

# Operating System Project #4

## Thread Pool

소프트웨어학부 2018044502 노하준  
2018044766 안요한  
2018045223 한동연

### INDEX

1. 스레드풀 알고리즘
2. 컴파일 과정
3. 결과 분석
4. 문제점과 느낀점

## 1. 알고리즘

**\*\*기존 pthread\_pool\_t 구조체에 q\_rear변수를 추가하여 원형 버퍼를 보완하여 구현하였다\*\***

### worker

1. 받아온 인자값으로 pthread\_pool\_t구조체를 pool로 정의해준다.
2. while문을 통하여 스레드가 실행중인 동안 mutex\_lock을 걸어준다.
3. queue가 비었다면 full 조건변수에서 대기한다.
4. (여기서, 실행이 종료되면, lock을 풀고 스레드를 종료시킨다.)
5. submit함수의 실행으로 인하여 queue에 task가 채워지면 full 에서 대기중이던 스레드가 깨어난다.
6. 채워진 queue의 task가 실행 되었으니 q\_len의 값을 줄여주고 q\_front값을 늘려준다.
7. task실행으로 queue에 빈자리가 생겼으므로 대기중인 empty조건변수에 signal을 보내준다.
8. lock을 풀고 큐에 있던 task를 실행시킨 후 pool 이 종료 될때 까지 위 과정을 반복한다.

### pthread\_pool\_init

1. 스레드풀 구조체의 변수를 모두 초기화 시켜준다.
2. 대기 원형 버퍼인 q와 일별 스레드의 ID를 저장하는 배열인 bee 의 메모리를 queue\_size,bee\_size 만큼 할당해준다.
3. bee\_size 만큼 스레드를 생성하며 worker함수를 실행한다.

### pthread\_pool\_submit

1. mutex\_lock을 걸어주고 queue에 빈자리가 있으면 입력 받은 값 (f,p) 을 q[q\_rear] (끝부분)에 넣는다.
2. task가 하나 증가했으므로 q\_len과 q\_rear의 값을 하나 올려준다.
3. 만일 queue에 빈자리가 없고 POOL\_WAIT상태라면 queer가 비어있는지 다시한번 확인하여 queue에 빈자리가 생길 때까지 조건변수를 통해 기다려준다.
4. (여기서, 실행이 종료된다면 lock을 풀어주고 종료시킨다.)
5. worker함수의 실행으로 인해 queue에 빈자리가 생겼다면 대기중이었던 task를 채워준다.
6. queue에 task가 채워졌으므로 q\_len의 값을 증가시켜주고 대기중인 full 조건변수를 signal 보내 깨워준다.
7. q\_rear의 값을 증가시켜주고 lock을 풀어준 뒤 POOL\_SUCCESS를 return한다.

### pthread\_pool\_shutdown

1. 실행상태인 running을 false로 바꾼다.
2. full과 empty에서 대기중인 모든 스레드를 broadcast를 통해 깨워준다.
3. 각각의 스레드 들은 running 상태가 false 로 바뀌었으므로, 깨워진 스레드들을 lock 을 풀고 종료한다. (Woker , submit 함수 참고)
4. 스레드 수만큼 pthread\_join을 통하여 스레드의 종료를 기다린 뒤 종료시켜준다.
5. Malloc 으로 할당 했던 메모리를 해체한다.
6. 스레드의 종료가 실패하면 POOL\_FAIL을, 성공하면 POOL\_SUCCESS를 return한다.



- 초반 thread\_pool 의 init 과 shutdown 검증 -> passed
- pool1 의 실행결과, 함수 number1 을 submit 하고, 임계구역 밖에서 함수를 실행하므로 중복되어 실행.
- 그 후 8스레드, 16 버퍼를 가진 스레드 풀을 가동하여, face 함수 실행.
- pool1 , pool2 를 동시 가동하여 , number 1 , number2 함수 병렬 실행. Pool2 만 shutdown
- 그 후 랜덤 스레드 갯수, 랜덤 버퍼 사이즈를 가진 pool2, pool3, pool4 가동
- 이 와중에도 pool1 은 계속 가동.
- 가동된 3개의 스레드 풀에 의하여 dot 함수 실행.
- Pool1 은 계속 사용하되, 앞서 가동한 3개의 스레드 풀은 shutdown
- Pool1 을 shutdown 후 종료.

#### 4. 과제를 수행하며 경험한 문제점과 느낀점

가장 어려웠던점은 코드를 짜고 실행하는 과정에 있어서 deadlock 이 왜 발생하는지 그 지점을 찾는 것과, 정확한 논리의 전개였다. 사실 mutex\_lock이나 POSIX조건변수에 대한 사용법은 지난 프로젝트인 reader-writer problem을 통하여 익혔기 때문에 그렇게 어렵지 않았다. 하지만 코드자체의 구조가 복잡하여 스레드 실행순서와 진행상황을 명확히 파악하기 어려웠고 어떠한 이유때문에 deadlock 이 걸리는지 확인하기가 정말 쉽지 않았다. 이를 해결하기 위하여 printf를 찍어 오류 지점을 찾아내 디버깅 작업을 하며 코드를 작성하였다. 추가적으로 구조체 타입에 익숙하지 않아 개념을 바로잡고 이를 활용하는데 많은 시간이 들었다. 구조체 q 에 접근 하는 것도 애 먹었을 뿐더러 구조체 내에 함수와 파라미터를 집어 넣고 함수를 실행하지 않는 등 어설픈 실수를 반복하기도 하였다.

사실 이런점에서 많은 시행착오를 겪었지만 시행착오를 통해 조금 더 성장 해 나갈 수 있었던 것 같다. 결과적으로 이번 프로젝트를 통해 threadpool의 정의와 구현, 무엇보다도 이를 사용하는 이유에 대하여 명확히 이해하게 되었다.