

PROJ5 보고서

소프트웨어학부 2019007329 이지현

1. 알고리즘 설명

➤ pthread_pool.h

pthread_pool_t struct에 running 변수를 bool이 아닌 int형으로 바꿔주었다. 기본 실행상태를 0, POOL_COMPLETE 종료 상태를 1, POOL_DISCARD 종료 상태를 2로 한다.

➤ worker

스레드가 작업을 수행할 함수이다. 스레드 풀을 생성하여 worker의 param 파라미터를 할당한다. 즉시 종료하는 옵션 POOL_DISCARD인 pool->running == 2가 발생하는 상황을 제외한 pool->running < 2일 경우 while문을 계속해서 반복한다. while문을 들어오면 대기열 접근을 위해 pool->mutex 락을 건다. 종료 명령이 없고(pool->running == 0) 대기열에 작업이 없는 경우(pool->q_len == 0) while문을 돌며 pool->full조건변수가 pthread_cond_wait으로 대기하도록 한다. while문을 빠져나온 뒤, 그 후 종료 명령이 발생했을 수도 있기 때문에 조건문으로 종료 명령 상태를 확인한다. running == 2이면, 락을 풀 뒤 pthread_exit(NULL)로 바로 종료한다. running == 1이고 대기열의 길이가 0이면 COMPLETE 종료 옵션이 발생했고 대기열에 있는 모든 스레드가 실행이 완료되었다는 의미이므로 마찬가지로 락을 풀 뒤 종료한다. 종료 명령이 없다면 작업을 진행한다. task_t 타입의 task에 (pool->q)[pool->q_front]로 버퍼의 q_front 인덱스에 있는 작업을 꺼낸다. 작업이 하나 빠져나왔으니 q_len을 하나 줄여주고, task.function(task.param)으로 작업을 실행한다. 작업을 실행시켰으니 empty 조건변수에 signal을 보내고 락을 해제한다.

➤ pthread_pool_init

스레드풀을 생성하는 함수이다. 일꾼 스레드의 크기와 대기열의 크기가 max를 넘으면 POOL_FAIL를 return하고 일꾼 스레드의 수보다 버퍼의 용량이 작으면 버퍼의 용량을 일꾼 스레드로 조정한다. 일꾼 스레드와 대기열을 malloc을 이용해 공간을 할당해준다. 그 후 pthread_pool_t 구조체의 모든 변수를 초기화시킨다. running은 기본 실행값인 0으로, q_size는 사용자가 요청한 버퍼 용량인 queue_size로, q_front와 q_len은 현재 버퍼가 비어있고 실행이 되지 않은 상태이므로 0으로, 일꾼 스레드도 사용자가 요청한 bee_size로 값을 할당한다.

일꾼 스레드를 동기화시킬 용도로 사용할 mutex, full, empty를 pthread_mutex_init, pthread_cond_init으로 초기화 한 후, 일꾼 스레드를 pthread_create로 생성한다. 생성한 pool->bee[i] 스레드가 worker 함수의 파라미터로 pool을 넘겨주며 실행하도록 설정하였다. 모든 과정을 마치면 POOL_SUCCESS를 return한다.

- pthread_pool_submit

스레드풀에서 실행시킬 함수와 인자의 주소를 넘겨주며 작업을 요청하는 함수이다. 대기열 접근을 위해 락을 건 뒤, 대기열이 꽉 찼을때(`q_len == q_size`) `flag`가 `POOL_NOWAIT`이라면 기다리지 않고 바로 락을 푼 뒤 `POOL_FULL`을 return한다. `flag == POOL_WAIT`인 경우 대기열이 빌때까지 while문을 반복하며 empty 조건 변수를 wait 시킨다. 작업 함수를 설정하기 위해 `task_t` 타입의 `task`를 정의하고, `task.function`으로 사용자가 제공한 `f`를 할당하고, `task.param`으로 `p`를 할당한다. 대기열에 작업을 넣기 위해 작업을 넣을 위치 `int tsak_back`을 `(pool->q_front + pool->q_len) % pool->q_size`로 설정한다. 다음에 실행될 위치와 대기열의 길이를 더하면 작업이 들어있는 대기열의 가장 끝 인덱스를 구할 수 있다. `q[task_back]`에 `task`를 할당하고, 작업을 하나 넣었기 때문에 `q_len`을 하나 늘린다. 작업 요청이 끝났으므로 조건변수 `full`에 `signal`을 보내고 락을 해제한 뒤 `POOL_SUCCESS`를 return한다.

- pthread_pool_shutdown

스레드풀을 종료하는 함수이다. 대기열 접근을 위해 락을 걸고 `how == POOL_COMPLETE`이면 `running`을 1로 바꾼다. 그 후 모든 대기중인 `full`, `empty` 조건변수를 깨우기 위해 `pthread_cond_broadcast` 사용한다. `how == POOL_DISCARD`이면 `running`을 2로 바꾸고 마찬가지로 `broadcast`로 대기중인 모든 `full`, `empty`를 깨운다. 대기열 접근이 끝났으니 락을 해제하고 일꾼스레드 `bee`를 `pthread_join` 한다. 마지막으로 `pthread_mutex_destroy`로 `mutex`, `full`, `empty`를 반납하고, `bee`와 `q`에 할당한 공간 또한 `free`로 해제한다. 마지막으로 `POOL_SUCCESS`를 `return`한다.

2. 컴파일

```
os@os-VirtualBox:~/Documents/theory_assignment/proj5$ make
gcc -o client client.o pthread_pool.o -lpthread
```

3. 실행 결과

```
os@os-VirtualBox:~/Documents/theory_assignment/proj5$ ./client
--- 스레드풀 파라미터 한계 검증 ---
pthread_pool_init(): 일꾼 스레드 최대 수 초과.....PASSED
pthread_pool_init(): 대기열 최대 용량 초과.....PASSED
--- 스레드풀 초기화와 종료 검증 ---
pthread_pool_init(): 완료.....PASSED
pthread_pool_shutdown(): 완료.....PASSED
pthread_pool_init(): 완료.....PASSED
pthread_pool_shutdown(): 완료.....PASSED
--- 스레드풀 기본 동작 검증 ---
<0><1><12><13><14><15><16><17><18><19><20><21><22><23><24><25><26><27><28><29><
30><31><32><33><34><35><36><37><38><39><40><41><42><43><44><45><46><47><48><49><
50><51><52><53><54><55><56><57><58><59><60><61><62><63><2><3><4><5><6><7><8><9
><10><11>[0][1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16][17][18][19]
[20][21][22][23][24][25][26][27][28][29][30][31][32][33][34][35][36][37][38][39]
[40][41][42][43][44][45][46][47][48][49][50][51][52][53][54][55][56][57][58][5
9].....PASSED
```

스레드풀을 초기화하는 `pthread_pool_init` 함수에 일꾼 스레드와 대기열의 최대 수, 용량을 초과했

을 때 POOL_FAIL을 제대로 return한 결과를 보인다. 또한 초기화 함수 init과 종료 함수 shutdown이 POOL_COMPLETE, POOL_DISCARD 종료 옵션에 맞게 제대로 종료된 모습을 보인다. 그 후 대기열이 차서 기다리지 않고 거절된 작업을 빨간색으로 출력하는 기본 동작을 제대로 검증한 모습이다.

```

--- 스레드풀 종료 방식 검증 ---
[T0]1152921500311879687
[T0]1152921500311879759
[T1]1152921500311879789
[T1]1152921500311879841
[T1]1152921500311879853
[T2]1152921500311879979
소수 6개를 찾았다.
일부 일꾼 스레드가 구동되기 전에 풀이 종료되었을 가능성이 높다. 오류는 아니다.
스레드가 출력한 소수의 개수가 일치하는지 아래 값과 확인한다.....PASSED
T0(2), T1(3), T2(1), T3(1), T4(5), T5(3), T6(1), T7(2)

```

스레드 풀 종료 요청이 POOL_DISCARD 옵션으로 들어왔기 때문에 대기열에서 기다리던 스레드가 종료되고 소수를 6개 찾은 뒤 끝난 모습이다.

```

[T0]1152921500311879687
[T0]1152921500311879759
[T1]1152921500311879789
[T1]1152921500311879841
[T1]1152921500311879853
[T2]1152921500311879979
[T3]1152921500311880077
[T4]1152921500311880111
[T4]1152921500311880113
[T4]1152921500311880119
[T4]1152921500311880171
[T4]1152921500311880177
[T5]1152921500311880203
[T5]1152921500311880237
[T5]1152921500311880251
[T6]1152921500311880357
[T7]1152921500311880449
[T7]1152921500311880461
[T8]1152921500311880531
[T8]1152921500311880573
[T10]1152921500311880707
[T11]1152921500311880797
[T11]1152921500311880839
[T11]1152921500311880867
[T12]1152921500311880977
[T13]1152921500311881029
[T13]1152921500311881049
[T13]1152921500311881071
[T15]1152921500311881227
[T15]1152921500311881253
[T15]1152921500311881269
소수 31개를 모두 찾았다.
스레드가 출력한 소수의 개수가 일치하는지 아래 값과 확인한다.....PASSED
T0(2), T1(3), T2(1), T3(1), T4(5), T5(3), T6(1), T7(2)
T8(2), T9(0), T10(1), T11(3), T12(1), T13(3), T14(0), T15(3)

```

스레드풀 종료 요청이 POOL_COMPLETE 옵션으로 들어왔기 때문에 소수 31개를 모두 출력한 뒤 종료된 모습이다.

```

--- 무작위 검증 ---
{0}{1}{2}{3}..{4}{5}{6}{7}{8}{9}{10}..{11}..{12}{13}{14}{15}{16}{17}..{18}{19}{20}{21}{22}{23}{24}{
25}{26}..{27}{28}{29}{30}{31}{32}{33}{34}{35}{36}{37}{38}{39}..{40}{41}..{42}{43}{44}{45}{46}{47}..
{48}{49}{50}{51}{52}{53}{54}{55}..{56}{57}{58}{59}..{60}{61}..{62}{63}{64}..{65}{66}{67}{68}..{69}..
{70}{71}{72}{73}..{74}..{75}{76}{77}{78}..{79}{80}{81}..{82}{83}{84}{85}..{86}{87}{88}{89}{90}{91}..{
92}{93}..{94}{95}{96}{97}{98}{99}{100}{101}{102}{103}..{104}{105}..{106}{107}..{108}{109}..{110}{111}
{112}{113}..{114}{115}{116}..{117}..{118}{119}{120}{121}..{122}{123}{124}{125}{126}..{127}{128}{129}{
130}{131}{132}{133}{134}..{135}{136}..{137}..{138}..{139}{140}{141}..{142}..{143}{144}{145}{146}..{147}
{148}{149}{150}{151}{152}{153}{154}{155}..{156}{157}..{158}{159}{160}{161}{162}{163}{164}..{165}{
166}{167}{168}..{169}{170}{171}..{172}{173}{174}{175}{176}{177}{178}{179}..{180}..{181}{182}..{183}
{184}..{185}{186}..{187}{188}{189}{190}..{191}..{192}..{193}..{194}{195}{196}{197}{198}..{199}{200}{2
01}{202}{203}{204}{205}{206}..{207}..{208}{209}{210}{211}{212}{213}{214}{215}{216}{217}{218}..{219}
{220}{221}..{222}{223}{224}..{225}{226}{227}..{228}{229}{230}{231}..{232}{233}{234}..{235}..{236}{237}
{238}{239}{240}..{241}{242}{243}{244}{245}{246}{247}..{248}{249}{250}..{251}{252}..{253}{254}..{2
55}{256}{257}{258}{259}{260}{261}{262}{263}{264}..{265}{266}{267}{268}{269}{270}..{271}{272}{273}{
274}{275}..{276}{277}..{278}{279}{280}..{281}{282}{283}..{284}..{285}{286}..{287}{288}..{289}{290}..{2
91}..{292}{293}{294}..{295}{296}{297}{298}{299}..{300}{301}{302}..{303}{304}{305}..{306}..{307}{308}..
{309}{310}{311}{312}{313}{314}..{315}{316}{317}..{318}{319}..{320}{321}{322}{323}{324}{325}{326}{
327}{328}{329}..{330}{331}{332}{333}{334}{335}{336}..{337}..{338}{339}{340}{341}{342}{343}{344}..{3
45}{346}{347}..{348}{349}..{350}{351}{352}{353}{354}{355}..{356}{357}..{358}{359}{360}..{361}{362}..{3
63}..{364}..{365}..{366}..{367}..{368}..{369}..{370}..{371}..{372}..{373}..{374}..{375}..{376}{377}{378}{37
9}{380}..{381}{382}{383}{384}{385}{386}{387}{388}{389}{390}{391}..{392}{393}{394}{395}{396}{397}{3
98}{399}{400}{401}{402}{403}..{404}..{405}..{406}..{407}..{408}..{409}{410}{411}..{412}{413}{414}{415}
{416}{417}{418}{419}{420}{421}{422}{423}{424}..{425}{426}..{427}{428}{429}{430}..{431}{432}..{433}{4
34}..{435}{436}..{437}..{438}{439}{440}..{441}{442}{443}{444}{445}{446}..{447}{448}{449}{450}{451}{45
2}{453}{454}{455}{456}{457}{458}{459}{460}..{461}{462}..{463}..{464}{465}..{466}{467}{468}..{469}..{47
0}{471}{472}..{473}..{474}..{475}{476}..{477}..{478}..{479}..{480}{481}..{482}{483}..{484}{485}{486}{4
87}{488}{489}..{490}{491}..{492}..{493}..{494}{495}..{496}{497}{498}..{499}..{500}..{501}..{502}{503}{
504}{505}{506}..{507}{508}{509}{510}{511}{512}{513}..{514}{515}{516}..{517}{518}{519}..{520}..{521}..{
522}{523}..{524}{525}..{526}..{527}..{528}..{529}{530}..{531}{532}{533}{534}..{535}..{536}{537}{538}{53
9}{540}{541}{542}{543}..{544}..{545}..{546}{547}{548}{549}..{550}{551}{552}{553}..{554}{555}{556}..{55
7}{558}{559}{560}{561}{562}{563}{564}..{565}{566}{567}..{568}{569}{570}{571}{572}{573}{574}{575}{5
76}{577}{578}{579}..{580}..{581}{582}..{583}{584}{585}{586}{587}{588}{589}..{590}{591}..{592}{593}{59
4}..{595}{596}{597}..{598}..{599}{600}{601}..{602}{603}{604}..{605}{606}{607}{608}..{609}..{610}{611}
{612}{613}{614}{615}{616}{617}{618}{619}{620}{621}..{622}{623}..{624}{625}{626}{627}..{628}{629}
{630}{631}{632}..{633}{634}..{635}..{636}{637}{638}..{639}{640}{641}..{642}{643}..{644}..{645}{646}{6
47}{648}{649}..{650}{651}{652}{653}{654}..{655}{656}{657}{658}..{659}{660}{661}{662}{663}..{664}..{6
65}{666}{667}{668}{669}{670}{671}{672}..{673}{674}{675}{676}{677}{678}{679}{680}..{681}{682}{683}{
684}..{685}..{686}{687}{688}..{689}..{690}..{691}{692}..{693}{694}{695}{696}{697}..{698}{699}{700}{70

```

```

1}..{702}{703}{704}{705}..{706}{707}{708}..{709}{710}{711}{712}{713}..{714}..{715}...{716}..{717}{718}
}..{719}{720}..{721}{722}{723}{724}{725}..{726}..{727}{728}..{729}{730}{731}{732}{733}..{734}{735}{73
6}{737}{738}..{739}{740}..{741}{742}{743}{744}{745}..{746}{747}{748}{749}{750}{751}..{752}..{753}..{75
4}..{755}{756}..{757}{758}{759}{760}{761}{762}{763}..{764}..{765}{766}{767}{768}{769}{770}{771}..{7
72}{773}{774}{775}{776}..{777}..{778}{779}..{780}{781}..{782}..{783}{784}{785}{786}..{787}..{788}..{78
9}{790}..{791}{792}{793}..{794}{795}{796}{797}..{798}..{799}{800}{801}{802}{803}..{804}..{805}..{806}
..{807}..{808}{809}{810}{811}{812}{813}{814}{815}{816}{817}..{818}..{819}{820}..{821}..{822}{823}{824}
}..{825}..{826}..{827}..{828}{829}{830}..{831}..{832}..{833}{834}..{835}{836}{837}..{838}{839}{840}{841}
{842}{843}{844}{845}{846}..{847}..{848}{849}..{850}..{851}..{852}{853}..{854}{855}{856}..{857}{858}{85
9}{860}{861}{862}{863}..{864}{865}..{866}{867}{868}{869}{870}{871}{872}..{873}{874}{875}..{876}{877}
..{878}..{879}..{880}{881}..{882}{883}..{884}..{885}..{886}{887}{888}{889}{890}{891}{892}{893}{894}{89
5}..{896}{897}{898}{899}{900}..{901}..{902}{903}..{904}..{905}{906}{907}{908}{909}{910}{911}..{912}..
{913}{914}{915}{916}..{917}..{918}{919}{920}..{921}{922}..{923}..{924}{925}..{926}..{927}..{928}..{929}{9
30}..{931}{932}..{933}{934}{935}{936}..{937}..{938}{939}..{940}{941}{942}{943}{944}{945}{946}{947}{9
48}{949}..{950}{951}..{952}..{953}..{954}{955}{956}..{957}{958}..{959}{960}..{961}{962}{963}..{964}{965
}..{966}{967}{968}..{969}{970}{971}{972}{973}..{974}{975}..{976}{977}{978}{979}{980}{981}{982}{983}{
984}{985}{986}{987}{988}{989}{990}..{991}{992}{993}{994}{995}{996}{997}{998}..{999}{1000}{1001}{10
02}{1003}..{1004}{1005}{1006}{1007}..{1008}..{1009}{1010}{1011}..{1012}..{1013}{1014}..{1015}..{1016}{1
017}{1018}..{1019}{1020}..{1021}{1022}{1023}.....PASSED
총 실행시간 : 5381.6518초

```

스레드풀이 잘 작동하는지 무작위로 생성해서 검증한다. 0부터 1023까지의 숫자가 랜덤한 dot과 함께 출력되는 것을 볼 수 있다. 총 실행시간은 5381.6518초이다.

4. 문제점과 느낀점

어려웠던 부분은 스레드풀 종료 방식이 두가지였던 점이였다. 종료 명령을 받았을 때, worker함수에서 바로 종료하느냐, 대기열에 있는 모든 작업을 실행한 뒤 종료하느냐에 따라 while문을 벗어나고 while문을 계속해서 반복하는지가 나뉘기 때문에 end_flag를 전역변수로 정의한 뒤 구분점 역할로 사용하였다. 그러나 무작위 검증을 하였을 때, 숫자가 1023까지 실행되지 못하는 결과를

얻게 되었다. 이 부분에 대해 교수님께 여쭙본 결과 전역변수로 flag를 설정하게 되면 스레드마다 다른 flag값을 가질 수도 있다는 조언을 얻을 수 있었다. 나는 flag를 POOL_COMPLETE로 설정하였지만 어떤 스레드는 다른 상태의 flag로 실행될 수 있다는 것이다. 교수님의 조언을 바탕으로 pthread_pool.h를 수정하여 pthread_pool_t 구조체에 bool running으로 사용하던 변수를 int running으로 바꿔주고, 기본 실행, POOL_COMPLETE, POOL_DISCARD 세 상태로 구분하였다. 이렇게 정의한 뒤 how에 따라 pool->running 값을 할당해 주었더니, 옳은 출력 결과를 얻을 수 있었다. 스레드를 관리할 때 가장 중요한 부분인 동기화를 놓친다면 큰 문제가 발생한다는 점을 알 수 있었고, 앞으로도 멀티스레드를 다루게 된다면 동기화 작업을 절대 잊지 말아야겠다고 다짐하게 된 계기가 되었다.