EE415 project 1 report

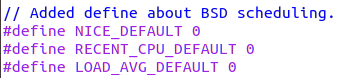
학번 : 20170538

이름 : 임승호

1.DATA STRUCTURE

**In thread.c**

아래 정의된 3가지는 BSD 스케줄링 할 때 initialize를 위하여 정의되었다.



load\_avg는 최근 1분간 수행가능한 평균 프로세스의 개수를 나타내는 Load average를 표시하기 위한 변수이다.

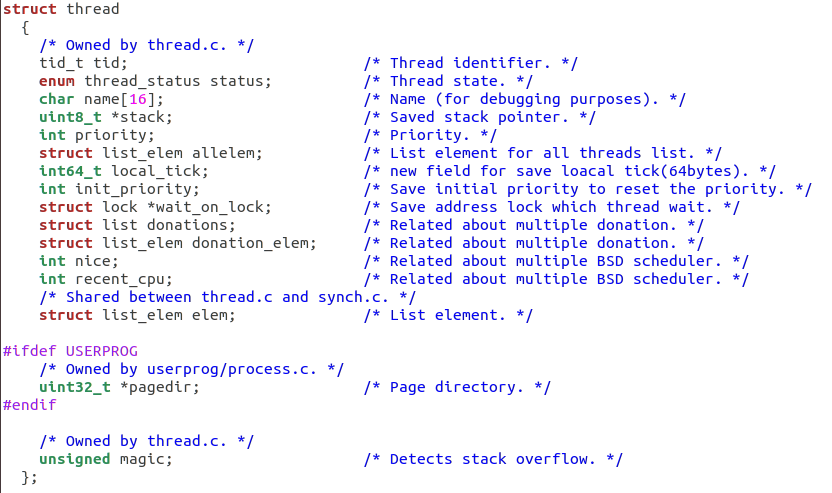


Sleep 상태의 스레드들을 관리하기 위하여 sleep list가 선언되었다.



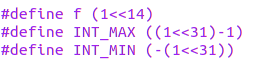
Sleep list에서 대기중인 스레드들의 local tick 값 중 최소값을 저장하는 변수이다.

**In thread.h**



thread 구조체에 thread가 깨어날 tick을 저장하기위한 local\_tick, 변하기전 초기 priority를 저장하는 init\_priority, 해당 스레드가 권한획득을 원하는 lock을 나타내는 wait\_on\_lock, multiple donation을 고려한 donations와 donation\_elem, BSD 스케줄러를 위한 nice,recent\_cpu가 새로 추가되었다.

**In fixed\_point.h**



17.14형식 고정소수점 숫자를 표현하기 위해 정의된 3가지의 상수이다.

f는 17.14형식 에서의 1 , INT\_MAX는 17.14형식에서의 최대정수, INT\_MIN은 17.14형식에서의 최소정수를 표현한다.

2. Algorithms

In timer.c

timer\_sleep() : 이 함수는 ticks를 인자로 받아 thread\_sleep()을 호출함으로써 ticks만큼 thread를 sleep 시키는 함수이다.

timer\_interrupt(): 이 함수는 기본적으로 호출되면 현재 시스템의 ticks을 하나 증가시키고 다음에 깨워야 할 시간인 tick을 get\_next\_tick()로 받아온 후 해당하는 thread를 thread\_wakeup()을 사용하여 깨운다. 만약 BSD 스케줄러를 사용한다면 매 호출마다 recent\_cpu를 증가시키고, 매 초마다 load\_avg,recent\_cpu,priority를 mlfqs\_load\_avg()와 mlfqs\_recalc()를 호출함으로써 재 계산한다.

또한 매 4tick 마다 mlfqs\_priority로 현재 thread의 priority를 재 계산한다.

In thread.c

thread\_yield() : 이 함수는 현재 thread를 ready state로 변경하는 역할을 한다. 또한 ready list에 현재 thread를 넣는데 cmp\_priority()를 이용하여 priority 순으로 삽입한다. 그 후에 schedule()로 context switch를 수행한다. 주로 스레드를 스케줄링할 필요가 생겼을 때 호출된다.

thread\_sleep() : 이 함수는 ticks를 인자로 받아 현재 thread의 깨울 시간을 저장한 후 sleep\_list에 넣은 후 thread\_block()을 통하여 sleep동작을 마무리한다.

update\_tick() : 이 함수는 ticks를 인자로 받아 next\_tick을 ticks로 설정한다.

get\_next\_tick() : 이 함수는 next\_tick을 반환한다.

thread\_wakup() : 이 함수는 ticks를 인자로 받아 sleep\_list에서 tick보다 작은 local\_tick을 가진 thread들을 sleep\_list에서 제거한 후 thread\_unblock을 사용함으로써 모두 ready state로 바꾸어 준다.

thread\_set\_priority() : 이 함수는 BSD 스케줄러가 아닐 때만 동작한다. 아닐 때의 동작은 인자로 받은 new\_priority를 현재 thread의 priority로 변경하고 바뀐 priority로 인한 새로운 priority donation을 고려하기위해 refresh\_priority를 이용하여 수행한다. 만약 donation 후의 priority가 원래의 priority보다 작다면 새로운 thread를 스케줄링 하기위해 test\_max\_priority()를 호출한다.

thread\_get\_priority() : 이 함수는 현재 thread의 priority를 반환한다.

thread\_set\_nice() : 이 함수는 현재 thread의 nice변수를 인자로 받은 nice로 변경한다. nice값이 바뀌었으므로 BSD 스케줄러의 priority 식에 따라 mlfqs\_priority()를 호출하여 priority를 재계산 한다. Priority가 바뀌었으므로 thread\_yield()의 일부분을 가져와서 스케줄링을 다시한다. Interrupt의 문제 때문에 yield를 직접 호출하지는 않았다.

thread\_get\_nice() : 이 함수는 현재 thread의 nice값을 반환하는 함수이다.

thread\_get\_load\_avg() : 이 함수는 현재 load\_avg값에 100을 곱하여 반환하는 함수이다. 고정 소수점 형식이므로 fixed\_point.h에 정의된 mult\_mixed() 함수를 이용하여 곱한 후 정수 반환을 위하여 fp\_to\_int\_round()로 return하였다.

thread\_get\_recent\_cpu() : 이 함수는 현재 recent\_cpu값에 100을 곱하여 반환하는 함수이다. 고정 소수점 형식이므로 fixed\_point.h에 정의된 mult\_mixed() 함수를 이용하여 곱한 후 정수 반환을 위하여 fp\_to\_int\_round()로 return하였다.

test\_max\_priority() : 이 함수는 현재 thread와 ready state인 모든 thread와 비교하여서 ready state thread의 priority가 더 크다면 그것을 thread\_yield()를 통해 스케줄링 해주는 함수이다. Ready\_list는 priority 순서로 정렬된 list임으로 list\_front로 첫번째 thread 만 비교해도 충분하다.

cmp\_priority() : 이 함수는 두개의 인자를 받아 각각에 해당하는 thread를 list\_entry를 통하여 얻은 후 첫번째 인자가 크다면 true 아니면 false를 반환하는 함수이다.

donate\_priority() : 이 함수는 현재 thread가 획득을 기다리는 lock의 holder인 thread가 현재 thread보다 priority가 낮다면 donation을 하는 함수이다. Nested donation을 고려하여 holder인 thread가 획득을 기다리는 lock의 holder인 thread에도 donation 하는 것을 계속 반복한다.

remove\_with\_lock() : 이 함수는 lock이 release 되었을 때 그 lock을 기다리는 모든 thread들을 donations list에서 없애는 함수이다.

refresh\_priority() : 이 함수는 현재 thread의 priority가 바뀌었을 때 호출되는 함수이다. 현재

thread의 priority에 init\_priority로 변경한후 donation list에서 가장 큰 priority와 비교하여 큰 값을 priority로 설정한다.

mlfqs\_priority() : 이 함수는 BSD 스케줄러에서 쓰이는 priority 공식에 의하여 인자로 받은 t의 priority를 다시 계산하는 함수이다,.

mlfqs\_recent\_cpu() : 이 함수는 BSD 스케줄러에서 쓰이는 recent\_cpu 공식에 의하여 인자로 받은 t의 recent\_cpu를 다시 계산하는 함수이다,.

mlfqs\_load\_avg() : 이 함수는 BSD 스케줄러에서 쓰이는 load\_avg 공식에 의하여 인자로 받은 t의 load\_avg를 다시 계산하는 함수이다. 단 load\_avg는 음수가 될 수없으므로 계산값이 음수가 나온다면 0으로 설정해준다.

mlfqs\_increment() : 이 함수는 매 tick마다 호출되는 함수이다. 현재 current의 recent\_cpu값을 1 증가시킨다.

mlfqs\_recalc() : 이 함수는 모든 thread들의 recent\_cpu와 priority를 mlfqs\_recent\_cpu(),

mlfqs\_priority()를 호출하여 다시 계산하는 함수이다.

count\_thread() : 이 함수는 thread들중 idle thread가 아니면서 ready state거나 running state인 thread의 개수를 구하는 함수이다. Load\_avg를 계산하기 위하여 필요한 값이다.

**In synch.c**

sema\_up() : 이 함수는 semaphore를 반환하고 value를 1 높이는 함수이다. 이 때 semaphore의 획득을 기다리는 thread들의 priority가 바뀌었을 수도 있으므로 waters리스트를 cmp\_priority()를 이용하여 priority 순서대로 정렬해준다. 후에 priority에 따라 스케줄링을 하기위해 thread\_yield()를 호출한다.

sema\_down() : 이 함수는 semaphore를 획득하고 value를 1 낮추는 함수이다. 이 때 semaphore가 남지않았다면 waiters list에 priority 순서대로 넣어주고 block 상태로 변경한다.

lock\_acquire() : 이 함수는 sema\_down()을 호출하여 현재 thread가 lock을 얻게 한다. 또한 BSD 스케줄러가 아닐 때만 lock을 얻기 전에 priority donation을 실행 한다.

lock\_release() : 이 함수는 sema\_up을 호출하여 현재 thread가 사용하는 lock을 방출한다. 또한 BSD스케줄러가 아닐 때만 remove\_with\_lock()과 refresh\_priority()를 호출하여 lock을 기다리는 thread들이 lock을 얻을 수 있는 상태로 만들어 준다.

cmp\_sem\_priority() : 이 함수는 인자로 받은 두개의 semaphore를 기다리고 있는 thread들 중 가장 큰 priority를 비교하여 첫번째 인자의 것이 크다면 true 아니면 false를 반환한다.

**In fixed\_point.h**

int int\_to\_fp() : 이 함수는 정수 n을 고정 소수점 17.14형식으로 변환하여 반환한다.

int fp\_to\_int() : 이 함수는 고정 소수점 17.14형식 x를 정수로 버림하여 반환한다.

int fp\_to\_int\_round() : 이 함수는 고정 소수점 17.14형식 x를 정수로 반올림하여 반환한다.

int add\_fp() : 이 함수는 고정 소수점 17.14형식인 두 수를 더한 후 반환한다.

int add\_mixed() : 이 함수는 고정 소수점 17.14형식 x와 정수 n을 더한 후 반환한다.

int sub\_fp() : 이 함수는 고정 소수점 17.14형식인 x에서 y를 뺀 후 반환한다.

int sub\_mixed() : 이 함수는 고정 소수점 17.14형식 x에서 정수 n을 뺀 후 반환한다.

int mult\_fp() : 이 함수는 고정 소수점 17.14형식인 두 수를 곱한 후 반환한다.

int mult\_mixed() : 이 함수는 고정 소수점 17.14형식 x와 정수 n을 곱한 후 반환한다.

int div\_fp() : 이 함수는 고정 소수점 17.14형식 x에서 y를 나눈 수를 반환한다.

int div\_mixed() : 이 함수는 고정 소수점 17.14형식 x에서 정수 n을 나눈 후 반환한다.

3. SYNCHRONIZATION

여러 thread가 한정된 자원을 공유하지 않도록 여러 함수를 사용하였다. 자원을 사용하기 위해 lock\_acquire()를 호출하면 sema\_down()을 통해 자원에 접근할 수 있게 해준다. 반대로 자원을 사용하고 나면 lock\_release()를 호출하여 sema\_up()을 통해 자원의 접근권한을 해제한다. 총 접근가능한 자원의 수는 semaphore 구조체의 member인 value안에 저장되어있고 sema\_down()을 통하여 이 값을 감소, sema\_up()을 통하여 이 값을 증가시킨다. sema\_down()은 남은 공유 자원이 있는지 확인 하여 있을 때만 thread에게 자원의 접근을 허락하고 sema\_up()은 lock의 방출 후에야 그 lock을 기다리고 있는 thread들을 접근할 수 있게 하기 때문에 확실한 SYNCHRONIZATION을 보장한다.

4. RATIONALE