운영체제 2차 과제

학과 : 컴퓨터학과

학번 : 2018320161

이름 : 송대선

제출 날짜 : 2020-06-09

Freeday 사용 일수 : 0일

과제 개요 및 프로세스와 스케줄러의 개념

과제의 개요는 다음과 같다.

: 이번 과제는 프로세스와 스케줄러의 개념을 이해하기 위해 진행되었고, 과제의 목적은 CPU burst의 시간을 실제로 측정해보고 그것을 통계적으로 분석하는 것이다.

이를 위해서 스케줄러 코드를 직접 수정할 필요성이 있다. kernel/sched/stats.h의 sched_info_depart 함수는 프로세스가 CPU점유를 마쳤을 때 호출되는 함수이다. 따라서 이 함수로부터 프로세스의 CPU burst를 측정하는 것이 합리적이다. sched_info_depart 함수에는 CPU brust를 의미하는 delta가 이미 구현되어 있고, PID는 t → pid에, 그리고 프로세스의 호출 횟수는 t → sched_info.pcount에 구현되어 있다. 따라서 이 변수들을 이용하여 SAMPLE_TIMES 1000을 기준으로, 즉 프로세스 당 1000번에 한번씩 CPU burst 값을 추출한다.

커널을 컴파일하고, 적용한 뒤에 크롬으로 유튜브 영상 10개, firefox tab 2개, Thunderbird Mail 창 2개, Files창 2개, Rhythmbox의 Radio → All 9 genres → HBR1.com-Dream Factory 라디오 1개, LibreOffice Writer 창 2개, Ubuntu Software 창 1개, Help 창 1개, 대기중인 Terminal 2개를 띄웠다. 그후 약 30분간 CPU burst 측정 실험을 진행하였다.

dmesg > log.txt 명령어로 로그를 출력한 뒤 엑셀로 파싱한 뒤에 그래프를 그렸다.

프로세스의 개념은 다음과 같다.

: 프로세스는 디스크에 저장되어 있는 프로그램 파일을 실행시켰을 때, 그 프로그램을 메모리로 옮겨놓은 것이다. 프로세스는 전원이 끊기면 사라지지만, 프로그램은 전원을 꺼도 유지된다.

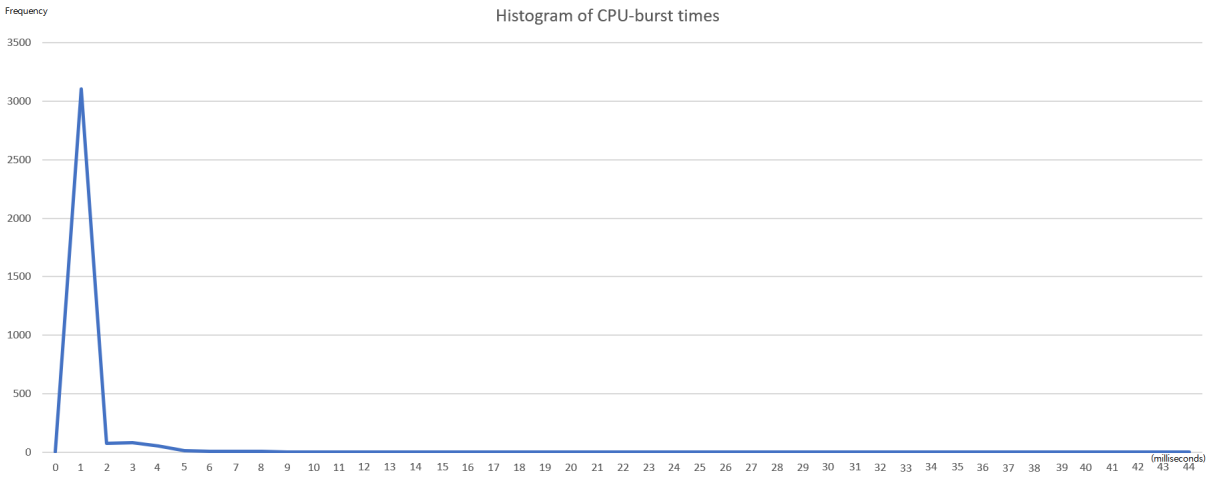
프로세스의 abstraction은 Execution unit과 Protection domain이다. Execution unit은 스케줄링의 단위라는 뜻이다. 이는 스케줄링이 process단위로 스케줄링이 된다는 의미이다. Protection domain이라는 말은 한 프로세스가 다른 프로세스의 메모리 영역을 침범할 수 없다(그렇게 되어서는 안된다)는 의미이다. 이 원칙을 깨는 것이 해킹의 기본적인 아이디어이다.

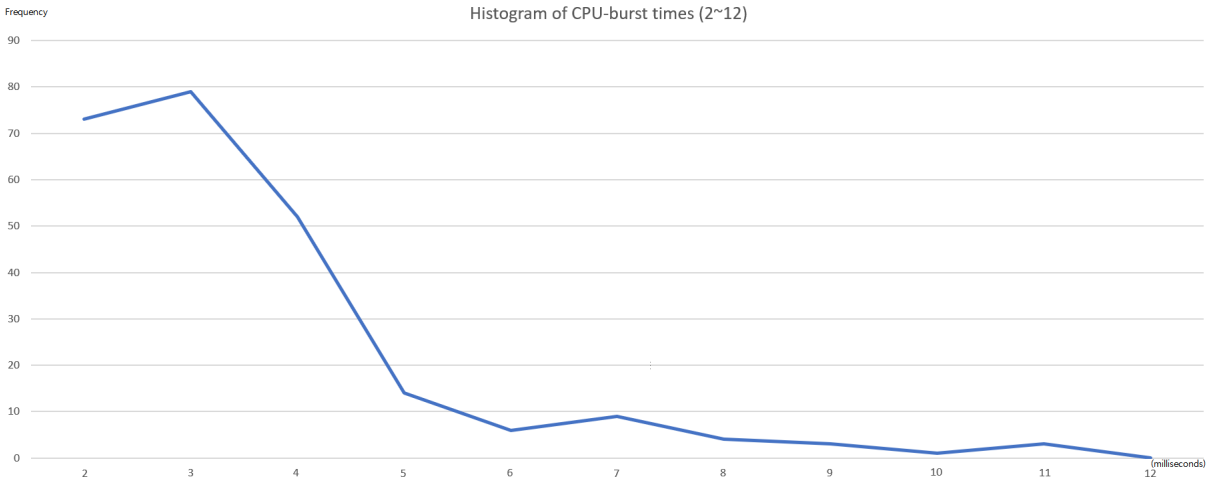
프로세스의 implemention는 다음과 같다. Program counter, Stack, Data section, 이 세가지 CPU의 레지스터를 구현하는 것이 프로세스의 implemention이다. Program counter은 프로세스의 코드 어느 부분을 수행하고 있는지를 계산하는 레지스터이다. Stack은 stack pointer로, stack 부분에서 내가 어디쯤 메모리를 사용하고 있는지를 알려주는 레지스터이다. Data section은 내 global data의 위치를 알려주는 레지스터이다.

스케줄러의 개념은 다음과 같다.

: 스케줄링이란 프로세스들을 실행을 시켜야 하는데, 어떤 순서로 프로세스와 CPU를 맵핑시키는지를 다루는 것이다. 스케줄링은 매우 중요한 영역이며, 스케줄링에 따라서 전체 시스템의 성능이 좌우된다. CPU 스케줄링의 목표는 당연하게도 CPU를 최대로 활용하는 것이다. 이 이야기는 다시말하자면 idle을 최소화하는 것과 같은 내용이다.

CPU burst에 대한 그래프 및 결과분석





분석결과 대부분의 CPU brust(3106개)가 1millisecond를 넘지 않는 것으로 나왔다. 좀 더 자세히 알아보기 위해 2~12milliseconds구간을 확대하여 보았다. frequency는 급격히 감소하며 11~12 milliseconds 구간은 frequency가 0이었다. 이는 수업시간의 제시된 그래프와 형태는 유사하다. 그러나 처음의 피크점에서 감소하는 속도가 훨씬 더 가파르다. 이 이유는 더욱 성능이 좋은 CPU를 사용했기 때문이라 생각된다.

작성한 모든 소스코드에 대한 설명

필자는 kernel/sched/stats.h의 sched_info_depart 함수만을 수정하였다. 원래 구현되었던 주어진 프로세스의 state가 running할 때의 조건문을 이용하였다. 그리고 t → sched_info.pcount를 이용하여, 그 프로세스가 SAMPLE_TIMES으로 나누어지는 수만큼 호출되었는지의 조건문을 만들 수 있었다. 만일 위 조건문 두개를 모두 만족한다면, 그 프로세스의 pid와 이미 구현되어있는 변수 delta를 이용해 CPU brust를 출력한다.

과제 수행 시의 문제점과 해결 과정 또는 의문 사항

처음에는 포인터 리스트나 구조체를 따로 만들어서 일일이 프로세스당 CPU 점유 횟수를 기록하게 만들었습니다. 하지만 알 수 없는 이유로 자꾸만 오류를 일으켰고, 프로세스당 CPU 점유 횟수를 의미하는 변수가 이미 있지 않을까하는 바램으로 코드를 다시 찾아보았고, pcount변수를 발견하게 되어서 그것을 이용해 구현하게 되었습니다.