#HW2

김태우 & 원상혁

1. 설명
2. Alarm System Call
   1. 과제 목표

Alarm이란 호출한 프로세스를 정해진 시간 후 다시 시작하는 커널 내부 함수로, 핀토스에서는 알람 기능이 \*Busy waiting을 이용하여 구현하며, 과제에서는 알람 기능을 sleep/wake up을 이용하여 다시 구현 한다.

* + - Busy waiting : 쓰레드가 CPU를 점유하면서 대기하는 상태로 자원 낭비와 소모 전력이 불필요하게 낭비됨.
  1. 개념

Sleep/Wake up 기반으로 구현.

Sleep된 쓰레드를 저장하는 Sleep Queue를 사용하고, timer\_sleep 함수를 호출한 쓰레드를 저장한다.

Timer\_sleep함수 호출시 해당 쓰레드를 ready\_list에서 제거하고 sleep queue에 추가해준다. Timer interrupt 발생 시 tick체크하여 시간이 다 된 쓰레드는 sleep queue 삭제 후 ready\_list에 추가해주어 wake up를 수행한다.

* 1. 함수 구현
     + void thread\_sleep ( int64\_t ticks )

실행 중인 스레드를 재우는 함수로 intr\_disable함수를 사용해 인터럽트를 받아들이지 않았고, idle\_thread가 아닌 경우 현재 쓰레드의 tick값을 인자로 받아온 ticks값으로 변경후 sleep\_list에 해당 쓰레드를 추가해주었습니다.

그 후 update\_next\_tick\_to\_awake 함수를 호출하여 가장 작은 tick을 갖도록 해주었습니다. 그 후 해당 쓰레드를 블락시킨 후 intr\_enable 함수를 사용해 인터럽트 허용 후 함수 종료하도록 구현했습니다.

* + - void thread\_awake ( int64\_t ticks )

sleep\_list에 있는 쓰레드를 순회하면서 wakeup\_tick값이 인자로 받은 ticks보다 크거나 같은 쓰레드를 sleep\_list에서 제거하는 함수로, 위와 같은 조건인 경우 list\_remove함수를 사용하여 sleep\_list에서 제거해주었고, 해당 쓰레드의 블록을 풀어주도록 구현했고. 위와 같은 조건이 아닌 경우 update\_next\_tick\_to\_awake함수를 호출하여 가장 작은 tick을 갖도록 해주고 함수 종료하도록 구현했습니다.

* + - void update\_next\_tick\_to\_awake ( int64\_t ticks )

next\_tick\_to\_awake가 ready\_list의 쓰레드 중 가장 작은 tick을 갖도록 해주는 함수로 next\_tick\_to\_awake와 인자로 받은 ticks를 비교해 ticks < next\_tick\_to\_awake 인 경우 next\_tick\_to\_awake값에 ticks를 넣어주도록 구현했습니다.

* + - int64\_t get\_next\_tick\_to\_awake ( void )

next\_tick\_to\_awake 값을 반환해주는 함수로 반환해주도록 구현했습니다.

1. Priority Scheduling
   1. 과제 목표

현재 핀토스에선 라운드 로빈으로 구현되어 있는 스케줄러를 우선순위를 고려하여 스케줄링 하도록 다시 구현한다.

* 1. 개념

새로운 쓰레드의 우선순위가 현재 실행중인 쓰레드의 우선순위보다 높은 경우, 새로운 쓰레드가 실행중인 쓰레드를 선점하게 된다.

PRI\_MIN(=0) – PRI\_MAX(=63)의 범위를 가지며 숫자가 클수록 우선순위가 높다.

Ready\_list역시 우선순위로 정렬해야 한다.

* 1. 함수 구현
     + void test\_max\_priority ( void )

함수에 들어오면 intr\_disable 함수를 사용하여 인터럽트가 일어나지 않도록 해주었고, ready\_list가 우선순위로 정렬했다고 가정하고 첫번째 쓰레드와 현재쓰레드의 우선순위를 비교하고, 현재 쓰래드의 우선순위가 더 작다면 thread\_yield함수를 사용하여 양보해주고, intr\_enable함수를 사용하여 인터럽트 허용 후 함수 종료하도록 구현했습니다.

* + - bool cmp\_priority ( const struct list\_elem \*e, const struct list\_elem \*b, void \*aux UNUES )

첫번째 인자의 쓰레드와 두번째 인자의 쓰레드의 우선순위 비교를 하여 첫 번째가 높으면 1을, 두 번째가 높으면 0을 반환하도록 구현했습니다.

* + - void thread\_unblock ( struct thread \*t )
    - void thread\_yield ( void )

함수가 호출 되면 기존 ready\_list에 맨 뒤에 넣었던 방식을 list\_insert\_ordered함수를 사용하여 우선순위 순으로 정렬되어 쓰레드가 ready\_list에 삽입되도록 수정 구현했습니다.

1. Multi-Level Feedback Queue Scheduler (MLFQ)
   1. 과제 목표

앞선 과제에서 우선순위 스케줄러를 구현하였음으로 MLFQ 스케줄러의 나머지 3가지 규칙( 아래 개념 참고 ) 을 만족시킬 수 있도록 하여 MLFQ 스케줄러를 구현한다.

* 1. 개념

MLFQ 스케줄러는 짧은 작업을 먼저 실행시켜 반환시간을 최적화하고, 응답시간을 최적화하고자 하는 스케줄러이다.

우선순위를 지닌 여러 큐를 사용하여 스케줄링을 하며, 5가지 규칙이 있다.

1. 우선순위(A) > 우선순위(B) 인 경우, A 실행(O) , B 실행 (X)
2. 우선순위(A) = 우선순위(B) , RR 방식으로 실행
3. 작업이 시스템에 들어가면 최상위 큐에 배치
4. 작업이 지정된 단계에서 배정받은 시간을 소진하면 작업의 우선순위 감소
5. 일정 주기 S가 지난 후 시스템의 모든 작업은 최상위 큐로 이동

일정 주기 S는 부두 상수라고 불리며 적절한 값을 지정해주어야 한다.

* 1. 함수 구현
     + fixed\_point 연산 함수

fixed\_point 연산을 위해서 실습 자료에 나와있는 표를 보고 구현해주었습니다.

* + - void mlfqs\_priority ( struct thread \*t )

인자로 받아온 쓰레드가 idle\_thread가 아닌지 체크해주고, 실습 자료에 나와있는 계산식을 구현해주었고, priority는 정수이기 때문에 fp\_to\_int 함수를 사용하여 정수로 변환 후 인자로 받아온 쓰레드의 우선순위에 넣어주도록 구현해주었습니다.

* + - void mlfqs\_recent\_cpu ( struct thread \*t )

인자로 받아온 쓰레드가 idle\_thread가 아닌지 체크해주고, 실습 자료에 나와있는 계산식을 구현해주었고, recent\_cpu값은 실수이기에 정수 변환 없이 인자로 받아온 쓰레드의 recent\_cpu값에 넣어주도록 구현했습니다.

* + - void mlfqs\_load\_avg ( void )

실습 자료에 나와있는 계산식을 구현해주었고, load\_avg는 0보다 작을 수 없기 때문에 계산식 후에 작아진 경우 0으로 설정해주었습니다. load\_avg는 실수이기에 전역변수로 설정한 load\_avg값에 계산한 값을 넣어주도록 구현했습니다.

* + - void mlfqs\_increment ( void )

현재 쓰레드가 idle\_thread가 아닌지 체크해주고, add\_mixed함수를 사용하여 현재 쓰레드의 recent\_cpu값을 1 증가시키도록 구현했습니다.

* + - void mlfqs\_recalc ( void )

list\_entry매크로를 사용하여 all\_list의 모든 쓰레드의 recent\_cpu 와 priority값을 재계산하도록 각각의 쓰레드마다 mlfqs\_recent\_cpu함수와 mlfqs\_priority함수를 호출하여 재계산해주었고, mlfqs\_load\_avg함수를 호출하여 load\_avg역시 재계산해주도록 구현했습니다.

* + - void thread\_set\_nice ( int nice UNUSED)

intr\_disable함수를 사용하여 함수 내에서 인터럽트가 일어나지 않게 해주었고, 현재 쓰레드의 nice값을 인자로 받아온 nice값으로 변경 후 test\_max\_priority함수를 사용하여 우선순위 재계산과 우선순위 스케줄링을 해주도록 구현했습니다. intr\_enable 함수를 사용하여 인터럽트 허용 후 종료하게 구현했습니다.

* + - int thread\_get\_nice ( void )

intr\_disable 함수를 사용하여 함수 내에서 인터럽트가 일어나지 않게 해주었고, 현재 쓰레드의 nice값을 반환하도록 구현해주었습니다. 반환 전 intr\_enable 함수를 사용하여 인터럽트 허용 후 종료하게 구현했습니다.

* + - int thread\_get\_load\_avg ( void )
    - int thread\_get\_recent\_cpu ( void )

intr\_disable 함수를 사용하여 함수 내에서 인터럽트가 일어나지 않게 해주었고, mult\_mixed함수를 사용하여 load\_avg( recent\_cpu )에 100을 곱해주고, fp\_to\_int\_round함수를 사용하여 반올림후에 load\_avg( recent\_cpu )를 반환해주도록 구현하였고, 반환 전 intr\_enable함수를 사용하여 인터럽트 허용 후 종료하게 구현했습니다.

* + - static void timer\_interrupt ( struct intr\_frame \*args UNUSED)

thread\_mlfqs가 true인 경우 해당 함수 호출 시 mlfqs\_increment함수를 호출하여 recent\_cpu를 1증가시켜 주고, 매 1초(tick = 100) 마다 mlfqs\_recalc함수를 사용하여 load\_avg, recent\_cpu, priority 를 계산해주었고, 매 4tick 마다 mlfqs\_priority함수를 사용하여 prirority를 계산해도록 구현했습니다.

1. Make Check 결과

