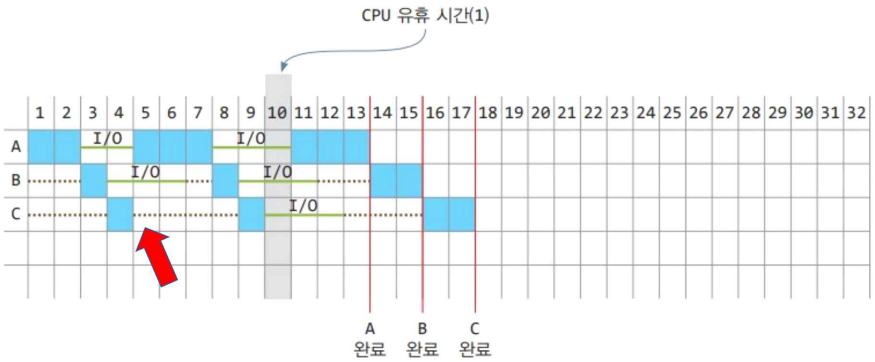
# Multi-programming System Simulation

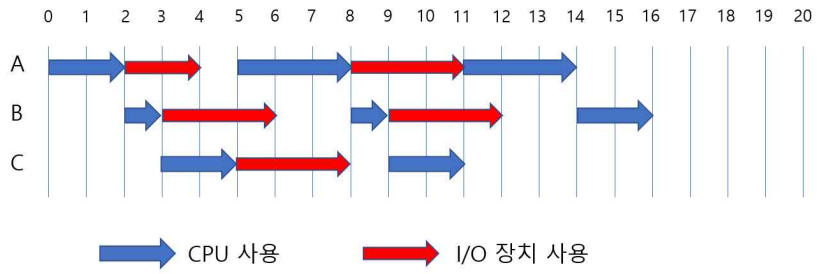
지난 과제에 이어, 여러분은 time machine을 타고 다시 한번 과거 시절로 여행을 떠난다. 1960년대 후반에 도착한 여러분에게 주어진 컴퓨터는 다중 프로그래밍 처리 방식으로 사용자 프로세스를 처리한다. 이제 여러분은 강의 시간에 다루었던 다중 프로그래밍 방법과는 살짝 변형된 방식의 다중 프로그래밍 운영체제를 흉내 내려고 한다. 이 변형은 단순히 과제의 난이도를 낮추기 위한 목적이다.

이 과제에서 흉내 낼 다중 프로그래밍 운영체제의 CPU 교체 방법이 강의 시간에서 다룬 것과 다른 점은 다음과 같다. 즉, **현재 CPU를 사용 중인 프로세스가 있으면, 그 프로세스가 종료하거나 또는 I/O를 할 때에만 CPU 교체가 일어난다.** (아래 <그림 1> 빨간 화살표로 표시된 곳에서 보듯이 교재에서는 현재 CPU를 사용하고 있는 프로세스 C가 시각 5에서 I/O를 요청하지 않았지만 프로세스 A가 I/O를 종료하였기 때문에 CPU가 프로세스 A에게 넘겨진다.)




<그림 1>

<그림 1>에서 보인 프로그램의 제출 상황과 동일한 상황에서 본 과제에서 구현할 변형된 다중 프로그래밍 운영체제에 의해 프로그램이 어떻게 실행되는지를 <그림 2>가 보여 준다.




<그림 2>

CPU를 다른 프로세스에게 할당해야 할 상황이 되면, OS는 현재 **I/O를 하지 않는 프로세스 중에서 프로세스 번호가 가장 낮은 것에 CPU를 할당한다.**

*N* 명의 사용자 프로그램 실행 패턴에 대한 정보가 주어질 때, 다중 프로그래밍 운영체제가 이들을 차례대로 처리한 후, 모든 작업이 종료된 시점과 CPU 유휴시간(idle time)을 계산하려고 한다.

## 입력 :

입력 파일의 이름은 multi.inp이다. 첫째 줄에는 시스템이 처리해야 하는 프로그램의 수를 나타내는 정수

$N(3 \le N \le 1000)$이 주어진다. 이 $N$개의 프로그램은 모두 시각 0에 제출되었다고 가정한다. 시스템이 처리하는 프로그램의 순서는 입력에서 주어지는 순서를 따른다.

각 프로그램은 아래와 같이 반복되는 CPU 작업과 I/O 작업에 소요되는 시간을 순서대로 나열한다. $t_1$은 CPU 사용 시간, $t_2$는 I/O 사용 시간, $t_3$는 CPU 사용시간, ... 을 의미한다. 즉, $t_k(3 \le k \le 100)$에서 $k$가 홀수이면 CPU 사용 시간을, 짝수이면 I/O 사용 시간을 의미한다. '-1'은 마지막 입력을 의미하며 이는 처리하지 않는다.

$t_1, t_2, t_3, \ldots, t_k$, -1

**출력 :**

출력화일의 이름은 multi.out이다. $N$개의 프로그램을 다중 프로그래밍 OS가 모두 처리한 후, CPU 유휴 시간과 처리가 종료된 시각을 한 줄에 출력하되, 두 값은 공백으로 구분한다.

**예제 :**

| 예 1 | 예 1에 대한 출력 |
|---|---|
| 3<br>2 2 3 3 3 -1<br>1 3 1 3 2 -1<br>2 3 2 -1 | 0 16 |

| 예 2 | 예 2에 대한 출력 |
|---|---|
| 5<br>2 10 3 12 5 12 -1<br>4 13 5 10 8 11 9 -1<br>6 11 8 12 5 11 9 12 -1<br>8 22 3 12 4 6 7 7 8 12 2 -1<br>3 3 8 12 3 22 12 23 12 23 3 22 -1 | 70 229 |

**제한조건:** 프로그램의 이름은 multi.{c,cpp,java}로 한다.