운영체제 프로그래밍 과제

Implementation of a Simulator for Banker's Algorithm

문제 :

Banker's algorithm의 동작원리를 이해하기 위한 간단한 simulator를 만들려고 한다. 시스템 상황을 나타내기 위한 자료가 주어지고, 시간이 지나면서 각 프로세스가 시스템 자원을 요청할 때, 그 요청을 당장 들어줄 것인지 아니면 보류할 것인지를 결정하려고 한다. 시스템이 deadlock에 빠지지 않도록 safe 상태를 항상 유지하기 위해 Banker's algorithm을 이용하려고 한다.

각 프로세스가 자원을 요청하고, 그 요청을 수락해도 문제가 없는 경우, 즉 요청을 수락해도 safe상태를 유지할 수 있으면 OS는 자원을 요청한대로 사용할 수 있도록 수락한다. 만약, 요청한 자원을 당장 수락할 수 없는 경우 그 요청을 보류해 두었다가 (즉, 그 요청을 대기 queue에 보관함) 시스템 상황이 변한 후 (즉, 다른 프로세스가 자원을 반환한 경우) 대기 queue에 있는 요청을 순서대로 검사하여 요청을 수락할 수 있는 만큼 수락한다.

만약 프로세스 i가 요청한 자원이 NEED[i] 보다 큰 경우는 잘못된 요청이다. 이런 경우의 요청은 그 자체를 아예 무시한다.

입력 :

입력 파일의 이름은 banker.inp이다. 첫째 줄에는 총 프로세스의 개수 n과 자원의 종류를 나타내는 m이 주어진다. 다음 줄에는 각 종류의 자원에 대해 시스템이 얼마만큼의 자원을 보유하고 있는지를 나타내는 m 개의 정수가 차례로 주어진다. (n, m은 모두 50이하이다.)

이어 각 프로세스가 각 자원에 대해 최대 몇 개의 자원을 요청할 것인지를 나타내는 정보인 행렬 MAX (크기는 $n \times m$)가 주어진다. 프로세스의 번호는 0부터 n-1까지 이며, 자원의 번호는 0부터 m-1까지이다.

이어 각 프로세스가 각 자원에 대해 현재 몇 개의 자원을 사용하고 있는지를 나타내는 정보인 행렬 ALLOCATION (크기는 $n \times m$)이 주어진다.

입력 자료의 가독성을 높이기 위해 행렬 MAX와 ALLOCATION 앞에는 한 줄의 빈 줄이 삽입되며, 모든 입력 정수는 하나 이상의 공백으로 구분된다. 가독성을 높이기 위해 공백의 개수는 제한하지 않는다.

이어서 각 프로세스가 OS에게 얼마만큼의 자원을 요청하는지, 또는 얼마만큼의 자원을 반환하는지에 대한 명령문이 주어진다. 다음은 사용되는 각 명령문에 대한 형식을 설명한 것이다. 마지막 명령인 quit이 들어오면 모든 동작을 중지한다.

1) 프로세스 i가 m개의 자원에 대해 요청을 하는 경우:
   request $i\ m_0\ m_1\ \cdots\ m_{n-1}$
2) 프로세스 i가 m개의 자원에 대해 반환을 하는 경우:
   release $i\ m_0\ m_1\ \cdots\ m_{n-1}$
3) 시스템의 동작을 종료하길 원하는 경우:
   quit

출력 :

출력화일의 이름은 banker.out이다. 입력에 있는 명령 각각에 대해, 그 명령을 처리한 후 시스템에 있는 자

운영체제 프로그래밍 과제

원이 얼마나 available한지 즉, 각 명령을 처리한 후 아직도 자원이 얼마나 사용가능한 지를 보여야 한다. 각 자원에 대해 사용가능한 개수를 보일 때, 각 값은 하나의 공백문자로 구분한다.

예제 :

입력	출력
4 5 10 10 6 10 7 2 1 1 3 3 3 3 2 2 1 3 1 3 5 0 1 5 0 4 1 1 1 0 0 2 3 1 0 1 0 2 1 1 2 0 1 3 0 3 1 request 0 1 0 1 3 1 request 3 0 1 0 1 0 request 2 0 0 2 3 0 release 0 1 1 1 2 2 release 3 0 0 0 2 0 quit	2 4 4 1 3 2 3 4 0 3 2 3 4 0 3 3 4 5 2 5 3 4 3 1 5

제한조건: 프로그램은 banker.{c,cpp,java}로 한다.