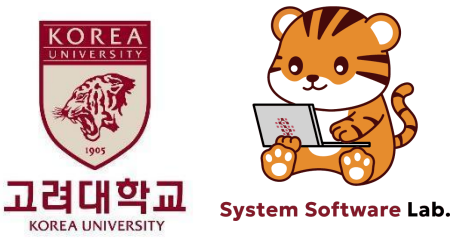


COSE341(01)

Operating system

Project #2 Report

Your name 이유림

Student # 2020390472

YOUR DEPARTMENT 컴퓨터학과

Submission Date : 25.11.17

Number of free days used for this project: 1

0. Development Environment

(1) Host OS : Mac ARM

(2) VirtualBox : lima-newlab

(3) Ubuntu : Ubuntu 22.04.3 LTS

(4) Linux Kernel : linux-6.6.36

1. Explanation of CPU scheduling and EEVDF [25 pts]

(1) Describe the need for CPU scheduling (5 pts)

CPU scheduling은 한정된 CPU 자원을 여러 프로세스가 효율적으로 사용할 수 있도록 어떤 process에 언제 CPU를 할당할지 결정하는 과정이다. process들은 각기 다른 I/O 및 CPU usage 패턴을 가진다. 따라서 적절한 스케줄링이 이루어지지 않으면 CPU가 불필요하게 idle 상태에 머물러 자원이 낭비될 수 있다. 또한 process별로 요구하는 CPU 사용량이 다름에도 이를 고려하지 않으면 일부 프로세스가 과도하게 CPU 를 독점하여 user가 불편함을 겪을 수 있다. 따라서 CPU 자원을 보다 효율적으로 활용하고, 프로세스 간 fairness를 위해 적절한 CPU scheduling이 필요하다.

(2) Explain the vruntime in CFS (6 pts)

CFS는 process간의 균등한 cpu 사용을 목적으로 하는 스케줄링 방식이다. process를 선택할 때 단순히 실제 CPU usage time만을 기준으로 하지 않고, 각 프로세스의 우선순위를 반영한 vruntime 값을 활용한다. vruntime은 process의 실제 실행 시간을 process의 우선 순위를 반영해 계산한 가상의 runtime 값이다. 이 때, priority는 nice라는 파라미터로 표현된다. CFS는 vruntime이 가장 작은 process를 다음 실행 대상으로 선택한다.

(3) Explain the vdeadline in EEVDF (6 pts)

EEVDF는 smallest virtual deadline을 가진 process를 다음에 실행할 process로 결정한다. Vdeadline은 현재 process의 virtual runtime에 time slice/weight을 더한 값이다. 여기서 slice는 모든 process에 동일하게 배당되는 고정된 값이며 weight는 각 process의 nice value에 따라 결정되는 가중치이다. Nice value가 클수록 weight이 작아지며, weight이

작아지면 (time slice / weight) 값은 커진다. 결과적으로 vdeadline 또한 증가한다. 즉, nice 값이 큰 프로세스는 상대적으로 더 낮은 우선순위를 가진다.

(4) Compare CFS and EEVDF (8 pts)

CFS와 EEVDF는 CPU scheduling policy이다. CFS는 process 간의 공정한 cpu 점유를 목적으로 하며, EEVDF는 CFS에서 latency와 responsiveness를 고려해 확장한 방법이다. CFS는 vruntime을 통해 다음 실행할 eligible한 process를 결정한다. vruntime이 작은 process를 다음에 실행할 대상으로 우선 선정하며, 이때 vruntime은 process의 실제 cpu 사용시간과 nice라는 우선순위에 의해 결정된다. 그러나 CFS는 high responsiveness를 가진 process를 고려한 scheduling은 하지 못한다. 또한, 이로 인해 process의 deadline이 guarantee되지 않는다는 한계를 지닌다.
EEVDF는 가장 작은 virtual deadline을 가진 process를 다음 process로 선정한다. 이때 vd는 vruntime에 (slice/weight)를 더한 값이다. weight는 각각의 nice 값에 따라 결정되며, slice는 모든 프로세스에 동일하게 부여된다. nice값이 작아질수록 weight는 커지고, (slice/weight)값이 작아져 vdeadline이 작아진다. 따라서 EEVDF에서는 latnecy에 민감한 프로세스에 작은 nice 값을 부여하여 우선적으로 실행될 수 있도록 한다.

2. Implementation [30 pts]

- **Describe how you implemented**
  (You don't need to attach the code, but you can attach it if necessary.)
- **A clear description of the implementation must be provided.**
- **Explanation of overall execution flow should be included.**

A) (linux)/kernel/sched/stats.h (15 pts)

수정한 stats.h의 sched_info_depart 부분은 커널 스케줄러에서 CPU 사용이 종료될 때 호출되는 함수이다. 코드에서 if ((t->policy == SCHED_NORMAL) && (strcmp(t->comm, "cpu") == 0)) 에서 SCHED_NORMAL은 CFS와 EEVDF 프로세스만을 대상으로 하며, 프로세스 이름이 cpu일 때만 실행되도록 하여 이외의 모든 스케줄링에서 동작하는 것을 방지했다.
pid_t pid = task_pid_nr_ns(t, &init_pid_ns);는 user level의 PID를 호출하는 부분으로, 명시적으로 작성하지 않으면 kernel level ID와 user level PID가 일치하지 않을 수 있다. 이 부분은 WSL의 요구사항이었지만 혹시 몰라 구현해두었다. int nice = task_nice(t);를 통해

nice 값을 가져오고, unsigned long long vruntime = t->se.vruntime;와 unsigned long long vdeadline = t->se.deadline;를 통해 각각 vruntime과 vdeadline을 확인한다.
커널 로그에서는 PID, vruntime, vdeadline, nice, CPU burst time을 출력하도록 하였다.
즉, stats.h의 sched_info_depart는 CPU burst 시간을 기록하고, 프로세스 실행 attribute 를 수집하여 PID, nice, vruntime, vdeadline을 커널 로그로 출력하는 역할을 한다.

parent 프로세스가 child 프로세스를 fork하면 child 프로세스는 sched_attr를 통해 스케줄링 attribute을 kernel에 전달한다. sched_info_enqueue()는 프로세스를 queue에 등록하여 대기 상태로 만든다. 프로세스가 CPU를 사용할 수 있게 되면 sched_info_arrive()가 호출되어 실제로 CPU를 점유하게 된다. 이때, user에서 정의한 matrix multiplication 연산을 수행하게 된다. CPU 시간이 만료되면 sched_info_depart() 호출하며 cpu를 떠나고 이 때 우리가 요청한 PID, vruntime, vdeadline, nice, CPU burst time을 kernel log를 출력하며, 프로세스는 다시 대기 queue에 들어간다. 모든 child process의 process가 종료될 때까지 해당 과정들이 반복된다.

B) /proj2/cpu.c (15 pts)

cpu.c코드는 user가 kernel의 sched_setattr 시스템 콜을 호출하여 프로세스의 스케줄링 속성을 설정하는 역할을 한다. sched_setattr는 성공하면 0을, 실패하면 -1을 반환하며, sched_attr struct를 통해 프로세스에 적용할 스케줄링 policy와 attribute를 커널에 전달한다. sched_attr struct는 size, sched_policy, sched_flags, sched_nice, sched_priority, sched_runtime, sched_deadline, sched_period를 담고 있다.
코드에서 if (pids[i] == 0) 구문은 fork() 호출 후 child 프로세스임을 확인하는 부분이다. fork()는 child 프로세스에게 0을 반환하고, parent 프로세스에게는 child의 PID를 반환한다. 따라서 해당 조건문 안의 코드는 child 프로세스이다. child 프로세스는 먼저 sched_attr 구조체를 초기화한 후, sched_policy를 SCHED_NORMAL로 설정하고, sched_nice를 각 프로세스에 지정된 값으로 설정한다. 이후 sched_setattr를 호출하여 커널에 nice 값을 변경하도록 요청하며 성공 시 0, 실패 시 -1을 반환한다. 프로세스의 실행 시간 측정을 위해 clock_gettime(CLOCK_MONOTONIC, &start_time)으로 CPU 점유 시작 시각을 기록하고, perform_calculation(matrix_size) 함수를 통해 실제 연산을 통해 CPU를 점유하여 실제 스케줄링 성능을 측정한다. 연산이 끝나면 clock_gettime(CLOCK_MONOTONIC, &end_time)으로 종료 시각을 기록한다. 기록된 시작 시간과 종료 시간을 통해 nano sec 단위로 측정된 turnaround 시간을 sec 단위로 변환하여 계산한다. child 프로세스는 실행이 끝나면 종료되며 PID, PPID, nice value, turn

around time을 출력한다. parent 프로세스는 wait을 통해 child 프로세스가 종료될 때까지 대기한다. wait(NULL)을 사용하여 child 프로세스가 종료되었음을 확인할 수 있게 하였다. 이 코드는 child 프로세스가 실행될 때 커널에 원하는 스케줄링을 적용하고, CPU 점유 시간을 측정하며, matrix multiplication을 통해 실제 CPU 사용을 테스트하도록 구성하였다.

3. Experiment results & Analysis [35 pts]

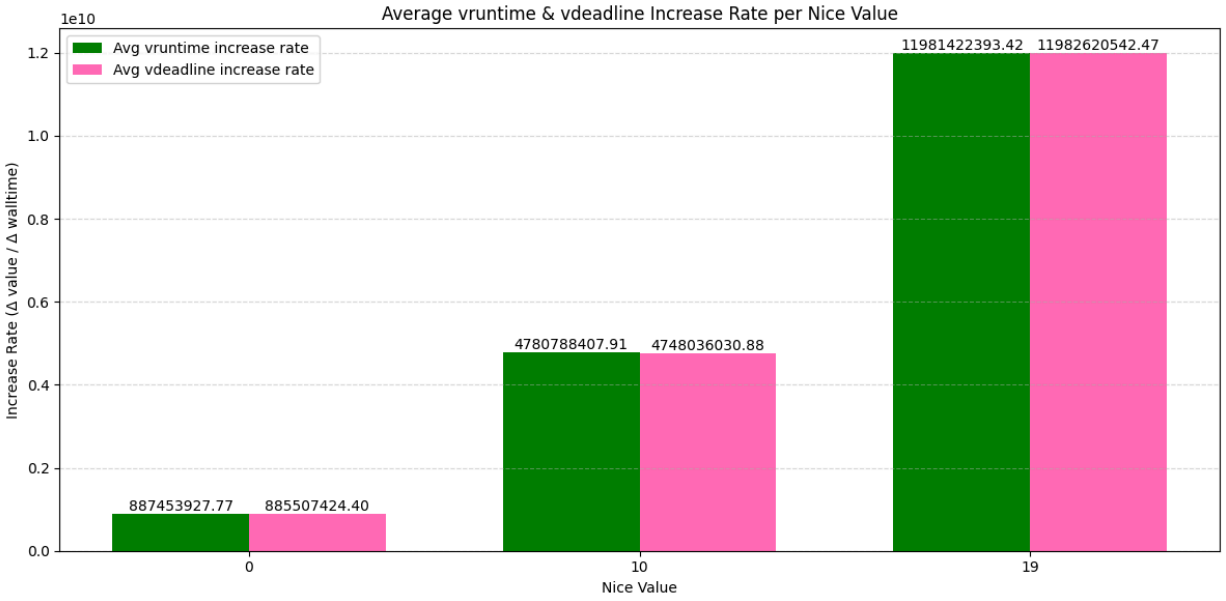
(1) Explain the behavior of EEVDF observed through experiments (10 pts)

Kernel log에는 PID가 9731부터 9736까지 총 6개가 나타난다. 보다 편한 가시성을 위해 user log 출력 시 child PID와 nice 값, turn around time뿐 아니라 parent PID(ppid)도 함께 출력하도록 하였다. 이를 통해 각 프로세스를 확인할 수 있는데, nice 0의 parent는 9732, child는 9735, nice 10의 parent는 9731, child는 9736, nice 19의 parent는 9733, child는 9734임을 알 수 있다.

Turn around time은 프로세스가 도착한 시간으로부터 작업을 마치기까지 총 소요된 시간을 의미하며, 각각 1.14초, 2.21초, 3.11초 경으로 nice가 클수록 turn around 시간이 오래 소요되었다는 것을 확인할 수 있었다. 그리고 이는 우리가 배운 nice 값이 클수록 우선 순위가 낮아지는 EEVDF의 내용과 일치한다.

로그를 보면, nice 값이 낮은 0과 10 프로세스가 여러 번 우선적으로 CPU를 점유하고, 후반부에야 nice 19 프로세스가 CPU를 사용함을 확인할 수 있었다. 반면 CPU burst time은 nice 값과는 명확한 상관관계를 나타내지 않았다.

(2) Draw a figure that compares the vruntime and vdeadline based on the nice value of each process (10 pts)



(3) Explain the result (15 pts)

먼저, dmesg 로그를 살펴보면 parent pid인 9731, 9732, 9733이 각 3회씩 등장한다. 3회 등장의 이유는 처음 wait 시작 시에 cpu를 한 번 점유하고, child가 종료되었을 때 또 한 번 점유한다. 그리고 child가 실행중일 때 parent를 wait상태로 두기 위하여 wait(null)이라는 function을 사용하는데, 이 함수는 주기적으로 깨어나서 child 상태를 확인한다고 한다. 그리고 이 때 cpu를 다시 점유하여 한 번의 log가 추가된다. Parent 로그에서 nice 값이 모두 0으로 표시되는 것은, parent 프로세스에 별도로 nice 값을 할당하지 않아 기본값 0이 적용되었기 때문이다. CPU 점유 시간 계산에서 child와 혼동될 수 있지만, skeleton에서 주어진 조건대로 두었다. 실제 vruntime 및 vdeadline 증가율 그래프를 그릴 때는 child의 nice 값만을 기준으로 계산하였다.

그래프를 보면, avg vruntime과 avg vdeadline은 nice 값이 증가함에 따라 증가하였다. 즉, nice 값이 클수록 vruntime과 vdeadline이 빠르게 증가하여 스케줄링 우선순위에서 멀어지는 경향을 보였다. 반대로 nice 값이 낮을수록 우선순위가 높아 vruntime과 vdeadline 증가율이 느리고, 더 빨리 CPU를 할당받아 turnaround 시간이 짧았다. 이 실험을 통해 CPU를 더 빨리 수행해야 하는 프로세스일수록 낮은 nice 값을 부여하여 높은 우선순위를 갖게 하는 이유를 확인할 수 있었다.