CYDF216 Operating System – Fall 2018

Project: CFS Scheduler Simulation

2016320012 박종영

소스코드: https://github.com/simonxhockey/2018-2_OS_TermProject

데모 영상: https://youtu.be/M5XAxs1KieM

A. Programming Environment

Linux ubuntu 4.15.0-42-generic #45~16.04.1-Ubuntu SMP Mon Nov 19 13:02:27 UTC 2018 x86_64 GNU/Linux

gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609

B. Usage of Program

Program: cfs

Usage of cfs:

```
simonxhockey@ubuntu:~/2018-2_OS_TermProject$ ./cfs
1: Add process
2: Run
3: Turn off(process)
4: Turn off(scheduler)
5: Show Processes
```

>>1

<input nice value>

<input when the process completes execution>

```
1

what is nice value of the process?-5

How long the process running to finish?70000000
```

>>2

(...running...)

>>3

<give process id>

```
3

type which process to stop: 0
```

>>4

(...waiting to being stopped...)

>>5

(...can see the information of the processes on running queue...)

C. Screen Dumps and Implementation Details

```
list of processes
pid: 0 / start time: 1545278963 / running time: 0 / weight: 3121 / time slice: 10868253 / vruntime: 0
pid: 1 / start time: 1545278978 / running time: 0 / weight: 1024 / time slice: 3565873 / vruntime: 0
pid: 2 / start time: 1545278983 / running time: 0 / weight: 1024 / time slice: 3565873 / vruntime: 0
```

nice값이 -5인 process 하나와 nice값이 0인 process 둘을 rq에 올려둔 상태. 보다시피 weight값이 3121과 1024인 것을 알 수 있고, queue에 존재하는 process의 수보다 sched_nr_latency(8) 값이 더 크므로 periods = 18000000인 것을 확인할 수 있다(time slice 값들을 다 더해보면 된다.)

```
2
1: Add process
2: Run
3: Turn off(process)
4: Turn off(scheduler)
5: Show Processes
5

list of processes
pid: 0 / start time: 1545278963 / running time: 10868253 / weight: 3121 / time slice: 10868253 / vruntime: 3565873
pid: 1 / start time: 1545278978 / running time: 0 / weight: 1024 / time slice: 3565873 / vruntime: 0
pid: 2 / start time: 1545278983 / running time: 0 / weight: 1024 / time slice: 3565873 / vruntime: 0
```

한 번 실행했을 때 process0이 실행되었다.

```
pid: 0 / start time: 1545278963 / running time: 10868253 / weight: 3121 / time slice: 10868253 / vruntime: 3565873
pid: 1 / start time: 1545278978 / running time: 0 / weight: 1024 / time slice: 3565873 / vruntime: 0
pid: 2 / start time: 1545278983 / running time: 3565873 / weight: 1024 / time slice: 3565873 / vruntime: 3565873
```

한 번 더 실행해 process2가 실행되었다.

```
pid: 0 / start time: 1545278963 / running time: 10868253 / weight: 3121 / time slice: 10206758 / vruntime: 3565873
pid: 1 / start time: 1545278978 / running time: 0 / weight: 1024 / time slice: 3348837 / vruntime: 0
pid: 2 / start time: 1545278983 / running time: 3565873 / weight: 1024 / time slice: 3348837 / vruntime: 3565873
pid: 3 / start time: 1545279685 / running time: 0 / weight: 335 / time slice: 1095566 / vruntime: 0
```

새로운 process3를 queue에 올린 상태이다. 기존에 존재하던 process들의 vruntime값 중에서 가장 작은 0을 process3에게 채택되었음을 볼 수 있다.

```
pid: 0 / start time: 1545278963 / running time: 10868253 / weight: 3121 / time slice: 10206758 / vruntime: 3565873
pid: 1 / start time: 1545278978 / running time: 3348837 / weight: 1024 / time slice: 3348837 / vruntime: 3348837
pid: 2 / start time: 1545278983 / running time: 3565873 / weight: 1024 / time slice: 3348837 / vruntime: 3565873
pid: 3 / start time: 1545279685 / running time: 1095566 / weight: 335 / time slice: 1095566 / vruntime: 3348834
```

각 process마다 한 번씩 실행시켰더니 weight와 time slice의 차이가 있었지만 vruntime은 거의 비슷한 값을 가지고 있다. 이때 weight가 동일한 process1과 process2가 다른 vruntime을 가지는 것은, process1이 실행되지 않은 상태에서 process3이 들어와 각각의 time slice값이 갱신되었기 때문이다.

```
pid: 0 / start time: 1545278963 / running time: 10868253 / weight: 3121 / time slice: 595263 / vruntime: 3565873
pid: 1 / start time: 1545278978 / running time: 3348837 / weight: 1024 / time slice: 195305 / vruntime: 3348837
pid: 2 / start time: 1545278983 / running time: 3565873 / weight: 1024 / time slice: 195305 / vruntime: 3565873
pid: 3 / start time: 1545279685 / running time: 1095566 / weight: 335 / time slice: 63894 / vruntime: 3348834
pid: 4 / start time: 1545280007 / running time: 0 / weight: 88761 / time slice: 16929250 / vruntime: 3348834
pid: 5 / start time: 1545280023 / running time: 0 / weight: 110 / time slice: 20980 / vruntime: 3348834
```

(nice, life) = (-20, 30000000)인 process4와 (10, 30000000)인 process5를 queue에 추가했다. 이때 둘의 vruntime은 제일 작은 값인 3348834가 되었다.

```
2

Process 4 completes execution.
```

```
pid: 0 / start time: 1545278963 / running time: 10868253 / weight: 3121 / time slice: 10006768 / vruntime: 3565873
pid: 1 / start time: 1545278978 / running time: 3544142 / weight: 1024 / time slice: 3283220 / vruntime: 3544142
pid: 2 / start time: 1545278983 / running time: 3565873 / weight: 1024 / time slice: 3283220 / vruntime: 3565873
pid: 3 / start time: 1545279685 / running time: 1159460 / weight: 335 / time slice: 1074100 / vruntime: 3544139
pid: 5 / start time: 1545280023 / running time: 41960 / weight: 110 / time slice: 352689 / vruntime: 3739442
```

여러 차례의 running후에 process4가 자신의 일을 마무리하고 먼저 종료되었다.

```
>>> 10006768+3283220*2+1074100+352689
17999997
```

이때의 time slice값들을 다 더해주면 periods에 거의 일치하는 것을 알 수 있다. 즉, 하나의 process가 종료되면 남은 process들끼리 periods를 공유하도록 했다.

```
pid: 0 / start time: 1545278963 / running time: 10868253 / weight: 3121 / time slice: 1494159 / vruntime: 3565873
pid: 1 / start time: 1545278978 / running time: 3544142 / weight: 1024 / time slice: 490233 / vruntime: 3544142
pid: 2 / start time: 1545278983 / running time: 3565873 / weight: 1024 / time slice: 490233 / vruntime: 3565873
pid: 3 / start time: 1545279685 / running time: 1159460 / weight: 335 / time slice: 160379 / vruntime: 3544139
pid: 5 / start time: 1545280023 / running time: 41960 / weight: 110 / time slice: 52661 / vruntime: 3739442
pid: 6 / start time: 1545280448 / running time: 0 / weight: 36 / time slice: 17234 / vruntime: 3544139
pid: 7 / start time: 1545280458 / running time: 0 / weight: 29154 / time slice: 13957296 / vruntime: 3544139
pid: 8 / start time: 1545280467 / running time: 0 / weight: 9548 / time slice: 4571045 / vruntime: 3544139
pid: 9 / start time: 1545280477 / running time: 0 / weight: 1991 / time slice: 953178 / vruntime: 3544139
pid: 10 / start time: 1545280487 / running time: 0 / weight: 655 / time slice: 313577 / vruntime: 3544139
```

queue에 올라가있는 process를 10개로 만들었다.

```
>>> 1494159+490233*2+160379+52661+17234+13957296+4571045+953178+313577
22499995
```

해당 time slice값들을 다 더해봤더니 22500000에 매우 근사했다. 앞에서 언급했던 sched_nr_latency(8)보다 queue에 존재하는 process의 수가 더 커져서 periods 값을 sched_min_granularity(2250000) * num_rq(10)으로 갱신해주었다.

```
pid: 0 / start time: 1545278963 / running time: 13707155 / weight: 3121 / time slice: 1344743 / vruntime: 4497316
pid: 1 / start time: 1545278978 / running time: 4475585 / weight: 1024 / time slice: 441210 / vruntime: 4475585
pid: 2 / start time: 1545278983 / running time: 4497316 / weight: 1024 / time slice: 441210 / vruntime: 4497316
pid: 3 / start time: 1545279685 / running time: 1159460 / weight: 335 / time slice: 144341 / vruntime: 3544139
pid: 5 / start time: 1545280023 / running time: 142016 / weight: 110 / time slice: 47395 / vruntime: 4670872
pid: 6 / start time: 1545280448 / running time: 32745 / weight: 36 / time slice: 15511 / vruntime: 4475551
pid: 7 / start time: 1545280458 / running time: 26518862 / weight: 29154 / time slice: 12561566 / vruntime: 4475582
pid: 8 / start time: 1545280467 / running time: 8684986 / weight: 9548 / time slice: 4113941 / vruntime: 4475582
pid: 9 / start time: 1545280477 / running time: 1811038 / weight: 1991 / time slice: 857860 / vruntime: 4475581
pid: 10 / start time: 1545280487 / running time: 595796 / weight: 655 / time slice: 282219 / vruntime: 4475581
1: Add process
2: Run
3: Turn off(process)
4: Turn off(scheduler)
5: Show Processes
4

CFScheduler will stop in 3secs.
CFScheduler stopped.
simonxhockey@ubuntu:~/2018-2_OS_TermProject$
```

Turn off(scheduler):4를 실행해 scheduler를 성공적으로 종료시켰다.

```
pid: 0 / start time: 1545281035 / running time: 7473417 / weight: 820 / time slice: 7473417 / vruntime: 9332657
pid: 1 / start time: 1545281040 / running time: 7473417 / weight: 820 / time slice: 7473417 / vruntime: 9332657
pid: 2 / start time: 1545281045 / running time: 6106328 / weight: 335 / time slice: 3053164 / vruntime: 18665312
```

```
pid: 0 / start time: 1545281035 / running time: 7473417 / weight: 820 / time slice: 12779220 / vruntime: 9332657
pid: 2 / start time: 1545281045 / running time: 6106328 / weight: 335 / time slice: 5220779 / vruntime: 18665312
```

Turn off(process):3을 실행해 process1을 종료시킨 후의 상태다.

D. Code Works

```c
typedef struct sched_entity {
  int weight;
  int on_rq; // 0 is not on rq, 1 is on rq
  int now_running; // 1 is now running
  int time_slice;
  int vruntime;
}sched_entity;
```

```c
typedef struct process {
  int pid;
  int nice;
  int start_time; // use time()
  int running_time;
  int finish_time; // count for process finish
  struct sched_entity se;
}process;
```

```c
typedef struct node {
  int data; // pid
  int priority; // vruntime

  struct node* next;
}Node;
```

이렇게 3개의 구조체를 사용하였다.

process와 sched_entity는 process를 구현할 때 사용했고 node는 Priority Queue를 구현하기 위해 사용했다. process의 실행 순서를 정할 때 vruntime이 작은 것부터 실행되므로 우선순위-큐를 사용해서 구현했다.

```c
Node* newNode(int d, int p) {
  Node* tmp = (Node*)malloc(sizeof(Node));
  tmp->data = d;
  tmp->priority = p;
  tmp->next = NULL;

  return tmp;
}
```

```c
/* Return the value at head */
int peek(Node** head) {
  return (*head)->data;
}
```

```c
/* Removes the element by priority */
void pop(Node** head) {
  Node* tmp = *head;
  (*head) = (*head)->next;
  free(tmp);
}
```

```c
void Erase(struct process *pro) {
  pro->se.weight = 0;
  pro->se.on_rq = 0;
  pro->se.time_slice = 0;
  pro->se.vruntime = 0;
}
```

```c
/* Push node order to priority */
void push(Node** head, int d, int p) {
  Node* start = (*head);

  Node* tmp = newNode(d, p); // create new Node

  if ((*head)->priority > p) {  // if head's priority is less than new one's
    tmp->next = *head;
    (*head) = tmp;
  }
  else { // find a position for new one
    while (start->next != NULL && start->next->priority < p) {
      start = start->next;
    }
    tmp->next = start->next;
    start->next = tmp;
  }
}
```

main 함수 이외에 이렇게 다섯 개의 함수를 사용했다. newNode, peek, pop, push는 우선순위-큐를 관리하기 위해 사용했고, Erase는 종료되는 process를 정리할 때 사용했다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
```

stdlib.h - malloc(),

time.h - time(),

unist.h – sleep() 을 사용하기 위해 참조했다.

```c
struct process *pro = (process *)malloc(sizeof(struct process) * 50);

int i = 0, j, num_rq = 0, tot_weight = 0;
int sched_latency_ns = 18000000;
int sched_min_granularity = 2250000;
int sched_nr_latency = 8; /* sched_latency_ns/sched_min_granularity */
int periods, min_vruntime;
int what_is_running, del_pid;

Node *que = newNode(0,-5); // it will be poped after the first process node comes; -5 is smaller than 0(first vruntime)
```

구조체 포인터와 나중에 사용할 변수들을 선언하고, queue를 생성해주었다. 이때 priority값으로 -5를 준 것은 나중에 새로운 process가 push해준 후 queue생성에 사용했던 노드를 쉽게 제거하기 위함이다. priority값이 제일 작은 것이 head이므로 음수가 아닌 vruntime보다 작은 -5를 주면은 새로운 노드가 들어와도 첫 노드가 head가 되므로 삭제하기가 쉽다.

```c
103   while (1){ // every cycle is divided by user's input and user can fork or kill process per execution of one process
104     if (num_rq > 0) { // update periods by the number of processes on rq ...
109     }
110
111     printf("1: Add process\n2: Run\n3: Turn off(process)\n4: Turn off(scheduler)\n5: Show Processes\n");
112     int input;
113     scanf("%d", &input);
114
115     if (input == 1) { // Add process ...
152     }
153     else if (input == 2) { //Run ...
173     }
174     else if (input == 3) { // kill the process by user's input ...
183     }
184     else if (input == 4) { // kill the scheduler ...
190     }
191     else if (input == 5) { // display which processes are on rq and their attr. ...
200     }
201   }
```

프로그램의 cycle을 보면 while을 통해 계속 반복하도록 해두었다.

```c
104     if (num_rq > 0) { // update periods by the number of processes on rq
105       periods = (num_rq <= sched_nr_latency) ? sched_latency_ns : sched_min_granularity * num_rq;
106       for (j = 0; j < i; j++) { // update time slice per every cycle
107         pro[j].se.time_slice = (int)((double)periods * (double)pro[j].se.weight / (double)tot_weight);
108       }
109     }
```

rq에 안 비어있을 때 periods값과 process의 time slice값들을 갱신해주는 부분.

```c
printf("1: Add process\n2: Run\n3: Turn off(process)\n4: Turn off(scheduler)\n5: Show Processes\n");
int input;
scanf("%d", &input);
```

어떤 기능을 실행할지 input으로 정하는 부분.

```
115     if (input == 1) { // Add process
116       pro[i].pid = i;
117       printf("\nwhat is nice value of the process?");
118       scanf("%d", &pro[i].nice);
119       pro[i].start_time = (int)time(NULL);
120       pro[i].running_time = 0;
121       printf("\nHow long the process running to finish?");
122       scanf("%d", &pro[i].finish_time);
123       pro[i].se.weight = prio_to_weight[pro[i].nice + 20];
124       pro[i].se.on_rq = 1;
125       pro[i].se.now_running = 0;
126       tot_weight += pro[i].se.weight;
127       pro[i].se.time_slice = (int)((double)periods * (double)pro[i].se.weight / (double)tot_weight);
128 
129       if (num_rq == 0) {
130         pro[i].se.vruntime = 0; // no process in rq
131         push(&que, pro[i].pid, pro[i].se.vruntime); // the first node with pid and vruntime
132         pop(&que);
133       }
134       else { // find min_vruntime for new process
135         for (j = 0; j < i; j++) {
136           if (pro[j].se.on_rq == 1) {
137             break;
138           }
139         }
140         min_vruntime = pro[j].se.vruntime;
141         for (int k = j; k < i; k++){
142           if (pro[k].se.on_rq == 1) {
143             min_vruntime = (min_vruntime <= pro[k].se.vruntime) ? min_vruntime : pro[k].se.vruntime;
144           }
145         }
146         pro[i].se.vruntime = min_vruntime;
147         push(&que, pro[i].pid, pro[i].se.vruntime);
148       }
149 
150       i++;
151       num_rq++;
152     }
```

process를 새로 추가해줄 때 비어있는 구조체를 초기화 해주는 부분. rq에 첫 process를 추가할 때 먼저 넣어줬던 (0, -5)노드를 잊지 않고 pop해줘야 한다. 이 절에서 134-148줄의 내용은, 새로운 process를 rq에 넣어줄 때 기존의 vruntime 중 가장 작은 vruntime을 계산하는 부분이다.

```
153     else if (input == 2) { //Run
154       what_is_running = peek(&que);
155       pro[what_is_running].se.now_running = 1;
156       pro[what_is_running].se.vruntime += (int)((double)(pro[what_is_running].se.time_slice) * (double)(1024) / (double)(pro[what_is_running].se.weight));
157       pro[what_is_running].se.now_running = 0;
158       pro[what_is_running].running_time += pro[what_is_running].se.time_slice;
159 
160       /* To update the node's priority */
161       pop(&que);
162       push(&que, pro[what_is_running].pid, pro[what_is_running].se.vruntime);
163       if (pro[what_is_running].running_time >= pro[what_is_running].finish_time) { // a process completes its execution
164         printf("\nProcess %d completes execution.\n", pro[what_is_running].pid);
165         pro[what_is_running].se.on_rq = 0;
166         pop(&que);
167         tot_weight -= pro[what_is_running].se.weight;
168         Erase(&pro[what_is_running]);
169         num_rq--;
170       }
171 
172       what_is_running = 0;
173     }
```

155-156줄이 process가 running state를 유지할 때이다. 이후, priority queue 노드의 vruntime을 수정하기 위해 161-162줄이 존재한다. 만약 process가 목표했던 일을 다 끝낸 경우 rq와 priority queue에서 제거되는 부분이 163-170줄에 구현되었다.

```
174       else if (input == 3) { // kill the process by user's input
175         printf("\ntype which process to stop: ");
176         scanf("%d", &del_pid);
177
178         pro[del_pid].se.on_rq = 0;
179         pop(&que);
180         tot_weight -= pro[del_pid].se.weight;
181         Erase(&pro[del_pid]);
182         num_rq--;
183       }
```

종료하고자 하는 process의 pid값을 입력 받아 종료해주는 기능이 구현되었다.

```
184       else if (input == 4) { // kill the scheduler
185         printf("\nCFScheduler will stop in 3secs.\n");
186
187         sleep(3);
188         printf("CFScheduler stopped.\n");
189         break;
190       }
```

scheduler 종료 요청을 받고 종료해주는 기능이 구현되었다.

```
191     else if (input == 5) { // display which processes are on rq and their attr.
192       if (i > 0) {
193         printf("\nlist of processes\n");
194         for (j = 0; j < i; j++) {
195           if (pro[j].se.on_rq == 1) {
196             printf("pid: %d / start time: %d / running time: %d / weight: %d / time slice: %d / vruntime: %d\n", pro[j].pid, pro[j].start_time, pro[j].running_time, pro[j].se.weight, pro[j].se.time_slice, pro[j].se.
197               vruntime);
198           }
199         }
200       }
201     }
```

현재 rq에 존재하는 process들의 상태 및 정보를 보여주는 기능이 구현되었다.