 9 | 28.101729 | 33.396836 | 40.000000 | 51.116292 |
| 10 | 25.476202 | 28.314634 | 32.000000 | 49.324060 |
| 11 | 12.000000 | 16.659884 | 20.000000 | 24.867827 |
| 12 | 20.791714 | 27.702188 | 37.655713 | 50.179558 |
| 13 | 25.577180 | 31.525126 | 39.000000 | 51.092755 |
| 14 | 29.422400 | 36.276377 | 47.000000 | 66.913469 |
| 15 | 14.452532 | 22.951153 | 31.000000 | 49.547950 |
| 16 | 20.000000 | 27.659061 | 34.000000 | 61.135796 |

**표1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Thread Pool 사이즈 | Priority당 waiting\_time(ms) | | | | |
| 0 | 1 | 2 | 3 | 4 |
| 1 | 0.300163 | 0.199526 | 0.681187 | 2.183588 | 4.819547 |
| 2 | 0.093224 | 0.044711 | 0.275361 | 1.294258 | 6.470967 |
| 3 | 0.252192 | 0.205300 | 0.594833 | 1.259361 | 4.196744 |
| 4 | 0.268823 | 0.221745 | 0.206590 | 0.915741 | 4.809884 |
| 5 | 0.088415 | 0.131924 | 0.381726 | 0.405984 | 3.931706 |
| 6 | 0.322795 | 0.383499 | 0.281262 | 0.475385 | 4.299485 |
| 7 | 0.424913 | 0.242594 | 0.360032 | 0.651768 | 3.755581 |
| 8 | 0.236708 | 0.344032 | 0.626419 | 0.618659 | 2.900923 |
| 9 | 0.108824 | 0.075980 | 0.126005 | 0.446501 | 1.666396 |
| 10 | 0.200579 | 0.097230 | 0.199398 | 0.432119 | 1.044851 |
| 11 | 0.080729 | 0.060497 | 0.051092 | 0.077662 | 0.262228 |
| 12 | 0.087110 | 0.130795 | 0.088005 | 0.450358 | 0.837246 |
| 13 | 0.051474 | 0.036804 | 0.098136 | 0.187306 | 0.244912 |
| 14 | 0.217052 | 0.025886 | 0.057659 | 0.126635 | 0.414189 |
| 15 | 0.063671 | 0.082045 | 0.067060 | 0.135648 | 0.297749 |
| 16 | 0.076211 | 0.136562 | 0.095828 | 0.111382 | 0.270492 |

**표2**

**성능 평가**

우선순위가 낮은 데이터파일이 큐에 들어가게되면 최소 힙으로 데이터들을 정렬하게 되기 때문에 가장 낮은 우선순위 데이터파일을 우선적으로 처리하게됩니다. 결과에서 보듯이 우선순위 0~4 순으로 waiting\_time이 커지는 결과를 보이고 있습니다.

버퍼사이즈가 커지면 커질수록 우선순위에 따른 대기시간의 차이가 서서히 줄어들어서 쓰레드 풀의 사이즈가 16일때는 거의 비슷한 대기시간을 가지는 것으로 보입니다.

**결론**

운영체제가 실행되면서 모든 프로세스들의 우선순위가 같을 수는 없기 때문에 각각의 다른 우선순위에 대해서 효과적으로 프로세스들에게 실행권을 주기위해서 우선순위 큐를 사용해서 스케줄러를 작성해야하고, 버퍼사이즈가 1인경우에는 각 우선순위에 대해서 똑 같은 대기시간을 가지지만 버퍼사이즈가 커지면서 우선순위에 따른 대기시간이 확실히 나타나게 됩니다.

표2의 결과를 쓰레드풀의 사이즈가 10개정도보가 작을때는 0~3의 우선순위는 대기시간이 크게 차이나지 않지만 “4”의 우선순위를 가지는 데이터파일의 대기시간을 보면 0~3과 비교해서 확실히 긴 대기시간을 가지는 것을 보인다.

하지만 쓰레드풀의 크기가 충분히 커지면 대기시간 없이 많은 쓰레드들에 의해서 프로그램이 동작하기 때문에 우선순위에 따른 대기시간 차이가 미미하지만, 분명한 차이는 보이고 있습니다.

**참고 문헌**

우분투 소프트웨어 패키지인 헥사코드 뷰어 OKteta사용법 및 다운로드

<http://packages.ubuntu.com/search?keywords=okteta>

인텔 시피유 사양 제공

<http://ark.intel.com/ko/products/80810/Intel-Core-i5-4690-Processor-6M-Cache-up-to-3_90-GHz>

우분투 하이퍼 스레딩 사용여부 확인 명령어

<http://zetawiki.com/wiki/%EB%A6%AC%EB%88%85%EC%8A%A4_%ED%95%98%EC%9D%B4%ED%8D%BC%EC%8A%A4%EB%A0%88%EB%94%A9_%ED%99%95%EC%9D%B8>

Clock\_gettime()함수 사용법

<http://linux.die.net/man/3/clock_gettime>

Glibc detected \*\*\* : free(): invalid pointer 오류 해결 참고

<http://www.hme.or.kr/m/post/68>

sem\_wait()사용법

[**http://linux.die.net/man/3/sem\_wait**](http://linux.die.net/man/3/sem_wait)

sem\_post()사용법

[**http://linux.die.net/man/3/sem\_post**](http://linux.die.net/man/3/sem_post)

[**http://www.hme.or.kr/m/post/68**](http://www.hme.or.kr/m/post/68)

pthread\_create()사용법

[**http://linux.die.net/man/3/pthread\_create**](http://linux.die.net/man/3/pthread_create)

pthread\_join()사용법

[**http://linux.die.net/man/3/pthread\_join**](http://linux.die.net/man/3/pthread_join)

fscanf()사용법

[**http://linux.die.net/man/3/fscanf**](http://linux.die.net/man/3/fscanf)

strcpy()사용법

[**http://linux.die.net/man/3/**](http://linux.die.net/man/3/pthread_join)**strcpy**

최소 heap 소스코드(insert\_min\_heap, delete\_min\_heap)

C언어로 쉽게 풀어쓴 자료구조 [저 천인국] 소스 참고 및 수정.

**실행 스냅샷**

**실행 명령어 : ./hw4 batch.txt 5 10 으로 실행했을 때의 스냅샷.**

