Project 2

Report

2017314331

김도엽

Report 2

0. Project1 PDF에 주어진 조건에 부합하는 CFS Scheduler를 xv6에 추가하는 작업을 거쳤습니다. 이하 내용은 추가된 Code, Code에 대한 설명, 그리고 Result 순으로 작성될 예정입니다.

1 . 추가된 Code

1-1 proc.h에 추가된 structure proc 내의 변수

```
  uint vruntime;
  int weight;
  int timeslice;
  int rtime; // Process runtime
  uint total_ticks;
```

1-2 proc.c에 추가 및 변경된 Code

1-2.1 fork( ) 함수에 추가된 Code

```
//…
  np->vruntime = curproc->vruntime
  np->weight = curproc->weight
  np->timeslice = 0;

  //…
```

fork 함수가 호출될 시에, 부모 Process의 vruntime과 weight 등을 상속받습니다.

1-2.2 update_weight() 함수 추가

```
void update_weight(struct proc *p) {
  p->weight = weights[p->nice];
}
```

어떤 Process가 할당된 timeslice를 전부 소진하였을 경우, trap.c에서 nice 값의 재조정을 통해 priority가 바뀌게 되는데, 그때의 변경된 nice에 해당하는 weight 값을 할당하기 위한 함수입니다.

1-2.3 wakeup1( ) 함수의 변경 code

```
static void
wakeup1(void *chan)
{
  struct proc *p;
  int runnable_exists = 0;

  // Check if there is any process in RUNNABLE state
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
    if (p->state == RUNNABLE) {
      runnable_exists = 1;
      break;
    }
  }

  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
    if (p->state == SLEEPING && p->chan == chan) {
      // Set the vruntime of the process to be woken up to 0 if no process in
RUNNABLE state
      if (!runnable_exists) {
        p->vruntime = 0;
      } else {
        p->vruntime += (1000 * 1024) / p->weight;
      }
      p->state = RUNNABLE;
    }
  }
}
```

PDF에 설명되어 있는 바, Process가 woken 되었을 때 virtual runtime의 값을 할당하는 규칙을 수정하였습니다. RUNNABLE 상태의 process가 존재하지 않을 시에는 process의 virtual runtime을 '0'으로 setting 하였습니다.

1-2.4 ps ( ) 함수의 변경 code.

```
void
ps(int pid)
{
  struct proc *p;
  int name_section = 0;

  acquire(&ptable.lock);
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
    if (pid == 0 || p->pid == pid) {
      if (!name_section) {
        cprintf("name\tpid\tstate\t\tpriority\truntime/weight\truntime\t\tvrun
time\t\ttick %d\n", ticks * 1000);
        name_section = 1;
      }
      if (p->state == UNUSED)
        continue;
      cprintf("%s\t%d\t", p->name, p->pid);

      switch (p->state) {
        case SLEEPING:
          cprintf("SLEEPING\t");
          break;
        case RUNNABLE:
          cprintf("RUNNABLE\t");
          break;
        case RUNNING:
          cprintf("RUNNING \t");
          break;
        case ZOMBIE:
          cprintf("ZOMBIE  \t");
          break;
        default:
          cprintf("UNKNOWN \t");
          break;
      }
      cprintf("%d\t\t", p->nice);
      cprintf("%d\t\t", p->rtime / p->weight);
      cprintf("%d\t\t", p->rtime);
      cprintf("%d\n", p->vruntime);
    }
  }
  release(&ptable.lock);
}
```

Project2에서 요구하는 Format으로 결과물이 출력되게 코드를 수정하였습니다.

1-2.5 scheduler( ) 함수의 CFS Scheduling implementation.

```
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;

  for(;;){
    // Loop over process table looking for process to run.
    sti();

    int total_weight = 0;
    struct proc *selected = 0;

    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
        if(p->state == RUNNABLE)
            total_weight += p->weight;


    }

    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      total_weight += p->weight;
      if(selected == 0 || p->vruntime < selected->vruntime){
        selected = p;
      }
      if(selected != 0){

        c->proc = selected;
        selected->timeslice = (10* 1000 * selected->weight) / total_weight;

        switchuvm(selected);

        selected->state = RUNNING;
        swtch(&(c->scheduler), selected->context);
        switchkvm();

        // update vruntime after the process is done running
        selected->vruntime += (selected->rtime * 1024) / selected->weight;

      }
```

```
    }
    release(&ptable.lock);
  }
}
```

CFS Scheduling 의 규칙에 따라, 상대적으로 높은 priority 를 가진 process 를 찾아, select 한 다음 이를 context switch 하여 먼저 실행되게 한 후 실행된 runtime 을 정해진 식에 따라 weight 로 나누어 virtual runtime 을 계산하여 해당 process 의 vruntime field 에 할당하였습니다.

1-2.6 allocproc( ) 함수의 변경된 code

```
static struct proc*
allocproc(void)
{
  ///… basic written codes … ///
  p->weight = weights[p->nice];
  return p;
}
```

Process 를 시작하면서, 해당 프로세스가 지니고 있는 field 의 정보에 기본 nice 에 대한 weight 를 할당하는 코드를 추가하였습니다.

1-3. trap.c 의 변경 Code

```
if(myproc() && myproc()->state == RUNNING &&
    tf->trapno == T_IRQ0+IRQ_TIMER) {

    myproc()->rtime+=1000; // Increment the process runtime

    if(myproc()->rtime >= myproc()->timeslice) {
      // Update the nice value before yielding
      myproc()->nice++;
      if (myproc()->nice > 39) {
        myproc()->nice = 39;
      }
      update_weight(myproc());
```

```
    yield();

  }
}
```

trap.c 에서 xv6 의 clock 에 따라 1tick 마다 trap 이 실행되어 실행 중인 process 의 상태를 확인하는 part 입니다. 확인한 process 가 실행될 때, 해당 process 가 RUNNING 인 상태 임을 확인할 때마다 rtime 값을 1000 을 더합니다. prcess 의 rtime field 의 갱신을 통해 실제 runtime 정보를 process 가 가지고 table 로 복귀합니다.

또한, PDF 에는 명시되어 있지 않지만 CFS Scheduling 의 규칙으로 process 가 할당받은 timeslice 를 모두 사용하여 실행을 마치면 yield() 함수 호출 전에 priority 가 후순위로 밀려 다른 process 보다 낮은 우선순위를 가지게되는 세부사항을 nice 값의 1 상승과 weight 값 갱신을 통해 적용하였습니다.

전체 결과

```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Student ID : 2017314331
Name : DoYeop Kim
==========xv6 Revised by DoYeop=========
$ ps
name    pid     state           priority        runtime/weight  runtime         vruntime        tick 328000
init    1       SLEEPING        20              9               10000           20000
sh      2       SLEEPING        20              0               1000            1000
ps      3       RUNNING         20              0               1000            0
$ ps
name    pid     state           priority        runtime/weight  runtime         vruntime        tick 965000
init    1       SLEEPING        20              9               10000           20000
sh      2       SLEEPING        20              4               5000            5000
ps      4       RUNNING         20              2               3000            0
```