1, 개요

thread 프로그래밍을 통해 concurrency 문제를 재현하고 이를 해결해보며 thread 프로그래밍시 일어날 수 있는 문제에 대해서 이해하기, 단일 스레드와 멀티스레드 프로그래밍의 수행시간을 비교해보기

2. 프로그램 구조 설명

2.1 함수에 대한 설명

pthread\_mutex\_t : 자료형으로 mutex를 선언한다.

pthread\_mutex\_lock() : 다른 thread의 접근을 제한한다

pthread\_mutex\_unlock() : 다른 thread의 접근을 허용한다

pthread\_mutex\_destroy() : mutex를 소멸한다.

pthread\_create() : 새로운 thread를 생성한다. 생성된 스레드는 부모와 소스, 시그널 핸들러, 열린파일 등을 공유한다.

pthread\_join() : sub thread의 처리가 끝나지 못하고 종료되는 상황을 방지하고 sub thread의 결과에 따른 처리를 위해 sub thread의 종료까지 대기함.

gettimeofday() :현재 시스템의 현재 시간 정보를 배열로 리턴한다.

2.2 다이어그램

|  |
| --- |
| 1. Concurrency 문제 재현 |
|  |

|  |  |
| --- | --- |
| 2. 단일 및 멀티스레드 행렬곱셈 (행렬 입력받는 경우) | |
|  |  |

|  |  |
| --- | --- |
| 단일 및 멀티 스레드 행렬 곱셈 (행렬 정해진 경우) | |
|  |  |

3. 실행 결과

|  |  |
| --- | --- |
| Concurrency 문제 발생한 경우/ 뮤텍스를 사용하여 정상 작동한경우   * 15번의 concurrency 문제가 발생하였다. | |
|  |  |

|  |  |
| --- | --- |
| 행렬 직접 입력받는 경우 단일 / 멀티스레드 곱셈연산 시간   * 멀티스레드 곱셈연산이 시간이 더 적다.(두 결과값을 평균낸경우) | |
|  |  |
| 행렬 정해져있는 경우 단일 / 멀티스레드 곱셈연산 시간   * 단일 스레드 코딩이 더 적은시간이 걸린다. | |
|  |  |

4. 고찰

우선, Concurrency 문제를 발생하는 것은 어려웠습니다. 처음에는 스레드 두개만 사용해서 전역변수 값에 ++ 연산을 1회 하는 형식으로 10번반복, 100번반복, 1000번 반복 했으나 나오지 않았고, 스레드를 3개로 늘려 10000번씩 수행해도 concurrency 문제가 일어나지 않았습니다. 전역변수를 연산하는 시간이 매우 짧아서 문제가 발생하지 않았을 것이라고 추측하여 ++연산과 – 연산을 반복해서 수행하였으나, 오류는 발생하지 않았고,

|  |  |
| --- | --- |
|  |  |

결국 스레드를 4개로 늘리고, sleep 시간을 매우 줄인 다음, ++연산을 10번, 25000번씩 반복하는 연산을 통해 오류를 재현할 수 있었습니다. 오류를 재현하고, 0.0015%의 확률이지만 이런 문제가 발생할 수도 있다는 것을 보고, 코딩시 세마포, 뮤텍스등을 사용해 이 문제를 예방해야겠다고 생각하였고, 더불어서 자율주행 자동차를 믿을 수 없게 됐습니다. 단일 스레드와 멀티 스레드 행렬 연산을 통해, 좀 더 쉬운 연산(함수가 정해진 경우) 일수록 멀티스레드 코딩을 할 때 단일 스레드보다 시간이 많이 소요됨을 확인하였습니다.

5. 프로그램 소스 파일

|  |  |
| --- | --- |
| Mutex.c | 주석 |
| #include <stdio.h>  #include <unistd.h>  #include <pthread.h>  volatile int ncount; //shared source  pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER; //clear thread  void\* do\_loop(void\*data)  {  int i;  for(i = 0; i<25000; i++)  {  pthread\_mutex\_lock(&mutex); // create lock  printf("loop1 : %d\n", ncount);  ncount ++;  ... //총 10번  printf("loop1 : %d\n", ncount);  ncount ++;    pthread\_mutex\_unlock(&mutex); // unlock  sleep(0.0001);  }  }  void\* do\_loop2(void\*data)  {  루프1과 동일  }  void\* do\_loop3(void\*data)  {  루프1과 동일  }  void\* do\_loop4(void\*data)  {  루프1과 동일  }  int main()  {  int thr\_id;  pthread\_t p\_thread[4];  int status;  int a = 1;    ncount =0;  thr\_id = pthread\_create(&p\_thread[0],NULL, do\_loop, (void \*)&a);  sleep(0.0001);  thr\_id = pthread\_create(&p\_thread[1],NULL, do\_loop2, (void \*)&a);  sleep(0.0001);  thr\_id = pthread\_create(&p\_thread[2],NULL, do\_loop3, (void \*)&a);  sleep(0.0001);  thr\_id = pthread\_create(&p\_thread[3],NULL, do\_loop4, (void \*)&a);    pthread\_join(p\_thread[0], (void \*) &status);  pthread\_join(p\_thread[1], (void \*) &status);  pthread\_join(p\_thread[2], (void \*) &status);  pthread\_join(p\_thread[3], (void \*) &status);  status = pthread\_mutex\_destroy(&mutex);  printf("check : %d\n", status);  return 0; | Pthread 라이브러리 삽입  전역변수 ncount  뮤텍스 선언  첫번째 함수  25000번 반복됩니다.  다른 스레드 접근 제한 (오류 원할 시 생략)  전역변수에 ++ 연산을  10번 수행하고 매번 출력  다른 스레드 접근 허용 (오류 원할 시 생략)  루프 1과 동일한 함수들 (루프 2, 3, 4)  스레드 선언  스레드 생성 (루프)  대기  스레드 생성 (루프2)  대기  스레드 생성 (루프3)  대기  스레드 생성 (루프4)  스레드의 종료까지 대기  뮤텍스 제거 |
| Matrix2.c | 주석 |
| #include <stdio.h>  #include <unistd.h>  #include <pthread.h>  #include <sys/time.h>  #define MILLION 1000000  #define A\_COL 2  #define A\_ROW 2  #define B\_ROW 2  #define B\_COL 2  pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER; //clear thread  void\* do\_loop(void\*data)  {  struct timeval tpstart, tpend;  long timediff;  int i, j, k;  static int a[A\_ROW][A\_COL];  static int b[B\_ROW][B\_COL];  static int c[A\_ROW][B\_COL];    if (A\_COL != B\_ROW){  printf("CANNOT CALCULATE\n");  return 0;  }    for(i=0;i<A\_ROW;i++){  for(j=0;j<A\_COL;j++){  c[i][j] = 0;  }  }    for(i=0;i<A\_ROW;i++){  for(j=0;j<A\_COL;j++){  printf("A(%d, %d)= ",i+1,j+1);  scanf("%d",&a[i][j]);  }  }  for(i=0;i<B\_ROW;i++){  for(j=0;j<B\_COL;j++){  printf("B(%d, %d)= ",i+1,j+1);  scanf("%d",&b[i][j]);  }  }  gettimeofday(&tpstart, NULL);  for(i=0;i<A\_ROW;i++){  for(j=0;j<B\_COL;j++){  for(k=0;k<A\_COL;k++){  c[i][j] += a[i][k]\*b[k][j];  }  }  }  printf("======RESULT======\n");  for(i=0;i<A\_ROW;i++){  for(j=0;j<B\_COL;j++){  printf("%d ",c[i][j]);  }  printf("\n");  }  printf("\n");  gettimeofday(&tpend, NULL);  timediff = MILLION\*(tpend.tv\_sec - tpstart.tv\_sec) + tpend.tv\_usec - tpstart.tv\_usec;  printf("It took %ld msec \n", timediff);  }  int main()  {  int thr\_id;  pthread\_t p\_thread[2];  int status;  int a = 1;    thr\_id = pthread\_create(&p\_thread[0],NULL, do\_loop, (void \*)&a);  sleep(10);  thr\_id = pthread\_create(&p\_thread[1],NULL, do\_loop, (void \*)&a);    pthread\_join(p\_thread[0], (void \*) &status);  pthread\_join(p\_thread[1], (void \*) &status);  printf("check : %d\n", status);  return 0;  } | Pthread 라이브러리  시간관련함수 라이브러리  곱연산할 행렬 A와 B 의 행,열 개수 정의  곱연산을 수행하는 함수  시작시간, 끝시간 선언  행렬 A 선언  행렬 B 선언  행렬 A\*B 담을 C 선언  행렬곱 안되는 경우 예외처리  행렬C 초기화 (멀티 스레드 코딩 시 필요함.)  행렬A입력  행렬B입력  시간측정 시작  A행렬과 B행렬 곱 과정  C행렬을 출력함  시간측정 종료  스레드 생성  입력받는시간 고려해서 대기  두번째 스레드 생성 (단일 스레드 코딩 시 생략 |
| matrix4.c (나머지 부분은 matrix 2와 동일) | 주석 |
| void\* do\_loop(void\*data)  {  struct timeval tpstart, tpend;  long timediff;  int i, j, k;  static int a[2][2];  static int b[2][2];  static int c[2][2];      for(i=0;i<2;i++){  for(j=0;j<2;j++){  c[i][j] = 0;  }  }    for(i=0;i<2;i++){  for(j=0;j<2;j++){  a[i][j] = i+j+1;  }  }  for(i=0;i<2;i++){  for(j=0;j<2;j++){  b[i][j] = i+j+1;  }  }  printf("======A======\n");  for(i=0;i<2;i++){  for(j=0;j<2;j++){  printf("%d ",a[i][j]);  }  printf("\n");  }  printf("\n");  printf("======B=====\n");  for(i=0;i<2;i++){  for(j=0;j<2;j++){  printf("%d ",b[i][j]);  }  printf("\n");  }  printf("\n");  gettimeofday(&tpstart, NULL);  for(i=0;i<2;i++){  for(j=0;j<2;j++){  for(k=0;k<2;k++){  c[i][j] += a[i][k]\*b[k][j];  }  }  }  printf("======A\*B======\n");  for(i=0;i<2;i++){  for(j=0;j<2;j++){  printf("%d ",c[i][j]);  }  printf("\n");  }  printf("\n");  gettimeofday(&tpend, NULL);  timediff = MILLION\*(tpend.tv\_sec - tpstart.tv\_sec) + tpend.tv\_usec - tpstart.tv\_usec;  printf("It took %ld msec \n", timediff);  }  int main()  {  int thr\_id;  pthread\_t p\_thread[2];  int status;  int a = 1;    thr\_id = pthread\_create(&p\_thread[0],NULL, do\_loop, (void \*)&a);  thr\_id = pthread\_create(&p\_thread[1],NULL, do\_loop, (void \*)&a);    pthread\_join(p\_thread[0], (void \*) &status);  pthread\_join(p\_thread[1], (void \*) &status);  printf("check : %d\n", status);  return 0;  } | 만들어진 행렬 곱 함수  행렬 A,B,C 크기 선언  2\*2행렬  C함수 초기화  A 행렬 :  1 2  2 3  B 행렬 :  1 2  2 3  A행렬과 B행렬 값을 출력함  시간 측정 시작  A\*B 곱셈 수행  A\*B 곱 출력  시간측정 종료  메인함수  스레드 생성  두번재 스레드 생성( 단일 스레드에서는 생략)\_ |

6. 자료 출처

뮤텍스 예제 : https://www.joinc.co.kr/w/Site/Thread/Beginning/Mutex

행렬을 입력받아 곱하는 함수 : https://blog.naver.com/kimht711/221260361690