2023-1 운영체제 과제 2

컴퓨터학과2019320110 정우성: 05/29 제출 (freeday 0일)

1. 과제 개요

본 과제는 Round robin scheduling에서 time slice 길이에 따른 cpu 연산 성능을 알아보는 과제이다. 과제를 진행하기 전, 예상한 실험 결과는 아래와 같다.

* Time slice가 늘어난다면, 동일한 시간 동안 context switching 횟수가 줄어들 것이므로 행렬 연산량이 증가
* Time slice가 늘어난다면, 동일한 이유로 cpu burst time이 증가

2. 코드

두 변인을 분석하기 위한 코드는 아래와 같다.

2.1 변수 선언

|  |
| --- |
| // 각 프로세스에 필요한 정보들을 저장  typedef struct proc\_info {  int proc\_num;  int calc\_count;  int calc\_total;  int total\_epoch\_dur; *// millisecond*  } Proc\_info;  // Round robin scheduling 설정을 위한 구조체 타입  struct sched\_attr {  …  };  // global variables  int forked\_procs;  Proc\_info info;  int epoch\_time;  int pid = -1;  int status;  //local variables  int total\_procs = atoi(*argv*[1]);  int exec\_time = atoi(*argv*[2]) \* 1000;  struct timespec init, clk\_proc, clk\_glb; // 프로세스 별로 && 전체 프로그램의 시간 측정을 위한 변수  struct sched\_attr attr; |

2.2 코드 실행 주요 부분

2.2.1 Round robin scheduling 설정

sched\_attr 구조체에 필요한 정보를 할당한 뒤 system call을 호출한다.

2.2.2 자식 프로세스의 연산

전체 행렬 연산이 끝나면 연산 횟수를 1 증가시키고, cpu.c의 실행 시간과 현재 프로세스의 실행 시간을 측정한다.

현재 프로세스가 100ms 이상 실행되었다면 이번 구간의 행렬 연산 로그를 출력한다.

cpu.c가 입력으로 주어진 시간보다 오래 실행되었다면 모든 프로세스가 종료된다.

2.2.3 SIGINT handler (추가 과제)

cpu.c가 실행되자마자 signal 함수를 사용하여 SIGINT 시그널에 대한 custom handler를 적용한다.

Handler의 동작은 정상적인 프로세스의 종료와 동일하다. (자식 프로세스면 total count 출력, 부모면 모든 자식 프로세스의 종료를 기다림)

2.2.4 cpu burst time 측정을 위해 dmesg를 kernel에 추가

/usr/src/linux-4.20.11/kernel/sched/stats.h를 과제 튜토리얼에 나온 것을 참고하여 변경하였다.

2.2.5 로그 분석을 위한 쉘 스크립트

자동화된 로그 분석을 위해 쉘 스크립트를 작성하였다.

한 bash shell 내에서는 과제 튜토리얼에 나온 cpu core 개수 제한이 유지되므로, 하나의 쉘 스크립트에서 모든 작업을 수행하면 연산에 사용되는cpu core개수를 쉘 스크립트 실행이 종료될 때까지 1개로 유지할 수 있다.

로그 분석을 위해 sed 명령어로 “total” 이 포함된 행을 추출하고, 별도의 Nms\_output.txt에 저장한다.

이 과정을 ITER 변수의 값만큼 진행한다.

반복이 끝나면 grep 명령어와 awk 명령어를 이용해 각 프로세스 별 총 연산 횟수를 계산한다.

grep 명령어와 awk 명령어로 dmesg 로그를 분석하여 총 cpu burst time을 계산한다.

위의 두 데이터를 모두 Nms\_result.txt에 저장한다.

3. 결과 분석

3.1 전제

연산 시간이 30초일 경우 시간이 context switching의 오버헤드가 충분히 반영되지 않는다고 판단하여, “최용욱” 학우의 질문에 대한 “정승우” 조교님의 답변을 참고하여 각 반복의 연산 시간을 300초로 늘린다.

또한, 모든 수치 값은 5회 반복의 평균을 사용하고, 시간의 단위는 second로 통일한다.

3.2 분석

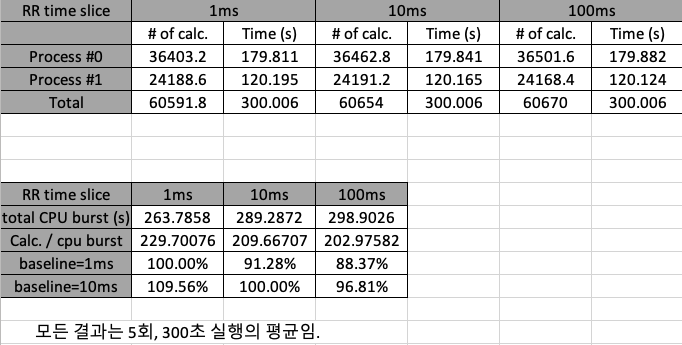


Figure 1: 전체 실행 결과

실험 결과는 위와 같다. 세 timeslice 모두 cpu.c가 약 300초 동안 실행되었다.

A picture containing text, screenshot, number, font

Description automatically generated

Figure 2: 각 timeslice별 전체 행렬 연산 횟수

각 timeslice 별로 전체 행렬 연산 횟수를 비교해보면, timeslice 길이가 늘어남에 따라 연산량이 증가함을 알 수 있다. 이에 기반하여 우리는 timeslice가 늘어날 때 context switching 횟수가 줄어 cpu burst time이 증가한다는 가설을 세울 수 있다.

A picture containing text, screenshot, number, line

Description automatically generated

Figure 3: CPU burst time을 이용한 초당 연산 횟수 비교

CPU burst time 이 가설에 맞게 증가하는 것을 볼 수 있다. 다만 눈여겨볼 점은, CPU burst 초당 연산 횟수가 줄어드는 경향성을 보이는 것이다. 따라서 성능 역시 timeslice가 늘어남에 따라 악화된다.

3.3 추가 연구 방향

Time slice가 줄어듦에 따라 cpu burst time, 행렬 연산 횟수 모두 증가했지만 성능은 악화되었다. 이는 time slice가 커짐에 따른 cpu burst time의 증가 경향성이 행렬 연산량 증가 경향성보다 높았기 때문이다. 따라서 충분한 양의 연산량 증가가 일어나기 위해, 다음의 아이디어를 제안할 수 있다.

* 연산 횟수를 3중 반복문의 제일 안쪽에서 증가시켜, 연산량 증가 경향성을 높인다.

이는 주어진 과제의 제약 조건에 어긋나기에 따로 실험하지는 않았지만, 추후 연구를 해보고 예상한 결과가 나오는지 알아보려고 한다.

4. 과제 수행 시의 문제점 및 보완할 사항

4.1 main process의 dmesg 데이터 제거

현재의 쉘 스크립트는 main process의 dmesg 로그를 계산 과정에 포함한다. 직접 데이터를 추적해본 결과, main process의 cpu burst time이 전체 burst time에 비해 무시할 수 있을 정도로 현저히 작고, 연산량 역시 마찬가지이다. 이 값을 포함해도 전체적인 데이터의 경향성에 차이가 없으리라 판단하여 따로 처리하지 않았다. 조금 더 정확한 데이터를 얻으려 한다면 이를 처리해줄 수 있다.

4.2 총 연산 횟수의 증가 경향성

3.3에 언급하였기에 따로 적지 않는다.

4.3 sudo 를 이용한 쉘 스크립트의 실행

쉘 스크립트 안에서 커널 조작 작업 및 cpu.c의 실행 등 sudo가 필요한 작업을 수행한다. 따라서 sudo bash ./core.sh 명령어를 사용하여야만 쉘 스크립트가 정상적으로 동작한다.

4.4 dmesg 버퍼 크기 조정

dmesg 버퍼 크기가 유한한데, 연산 시간을 300초 또는 그 이상으로 늘릴 경우 각 timeslice별 로그 개수가 dmesg 버퍼의 크기를 초과하는 경우가 발생한다. 따라서 dmesg 버퍼 크기를 32MB로 조정하고, dmesg 명령어에 -s 옵션을 추가하여 바뀐 버퍼 크기를 사용하도록 했다. 조정 방법은 stackoverflow를 참고하였다.

4.5 각 프로세스의 실행 시간100ms 별 연산량 출력

과제 튜토리얼에 나온 사진 그대로 출력하였다. “조얼” 학우의 질문에 대한 답변 및 수업시간에 공지된 사항에 따르면 예시 사진을 무시해도 괜찮다고 하였으나, 실험 결과의 경향성에 영향을 주는 로그가 아닐 뿐더러 불필요한 논쟁을 방지하기 위해 예시 사진을 그대로 따랐다.