OS PA_2 report

소프트웨어학과 2018314827 차승일

Abstract

바꾼 파일의 리스트는 아래와 같다.

```
02:53:57  |base|jet981217@jet981217-Z690-AORUS-ELITE-AX-DDR4 xv6-public ±|dev ✗|→ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   my_test.c
        modified:   proc.c
        modified:   proc.h
        modified:   trap.c
```

proc.h

```
37  // Per-process state
38  struct proc {
39    uint sz;                     //
40    pde_t* pgdir;                //
41    char *kstack;                //
42    enum procstate state;        //
43    int pid;                     //
44    struct proc *parent;         //
45    struct trapframe *tf;        //
46    struct context *context;     //
47    void *chan;                  //
48    int killed;                  //
49    struct file *ofile[NOFILE];  //
50    struct inode *cwd;           //
51    char name[16];               //
52    int nice;
53    unsigned int vruntime;
54    unsigned int runtime;
55    unsigned int cur_runtime;
56    unsigned int overflow_times;
57    unsigned int time_slice;
58  };
59
```

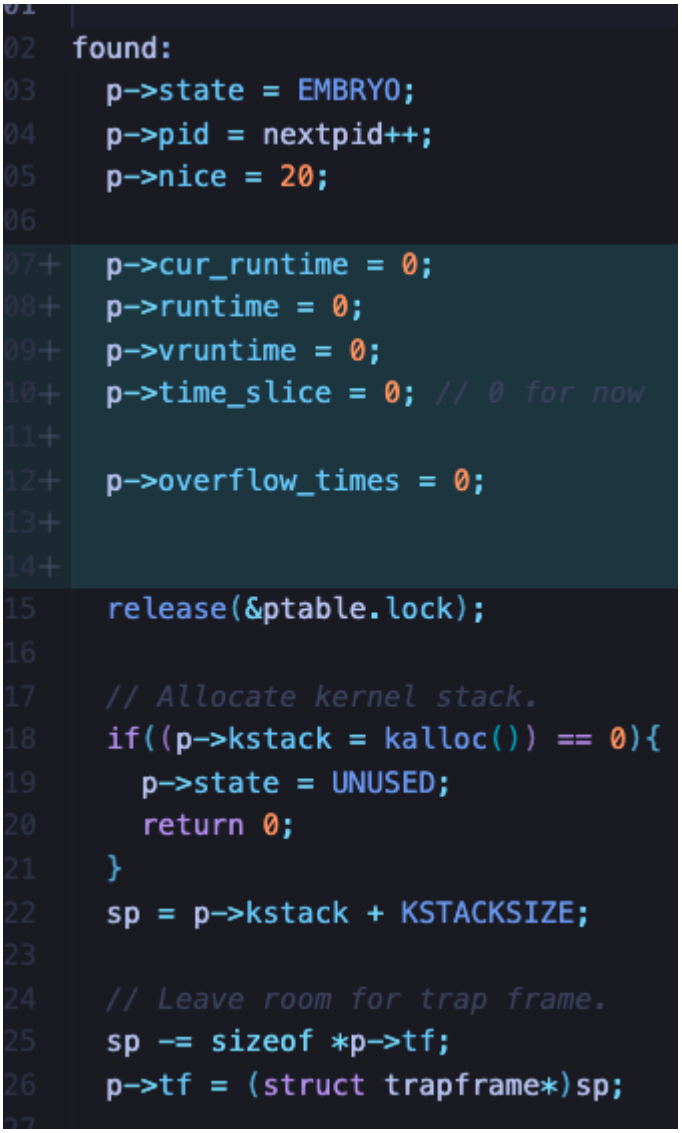
각 프로세스 구조체마다 vruntime, runtime을 나타내는 uint를 정의해준다.
또한 각 프로세스의 timeslice도 정의하고(mili-tick 단위이므로 그냥 uint로 정의), 현재 어느만큼의 mili-tick 동안 running(스케쥴러에 의해 돌아갔는지)했는지 나타내는 cur_runtime 도 정의해준다.

또한 vruntime 의 오버플로우도 핸들링 해야하므로 unsigned int의 범위를 넘어서는 횟수를 기록하는 overflow_times라는 unsigned int 변수를 정의해주자.

proc.c

```
10+ int weight_table[40] =
11+ {
12+   88761,  71755,  56483,  46273,  36291,
13+   29154,  23254,  18705,  14949,  11916,
14+   9548,   7620,   6100,   4904,   3906,
15+   3121,   2501,   1991,   1586,   1277,
16+   1024,   820,    655,    526,    423,
17+   335,    272,    215,    172,    137,
18+   110,    87,     70,     56,     45,
19+   36,     29,     23,     18,     15
20+ };
21+
22+ unsigned int total_weight = 0;
23+
```

weight table을 하드코딩 해주었다.
또한 time slice를 연산할 때 runnable process의 weight의 합이 필요하기 때문에 uint로 정의해주자.



```
found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->nice = 20;

+ p->cur_runtime = 0;
+ p->runtime = 0;
+ p->vruntime = 0;
+ p->time_slice = 0; // 0 for now
+
+ p->overflow_times = 0;
+
+
  release(&ptable.lock);

  // Allocate kernel stack.
  if((p->kstack = kalloc()) == 0){
    p->state = UNUSED;
    return 0;
  }
  sp = p->kstack + KSTACKSIZE;

  // Leave room for trap frame.
  sp -= sizeof *p->tf;
  p->tf = (struct trapframe*)sp;
```

새 프로세스를 만들었을 때 아까 정의해주었던 요소들에 대해서도 초기화를 해주자.

```
203  int
204  fork(void)
205  {
206    int i, pid;
207    struct proc *np;
208    struct proc *curproc = myproc();
209
210    // Allocate process.
211    if((np = allocproc()) == 0){
212      return -1;
213    }
214
215    // Copy process state from proc.
216    if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
217      kfree(np->kstack);
218      np->kstack = 0;
219      np->state = UNUSED;
220      return -1;
221    }
222    np->sz = curproc->sz;
223    np->parent = curproc;
224+
225+   np->nice = curproc->nice;
226+   np->overflow_times = curproc->overflow_times;
227+   np->vruntime = curproc->vruntime;
228+
229    *np->tf = *curproc->tf;
```

자식 프로세스를 fork 하였을 때, 조교님께서 QA에서 말씀하신대로 부모의 nice값을 따라가도록 하였고. 부모 vruntime 과 overflow 횟수를 복사하였다(overflow 횟수를 복사한 이유: 저것 또한 vruntime으로 추후 해석할 예정이라)

```
350 void
351 scheduler(void)
352 {
353   struct proc *p;
354   struct cpu *c = mycpu();
355   c->proc = 0;
356
357   for(;;){
358     // Enable interrupts on this processor.
359     sti();
360
361+    struct proc *min_vrun_proc = 0;
362+    int is_there_min_proc = 0;
363+    unsigned int min_vruntime = (unsigned int) -1;
364+    unsigned int min_overflow_times = (unsigned int) -1;
365+
366     acquire(&ptable.lock);
367+
368+    total_weight = 0;
369+
370+    // Get total weight of the process
371     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
372+      if(p->state == RUNNABLE)
373+        total_weight += weight_table[p->nice];
374+    }
375+
376+    // Set time slice for every process
377+    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
378+      if(p->state == RUNNABLE)
379+        p->time_slice = 1000*10*weight_table[p->nice]/total_weight;
380+    }
381+
382+    // choose process to run(with min vruntime)
383+    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
384+      if (p->state == RUNNABLE){
385+        if (p->vruntime == 0 && p->overflow_times == 0)
386+        {
387+          is_there_min_proc = 1;
388+          min_vrun_proc = p;
389+          break;
390+        }
```

```
        if (
          p->overflow_times < min_overflow_times ||
          (p->vruntime < min_vruntime && p->overflow_times == min_overflow_times)
        )
        {
          is_there_min_proc = 1;
          min_vruntime = p->vruntime;
          min_overflow_times = p->overflow_times;
          min_vrun_proc = p;
        }
      }

    // Switch to chosen process.  It is the process's job
    // to release ptable.lock and then reacquire it
    // before jumping back to us.
    if (!is_there_min_proc){
      release(&ptable.lock);
      continue;
    }

    c->proc = min_vrun_proc;
    min_vrun_proc->state = RUNNING;
    switchuvm(min_vrun_proc);

    swtch(&(c->scheduler), min_vrun_proc->context);
    switchkvm();

    // Process is done running for now.
    // It should have changed its p->state before coming back.
    c->proc = 0;

    release(&ptable.lock);

  }
}
```

대망의 스케쥴러이다. 위의 코드는 전체 코드이고 세부 사안은 밑에서 자세히 설명하겠다.

```
350  void
351  scheduler(void)
352  {
353    struct proc *p;
354    struct cpu *c = mycpu();
355    c->proc = 0;
356
357    for(;;){
358      // Enable interrupts on this processor.
359      sti();
360
361+     struct proc *min_vrun_proc = 0;
362+     int is_there_min_proc = 0;
363+     unsigned int min_vruntime = (unsigned int) -1;
364+     unsigned int min_overflow_times = (unsigned int) -1;
365+
366      acquire(&ptable.lock);
367+
368+     total_weight = 0;
369+
370+     // Get total weight of the process
```

for 문 내부가 다음 프로세스를 고르는 부분이다.

*min_vrun_proc 로 추후에 최소의 vruntime을 가지는 프로세스를 찾았을 때 그 프로세스를 가르킬 포인터를 저장하는 값을 초기화 해주었다.

is_there_min_proc 는 다음으로 실행할 최소 vruntime을 가지는 프로세스가 존재하느냐를 나타내는 변수이다. 1이면 있다는 것이고 0이면 없다는 것

min_vruntime, min_overflow_times는 최소 vruntime/overflow_times를 담고있는 변수이다. 이 변수를 이용해 프로세스들을 서치하며 최소 프로세스를 찾는 것인데, 첨에 초기화 할때는 11111.....111, 즉 unsigned int가 가질 수 있는 최대 수를 채워준다.

또한 위에서 정의한 total_weight를 매 프로세스 선택 단계마다 0으로 초기화해준다.

```
371      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
372+       if(p->state == RUNNABLE)
373+         total_weight += weight_table[p->nice];
374+     }
```

그 다음, ptable을 뒤져보며 runnable 한 프로세스들의 weight들을 전부 더해 total_weight에 담아준다.

```
377+    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
378+      if(p->state == RUNNABLE)
379+        p->time_slice = 1000*10*weight_table[p->nice]/total_weight;
380+    }
```

그다음 runnable 프로세스들에 대해 time_slice 를 계산하여 넣어준다. 밀리틱이므로 1000을 곱해준다. 식은 아래와 같다.

$$time\,slice = 10tick \times \frac{weight\,of\,current\,process}{total\,weight\,of\,runnable\,processes}$$

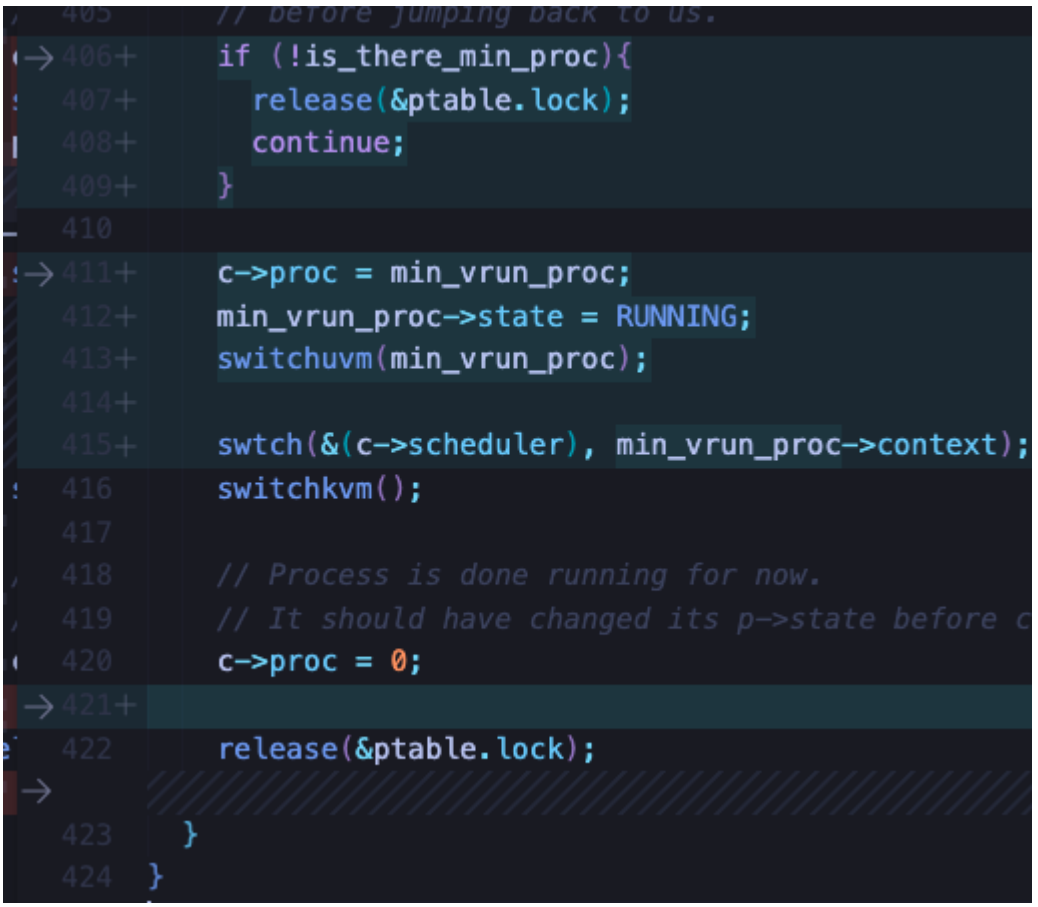
자 이제 ptable을 서치하며 최소 vruntime을 갖는 프로세스를 찾자.

```
382+    // choose process to run(with min vruntime)
383+    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
384+      if (p->state == RUNNABLE){
385+        if (p->vruntime == 0 && p->overflow_times == 0)
386+        {
387+          is_there_min_proc = 1;
388+          min_vrun_proc = p;
389+          break;
390+        }
391+        if (
392+          p->overflow_times < min_overflow_times ||
393+          (p->vruntime < min_vruntime && p->overflow_times == min_overflow_times)
394+        )
395+        {
396+          is_there_min_proc = 1;
397+          min_vruntime = p->vruntime;
398+          min_overflow_times = p->overflow_times;
399+          min_vrun_proc = p;
400+        }
401+      }
```

아래 설명은 전부 ptable의 process들중 runnable인 것들에만 해당된다.

조교님과의 QA에서, vruntime이 0인 프로세스가 여러개인 경우에는 들어온 순서대로 고르면 된다고 하셨으므로, vruntime과 overflow_times가 둘다 0인 프로세스를 찾으면 바로 그 프로세스를 min_vrun_proc로 가르키고, is_there_min_proc를 1로 한 후 break 한다. 현재 프로세스를 선택했으므로 ptable을 더뒤져보지 않으므로(물론 이 선택한 프로세스에서 다음 프로세스로 넘어갈 때에는 찾아보지만, 그건 위의 전체 for문에서 한번 더 돌아가므로 상관 없음)

그게 아닌 경우 먼저 min_overflow_times와 enumerate 중인 ptable의 멤버 프로세스의 overflow_times를 비교해, 만약 멤버 프로세스의 값이 작으면 vruntime을 볼것도 없이 무조건 이게 최소이므로 min_vrun_proc를 가르키고 min_vruntime, min_overflw_times을 이 프로세스의 것들로 값을 할당한다. 또한 is_there_min_proc도 1로 마킹해준다. 또한 min_overflow_times 이 ptable멤버 프로세스와 같은 경우에도 min_vruntime과 p->vruntime을 비교해 p의 것이 작다면 똑같이 한다. 이 다음에 ptable의 나머지 것들도 뒤져보며 최소 vruntime(당연히 overflow_times도 고려) process를 찾는다.



위의 과정에서 최소 vruntime 프로세스를 못 찾은 경우(더 정확히 말하자면 이 경우는 runnable이 없을 경우에만 해당됨), 제일 위의 for문으로 다시 돌아가도록 ptable의 락킹을 풀어주고 continue해준다.

아니라면 기존의 스케쥴링 코드대로 context switching을 해준다.

그다음 wakeup1을 수정한 부분을 보겠다.

```
525  static void
526  wakeup1(void *chan)
527  {
528    struct proc *p;
529+   // 고친부분
530+
531+   struct proc *min_vrun_proc = 0;
532+   int is_there_runable = 0; // runable 존재하는지
533+   unsigned int min_vruntime = (unsigned int) -1;
534+   unsigned int min_overflow_times = (unsigned int) -1;
535+
536+   // choose process min process(with min vruntime)
537+   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
538+     if (p->state == RUNNABLE){
539+       if (p->vruntime == 0 && p->overflow_times == 0)
540+       {
541+         is_there_runable = 1;
542+         min_vrun_proc = p;
543+         break;
544+       }
545+       if (
546+         p->overflow_times < min_overflow_times ||
547+         (p->vruntime < min_vruntime && p->overflow_times == min_overflow_times)
548+       )
549+       {
550+         is_there_runable = 1;
551+         min_vruntime = p->vruntime;
552+         min_overflow_times = p->overflow_times;
553+         min_vrun_proc = p;
554+       }
555+     }
```

```
558   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
559     if(p->state == SLEEPING && p->chan == chan)
560+    {
561       p->state = RUNNABLE;
562+      if(is_there_runable)
563+      {
564+        unsigned int one_tick_minus = 1000*1024/weight_table[myproc()->nice];
565+
566+        if(min_vrun_proc->vruntime >= one_tick_minus)
567+        {
568+          p->overflow_times = min_vrun_proc->overflow_times;
569+          p->vruntime = min_vrun_proc->vruntime - one_tick_minus;
570+        }
571+        else if(min_vrun_proc->overflow_times)
572+        {
573+          p->overflow_times = min_vrun_proc->overflow_times - 1;
574+          p->vruntime = ((unsigned int) -1) - one_tick_minus + min_vrun_proc->vruntime;
575+        }
576+        else
577+        {
578+          p->overflow_times = 0;
579+          p->vruntime = 0;
580+        }
581+      }
582+      else
583+      {
584+        p->overflow_times = 0;
585+        p->vruntime = 0;
586+      }
587+    }
588 }
```

전체 코드는 위와 같고 자세한 설명은 아래에 하겠다.(wakeup1을 부르기 전/후에 lock을 핸들하는게 wakeup함수에 구현되어 있으므로 여기에선 락킹을 신경쓰지 않았다)

```
526  wakeup1(void *chan)
527  {
528    struct proc *p;
529+  // 고친부분
530+
531+  struct proc *min_vrun_proc = 0;
532+  int is_there_runable = 0; // runable 존재하는지
533+  unsigned int min_vruntime = (unsigned int) -1;
534+  unsigned int min_overflow_times = (unsigned int) -1;
535+
536+  // choose process min process(with min vruntime)
537+  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
538+    if (p->state == RUNNABLE){
539+      if (p->vruntime == 0 && p->overflow_times == 0)
540+      {
541+        is_there_runable = 1;
542+        min_vrun_proc = p;
543+        break;
544+      }
545+      if (
546+        p->overflow_times < min_overflow_times ||
547+        (p->vruntime < min_vruntime && p->overflow_times == min_overflow_times)
548+      )
549+      {
550+        is_there_runable = 1;
551+        min_vruntime = p->vruntime;
552+        min_overflow_times = p->overflow_times;
553+        min_vrun_proc = p;
554+      }
555+    }
556+
```

스케쥴러에서와 똑같은 방식으로 제일작은 vruntime(overflow_times도 고려) 프로세스를 찾으므로 설명은 생략하겠다. 유일하게 다른 점은 is_there_min_proc가 is_there_runnable로 이름이 달라진 점이다.

```
558     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
559       if(p->state == SLEEPING && p->chan == chan)
560+      {
561         p->state = RUNNABLE;
562+        if(is_there_runable)
563+        {
564+          unsigned int one_tick_minus = 1000*1024/weight_table[myproc()->nice];
565+
566+          if(min_vrun_proc->vruntime >= one_tick_minus)
567+          {
568+            p->overflow_times = min_vrun_proc->overflow_times;
569+            p->vruntime = min_vrun_proc->vruntime - one_tick_minus;
570+          }
571+          else if(min_vrun_proc->overflow_times)
572+          {
573+            p->overflow_times = min_vrun_proc->overflow_times - 1;
574+            p->vruntime = ((unsigned int) -1) - one_tick_minus + min_vrun_proc->vruntime;
575+          }
576+          else
577+          {
578+            p->overflow_times = 0;
579+            p->vruntime = 0;
580+          }
581+        }
582+        else
583+        {
584+          p->overflow_times = 0;
585+          p->vruntime = 0;
586+        }
587+      }
588   }
```

그다음, 깨울 sleeping 프로세스를 찾아, runnable로 바꾸어준다. 그다음, 이전에(이 프로세스를 runnable로 바꾸기 전 기준) runnable이 있냐 없냐(is_there_runnble)에 따라 달라진다.

1) runnable이 있는 경우

현재 실행중인 프로세스 기준 1000 mili-tick(1tick)의 vruntime을 계산해 one_tick_minus에 저장한다.

그 후 overflow_time를 고려해 뺄셈을 한다.

1. min_vrun_proc의 vruntime이 1000 mili tick의 vruntime(overflow times 고려하지 않고 그냥 나머지)보다 큰 경우 그냥 vruntime으로 뺄셈해서 할당.
2. min_vrun_proc의 vruntime이 1000 밀리틱의 vruntime보다 작은 경우에는 overflow_times에서 쪼개서 뺄셈 vruntime과 같이 이용해 뺄셈. 각 overflow_times는 1111….1111이다.(100000000..0000아님!)
3. min_vrun_proc의 vruntime이 1000 밀리틱의 vruntime보다 작으면서 overflow_times도 0인 경우에는 그냥 전부 0으로 초기화 해서 줌.

2) 없는 경우

1. 전부 0으로 초기화 해서 줌.

아래는 kill 부분 수정 부분인데, wakeup1 부분과 완전히 똑같으므로 설명을 생략하겠다.

```
602 int
603 kill(int pid)
604 {
605   struct proc *p;
606
607   acquire(&ptable.lock);
608+
609+  struct proc *min_vrun_proc = 0;
610+  int is_there_runable = 0; // runable 존재하는지
611+  unsigned int min_vruntime = (unsigned int) -1;
612+  unsigned int min_overflow_times = (unsigned int) -1;
613+
614+  // choose process min process(with min vruntime)
615+  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
616+    if (p->state == RUNNABLE){
617+      if (p->vruntime == 0 && p->overflow_times == 0)
618+      {
619+        is_there_runable = 1;
620+        min_vrun_proc = p;
621+        break;
622+      }
623+      if (
624+        p->overflow_times < min_overflow_times ||
625+        (p->vruntime < min_vruntime && p->overflow_times == min_overflow_times)
626+      )
627+      {
628+        is_there_runable = 1;
629+        min_vruntime = p->vruntime;
630+        min_overflow_times = p->overflow_times;
631+        min_vrun_proc = p;
632+      }
633+    }
634+
```

```
635   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
636     if(p->pid == pid){
637       p->killed = 1;
638       // Wake process from sleep if necessary.
639       if(p->state == SLEEPING)
640+      {
641         p->state = RUNNABLE;
642+        if(is_there_runable)
643+        {
644+          unsigned int one_tick_minus = 1000*1024/weight_table[myproc()->nice];
645+
646+          if(min_vrun_proc->vruntime >= one_tick_minus)
647+          {
648+            p->overflow_times = min_vrun_proc->overflow_times;
649+            p->vruntime = min_vrun_proc->vruntime - one_tick_minus;
650+          }
651+          else if(min_vrun_proc->overflow_times)
652+          {
653+            p->overflow_times = min_vrun_proc->overflow_times - 1;
654+            p->vruntime = ((unsigned int) -1) - one_tick_minus + min_vrun_proc->vruntime;
655+          }
656+          else
657+          {
658+            p->overflow_times = 0;
659+            p->vruntime = 0;
660+          }
661+        }
662+        else
663+        {
664+          p->overflow_times = 0;
665+          p->vruntime = 0;
666+        }
667+      }
668       release(&ptable.lock);
669       return 0;
670     }
671   }
672   release(&ptable.lock);
673   return -1;
674 }
```

그 다음은 ps 와 관련된 함수들이다.

```
763+ void itoiarr(unsigned int num, int *iarr) {
764+     int i = 0;
765+     do {
766+         iarr[i] = (num % 10);
767+         i += 1;
768+         num = num / 10;
769+     } while (
770+       num > 0
771+     );
772+
773+     for (int j = i; j<20; j++)
774+     {
775+       iarr[j] = 0;
776+     }
777+
778+ }
```

unsigned int를 길이 20짜리 배열로 하나하나 나타내는 함수이다.
예를들어, 5002315이란 수가 있으면 이 함수에 넣으면 [5, 1, 3, 2, 0, 0, 5, 0……, 0]을 리턴한다.
vruntime을 overflow_times를 고려해 출력할 때(마치 오버플로우가 발생하지 않은 것처럼 핸들링한 것을 ps로 출력할 때) 이용할 예정.

```
780+ void print_int(
781+   int num,
782+   int max_len
783+ )
784+ {
785+   cprintf("%d", num);
786+
787+   int num_len = 0;
788+   if (num)
789+   {
790+     while (num != 0) {
791+       num = num / 10;
792+       num_len++;
793+     }
794+   }
795+   else num_len = 1;
796+
797+   for(int left=0; left < max_len - num_len; left++)
798+   {
799+     cprintf(" ");
800+   }
801+ }
```

```
803+ void print_unsigned(
804+   unsigned int num,
805+   int max_len
806+ )
807+ {
808+   cprintf("%d", num);
809+
810+   int num_len = 0;
811+   if (num)
812+   {
813+     while (num != 0) {
814+       num = num / 10;
815+       num_len++;
816+     }
817+   }
818+   else num_len = 1;
819+
820+   for(int left=0; left < max_len - num_len; left++)
821+   {
822+     cprintf(" ");
823+   }
824+ }
```

```
826+ void print_string(
827+   char* string,
828+   int max_len
829+ )
830+ {
831+   cprintf("%s", string);
832+
833+   int idx = 0;
834+
835+   while (string[idx] != '\0') {
836+     idx++;
837+   }
838+
839+   for(int left=0; left < max_len - idx; left++)
840+   {
841+     cprintf(" ");
842+   }
843+ }
```

ps에서 출력할 때, 정렬을 하기 위해 각각의 string, int, uint마다 cprintf로 패딩을 추가하여(즉 정해진 길이 단위로 출력하게 만드는)함수를 정의하였다.

그 다음은 vruntime을 출력하는 함수를 보이겠다.

```
void print_vruntime(
  unsigned int overflow_num,
  unsigned int vruntime_num
){
  if (overflow_num){
    int output_number[20] = {0};

    int carry = 0;
    // overflow part
    for (int overflow_idx = 0; overflow_idx<overflow_num; overflow_idx++){
      int to_iarr[20];
      unsigned int overflowed = (unsigned int) -1;
      itoiarr(overflowed, to_iarr);

      for (int idx = 0; idx < 20; idx++){
        int result = to_iarr[idx] + output_number[idx] + carry;
        output_number[idx] = (
          result
        ) % 10;
        if (result >= 10) carry = 1;
        else carry = 0;
      }
    }
    // Leftover part
    int to_iarr[20];
    itoiarr(vruntime_num, to_iarr);

    carry = 0;

    for (int idx = 0; idx < 20; idx++){
      int result = to_iarr[idx] + output_number[idx] + carry;
      output_number[idx] = (
        result
      ) % 10;
      if (result >= 10) carry = 1;
      else carry = 0;
    }

    int start = 0;
```

```
884+
885+     for (int idx = 19; idx >= 0; idx--){
886+       if (!start && output_number[idx] != 0){
887+         start = 1;
888+       }
889+       if (start){
890+         cprintf("%d", output_number[idx]);
891+       }
892+     }
893+   }
894+   else{
895+     cprintf("%d", vruntime_num);
896+   }
897+   cprintf("\n");
898+ }
899+
900+
```

하나하나 자세히 설명하겠다.

```
845+ void print_vruntime(
846+   unsigned int overflow_num,
847+   unsigned int vruntime_num
848+ ){
849+   if (overflow_num){
850+     int output_number[20] = {0};
851+
852+     int carry = 0;
853+     // overflow part
854+     for (int overflow_idx = 0; overflow_idx<overflow_num; overflow_idx++){
855+       int to_iarr[20];
856+       unsigned int overflowed = (unsigned int) -1;
857+       itoiarr(overflowed, to_iarr);
858+
859+       for (int idx = 0; idx < 20; idx++){
860+         int result = to_iarr[idx] + output_number[idx] + carry;
861+         output_number[idx] = (
862+           result
863+         ) % 10;
864+         if (result >= 10) carry = 1;
865+         else carry = 0;
866+       }
867+     }
```

먼저 overflow_num(overflow_times랑 같은 것이라 생각 하면 됨)이 0보다 큰 경우에는 output_number을 일단 길이 20짜리 배열로 정의해준다. 배열 인덱스 하나하나로 decimal을 나타낼 예정.

overflow_num은 overflow가 된 횟수, 즉 예를들어 overflow_num이 3이라면 1111….1111이 3번 있다는 것이다. 따라서 adder(decimal)을 만들어 output_number에 낮은 자릿수부터 더해서 output_number에 overflow_num만큼 1111…..1111을 더해준다.(10^0, 10^1 이 순서대로 하는 것이라 생각, itoiarr로 뒤집은 것을 더하는 거.)

```
    // Leftover part
    int to_iarr[20];
    itoiarr(vruntime_num, to_iarr);

    carry = 0;

    for (int idx = 0; idx < 20; idx++){
      int result = to_iarr[idx] + output_number[idx] + carry;
      output_number[idx] = (
        result
      ) % 10;
      if (result >= 10) carry = 1;
      else carry = 0;
    }

    int start = 0;

    for (int idx = 19; idx >= 0; idx--){
      if (!start && output_number[idx] != 0){
        start = 1;
      }
      if (start){
        cprintf("%d", output_number[idx]);
      }
    }
  }
```

그 다음 carry를 0으로 다시 만들어주고(임시 variable이므로) overflow_num이 아닌 vruntime_num(이건 횟수가 아니라 그냥 process의 vruntime p->vruntime이라 생각하면 됨. 즉 overflow된 횟수를 제외한 uint 범위 내의 나머지라 생각하면 됨) 을 위와 똑같은 과정으로 역순으로 뒤집은 배열을 adder(decimal) 을 만들어 output_number에 낮은 자릿수부터 더해서 output_number에 더해준다.

그 다음 최종 output_number을 역순으로 출력한다(처음에는 0이면 출력 안 하다가 0이 아닌것이 처음 나온 순간 전부 출력).

즉 예를들어, 한 프로세스의 overflow 횟수와, 나머지 vruntime 전부 고려해 계산한 총 vruntime이 123456789라면 output_number에는 [9,8,7,6,5,4,3,2,1,0,0,0,0 …., 0] 이 저장되고 역순으로(위의 설명한대로 0을 핸들링하여) 수를 출력함

```
  else{
    cprintf("%d", vruntime_num);
  }
  cprintf("\n");
}
```

overflow가 된 적이 없는 프로세스라면 그냥 출력한다.

```
901  void ps(int pid){
902+   char *states_by_idx[] = {"UNUSED", "EMBRYO", "SLEEPING", "RUNNABLE", "RUNNING", "ZOMBIE"};
903    struct proc *p;
904
905    acquire(&ptable.lock);
906+   print_string("name", 10);
907+   print_string("pid", 10);
908+   print_string("state", 20);
909+   print_string("priority", 10);
910+   print_string("runtime/weight", 20);
911+   print_string("runtime", 20);
912+   print_string("vruntime", 20);
913+   cprintf("tick %d\n", ticks*1000);
914+
915    if(pid){
916      for(
917        p = ptable.proc;
918        p <= &ptable.proc[NPROC];
919        p++
920      ){
921        if(p->pid == pid){
922          if(p->state != 0 && p->state <= 5){
923+           print_string(p->name, 10);
924+           print_int(p->pid, 10);
925+           print_string(states_by_idx[p->state], 20);
926+           print_int(p->nice, 20);
927+           print_unsigned(p->runtime/weight_table[p->nice], 20);
928+           print_unsigned(p->runtime, 20);
929+           print_vruntime(p->overflow_times, p->vruntime);
930          }
931          break;
932        }
933      }
934    }
```

```
935    else{
936      for(
937        p = ptable.proc;
938        p <= &ptable.proc[NPROC];
939        p++
940      ){
941        if(p->state != 0 && p->state <= 5 && p->pid >= 0){
942+         print_string(p->name, 10);
943+         print_int(p->pid, 10);
944+         print_string(states_by_idx[p->state], 20);
945+         print_int(p->nice, 10);
946+         print_unsigned(p->runtime/weight_table[p->nice], 20);
947+         print_unsigned(p->runtime, 20);
948+         print_vruntime(p->overflow_times, p->vruntime);
949        }
950      }
951    }
952
953    release(&ptable.lock);
954  }
955
```

ps로 출력할 때 위의 함수들을 이용해 잘 정렬할 수 있다. 10, 10, 20 등의 것들은 적당히 잘 align 하도록 맞춘 하이퍼 파라미터이다.

trap.c

Project 2. Implement CFS on xv6

- Implement CFS on xv6
  - Select process with minimum virtual runtime from runnable processes
  - Update runtime/vruntime for each timer interrupt

교안대로 매 타이머 인터럽트마다 **runtime/vruntime update**

```
17+ int weights[40] =
18+ {
19+   88761,  71755,  56483,  46273,  36291,
20+   29154,  23254,  18705,  14949,  11916,
21+   9548,   7620,   6100,   4904,   3906,
22+   3121,   2501,   1991,   1586,   1277,
23+   1024,   820,    655,    526,    423,
24+   335,    272,    215,    172,    137,
25+   110,    87,     70,     56,     45,
26+   36,     29,     23,     18,     15
27+ };
28+
```

weight를 하드코딩한다.

```
117+
118+  // Increase cur_running and running and vruntime every time slice
119    if(myproc() && myproc()->state == RUNNING &&
120       tf->trapno == T_IRQ0+IRQ_TIMER)
121+  {
122+    myproc()->cur_runtime += 1000;
123+    myproc()->runtime += 1000;
124+
125+    unsigned int result = 1000*1024/weights[myproc()->nice];
126+
127+    // Overflow detection
128+    if (4294967295 - myproc()->vruntime < result)
129+    {
130+      unsigned int limit = (unsigned int) -1;
131+
132+      unsigned int leftover = (unsigned int) result - (limit-(myproc()->vruntime));
133+      myproc()->overflow_times += 1;
134+      myproc()->vruntime = leftover;
135+    }
136+    else{
137+      myproc()->vruntime += (unsigned int) result;
138+    }
139+
140+    // Every tick -> check if time slice is done -> if done ? yeild and reset.
141+    if (myproc()->cur_runtime >= myproc()->time_slice)
142+    {
143+      myproc()->cur_runtime = 0;
144      yield();
145+    }
146+
147+  }
```

매 타이머 인터럽트마다 핸들링 하는 코드이다. 자세히 설명하겠다.

```
119    if(myproc() && myproc()->state == RUNNING &&
120       tf->trapno == T_IRQ0+IRQ_TIMER)
121+  {
122+    myproc()->cur_runtime += 1000;
123+    myproc()->runtime += 1000;
124+
125+    unsigned int result = 1000*1024/weights[myproc()->nice];
126+
```

이 부분이 매 타이머 인터럽트마다 불리는 부분에서 running 하는 프로세스에 대해, 해당 프로세스의 현재 스케쥴링 단계에서의 runtime(cur_runtime), 그리고 전체 runtime을 1000 더해주는 부분이다.

또한 result는 해당 프로세스의 1000 밀리틱(1틱) 단위 vruntime의 변화량이다. 뒤에 이 것을 vruntime에 더할 예정이다.

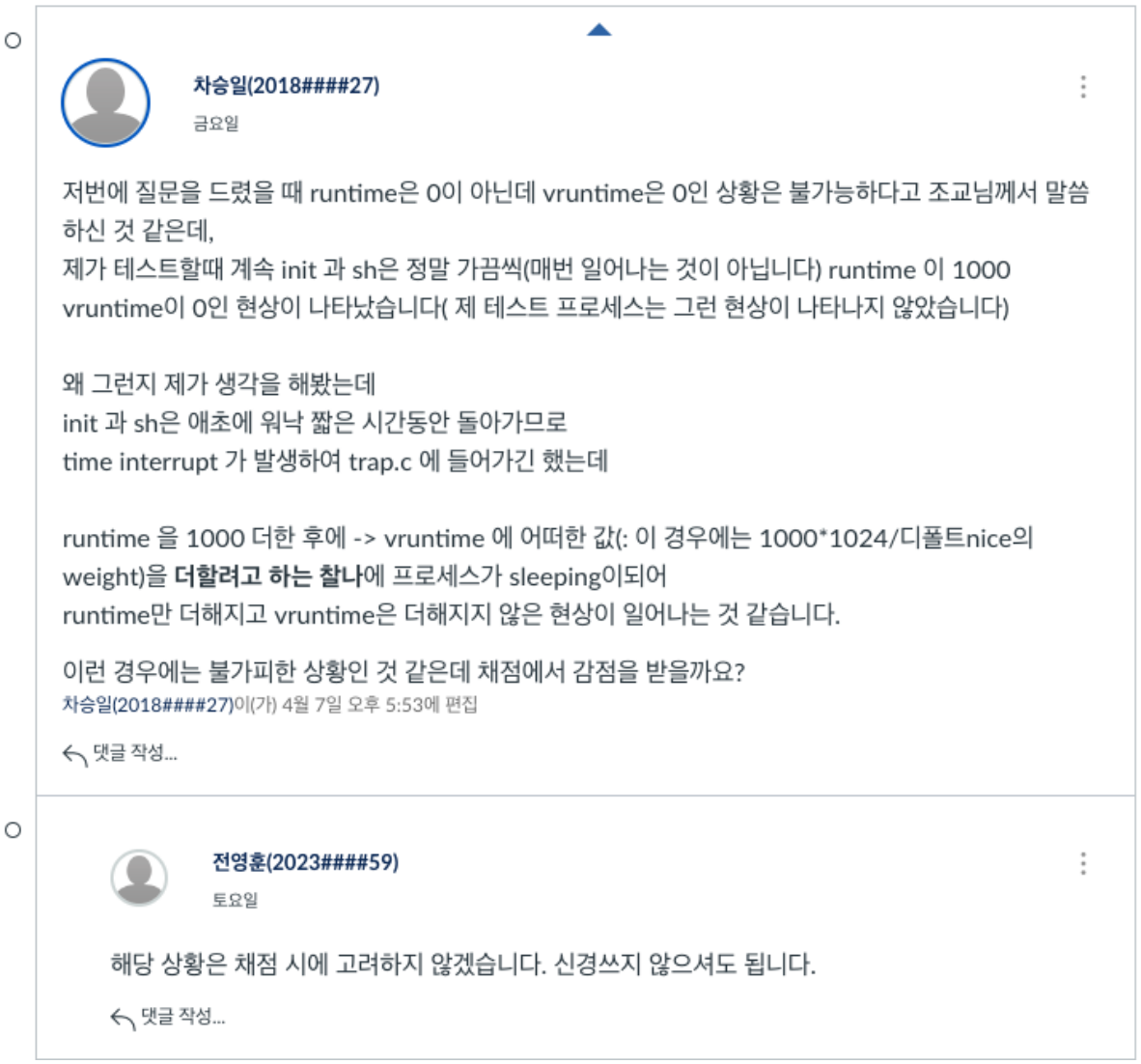
```
127+    // Overflow detection
128+    if (4294967295 - myproc()->vruntime < result)
129+    {
130+      unsigned int limit = (unsigned int) -1;
131+
132+      unsigned int leftover = (unsigned int) result - (limit-(myproc()->vruntime));
133+      myproc()->overflow_times += 1;
134+      myproc()->vruntime = leftover;
135+    }
136+    else{
137+      myproc()->vruntime += (unsigned int) result;
138+    }
```

4294967295는 unsigned 32bit int에서 최댓값이다. 즉 1111…..11111이다.
이 값을 이용해 4294967295 - myproc()->vruntime이 result보다 작은 경우, 즉 오버플로우가 일어날 예정인 경우, overflow_times를 1을 더하고 남은 값으로 vruntime을 설정하여 핸들링한다.

아닌 경우는 오버플로우가 일어나지 않은경우이므로 그냥 vruntime에 더해준다.

```
141+    if (myproc()->cur_runtime >= myproc()->time_slice)
142+    {
143+      myproc()->cur_runtime = 0;
144       yield();
145+    }
146+
147+  }
```

이 다음(위의 코드들과 같은 레벨에서 실행되는 time interrupt 조건문 내의 코드이다)에 cur_runtime, 즉 스케쥴링 된 후 돌아간 시간이 time_slice 이상인 경우 cur_runtime을 0으로 하고 yield해 스케쥴링을 다시 한다.



이 경우에는 고려하지 않는다고 하셨으므로, 따로 핸들링 하지 않았다.