운영체제

- Term Project_Unix Scheduling 해설서 -

담당교수	서희석 교수님
학번	2009135044
이름	나동희
제출일	2013.06.02

[목 차]

*수정 사항

JAVA 파일에 대해서 클래스가 학번(숫자)으로 생성되지 않는 점 보고서를 통해 양해드리며 따라서 파일명 또한 바뀜을 알려드립니다.
(2009135044.JAVA / CLASS -> OS_20009135044.JAVA / CLASS)
또한 입력은 파일을 통한 것이 아닌 프로그램 내에서 SCAN 함수를 받아 프로세스의 개수를 입력하여 구동합니다.
(2009135044_나동희.zip 내에 값 입력 파일에 해당하는 파일 없음)

0. 프로젝트 파일 구성

2009135044_나동희.ZIP에 대해 압축을 풀면 다음과 같이 나온다.

해설서 (SOURCE CODE 포함), 실행 파일 (OS_2009135044.JAVA, OS_2009135044.CLASS)

해설서: 프로젝트에 대한 전체적인 해설 내용이 담긴 HWP 파일이다.
실행파일: 프로젝트를 구현하기 위해 필요한 JAVA와 CLASS 파일이다.

1. 프로젝트의 주제

Unix 운영체제의 간단한 스케줄링 방법을 구현하는 프로젝트
(시각적인 효과를 최대한 높이도록 구현.)

유닉스 스케줄링 (Unix Scheduling)

	프로세스 P1		프로세스 P2		프로세스 P3	
	우선 순위	CPUCount	우선 순위	CPUCount	우선 순위	CPUCount
0	60	0 1 2 … 60	60	0	60	0
1	75	30	60	0 1 2 … 60	60	0
2	67	15	75	30	60	0 1 2 … 60
3	63	7 8 9 … 67	67	15	75	30
4	76	33	63	7 8 9 … 67	67	15
5	68	16	76	33	63	7

그림 1. UNIX SCHEDULING

Unix 운영체제는 대회형 사용자들에게 적절한 응답 시간을 제공하고 우선순위 기반 스케줄링을 사용한다. 우선순위가 높은 프로세스부터 디스패치를 하며, 우선순위는 사용량에 따라 주기적으로 변한다. 또한 MFQ 기반으로 스케줄링을 시행한다. 우선순위의 조정은 프로세스의 모드에 따라 순서가 나뉘는데 커널 모드에 있는 프로세스가 사용자 모드에 있는 프로세스보다 먼저 배정을 받는 커널 우선순위(Kernel Priority)를 통해 커널 모드 프로세스의 우선권을 주고 그 후 사용자 프로세스들 내에서 우선순위 (User Priority)를 정해 스케줄링을 시작한다. 이때 주기적으로 인터럽트를 발생시키는 클럭 핸들러(Clock Handler)에 의해 정해진 시간에 따라 모든 프로세스들의 우선순위를 조정하는데 이 과정은 Decay 연산과 기본 우선순위와 CPU COUNT의 더해짐에 새롭게 결정되는 우선순위에 따라서 이루어진다.

2. 프로젝트의 구현

구현을 위해 유닉스 스케줄링 기법에 대한 구성 요소 설정을 하고, 순서도로 이 과정이 어떻게 되는지 쉽게 표현해본다. 그리고 기초적인 요소가 정립된 뒤에는 그래픽을 어떻게 표현할 것인지 GUI 설계를 통해 결정한다.

2_1. 구성 요소 설정

유닉스 스케줄링을 구현하기 위해 각 프로세스에 필요한 요소는 다음과 같다.

모드, 우선순위, 기본 우선순위, 할당 시간, CPU COUNT, ~~NICE VALUE~~

모드: 커널 프로세스와 사용자 프로세스로 이루어져 있다.
우선순위: 프로세스의 첫 우선순위로 값은 프로세스마다 틀리다.
기본 우선순위: 새로운 우선순위를 형성할 때 Decayed Cpu Count에 더해주는 값.
할당 시간: 프로세스가 끝나기까지 필요한 시간
CPU COUNT: 클럭 핸들러에 의해 발생하는 인터럽트 타이밍을 위한 카운터.
~~NICE VALUE~~: 우선순위를 낮추기 위해 할당해주는 사용자 지정 값 (교재와 마찬가지로 0으로 설정.)

2_2. LOGIC 설계

유닉스 스케줄링의 우선순위는 각 프로세스들의 모드가 커널인지 사용자인지를 살펴보는 것에서부터 순서를 정할 수 있다. 만약 N개의 프로세스에서 3개가 커널 모드 프로세서라면 나머지 N-3개의 사용자 모드 프로세스는 커널 모드 프로세서들끼리의 우선순위 연산을 통해 프로세스가 완료될 때까지 대기한다. 그 후에 남은 사용자 모드 프로세스끼리의 우선순위 비교를 통해 기법을 진행한다. 우선순위는 프로세스의 구성요소를 통해 매 단계마다 조정된다. 이에 관련된 연산은 3가지로 이루어져있다.

ⓐDecay 연산
CPUCount = CPUCount + (1~60)
CPUCount = CPUCount/2

ⓑ우선순위 연산
Priority = basePriority + (CPUCount/2) +niceValue(=0)

ⓒ할당시간 연산
Time = Time - 1

위에 나온 연산들로 모든 프로세스들의 처리가 끝날 때까지 클럭 핸들러에 의한 인터럽트에서 우선순위가 바뀌어가며 과정을 진행시키는데 이 과정에서 가장 낮은 우선순위를 가진 프로세스를 찾는다. 다음으로는 모든 프로

세스에 대해 우선순위와 각 프로세스에 내재된 CPU COUNT에 대해 아래처럼 4가지의 케이스를 따져야 한다.

Case_1: 한 번도 실행되지 않은 상태에서 우선순위 값이 가장 낮아 프로세스를 처리하는 경우
단계 별 과정에서 Priority 값이 가장 낮으며, CPU COUNT =0인 프로세스를 가르킨다.
이때는 위에 나온 ⓐ,ⓑ,ⓒ 연산을 모두 실행해주면 된다.

Case_2: 이미 실행되었던 상태에서 우선순위 값이 가장 낮아 프로세스를 처리하는 경우
단계 별 과정에서 Priority 값이 가장 낮으며, CPU COUNT가 예전에 이미 한번 이상 실행되었기 때문에 0 이상인 프로세스를 가르킨다. 이때도 위에 나온 ⓐ,ⓑ,ⓒ 연산을 모두 실행해주면 된다.

Case_3: 한 번도 실행되지 않았고, 우선순위 비교에 밀려 대기하는 경우
단계 별 과정에서 우선순위 비교를 통해 실행되지 않고, 그 이전에도 한번도 실행되지 않아 CPU COUNT가 0인 프로세스이다. 이 경우 아무런 연산 없이 과정을 이어나가면 된다.

Case_4: 이미 실행되었던 상태에서 우선순위 비교에 밀려 대기하는 경우
단계 별 과정에서 우선순위 비교를 통해 실행되지는 않지만, 그 이전에 한 번 이상 실행되어 CPU COUNT가 0 이상인 프로세스이다. 이 과정에서는 처리가 된 것이 아니므로 시간 연산 없이 연산 ⓐ와 ⓑ를 행하면 된다. 이때 ⓐ 연산에서 CPU COUNT 값을 60까지 더해주는 것 없이 Decay 연산만 시행하면 된다.

이를 정리하면 아래의 순서도와 같다.

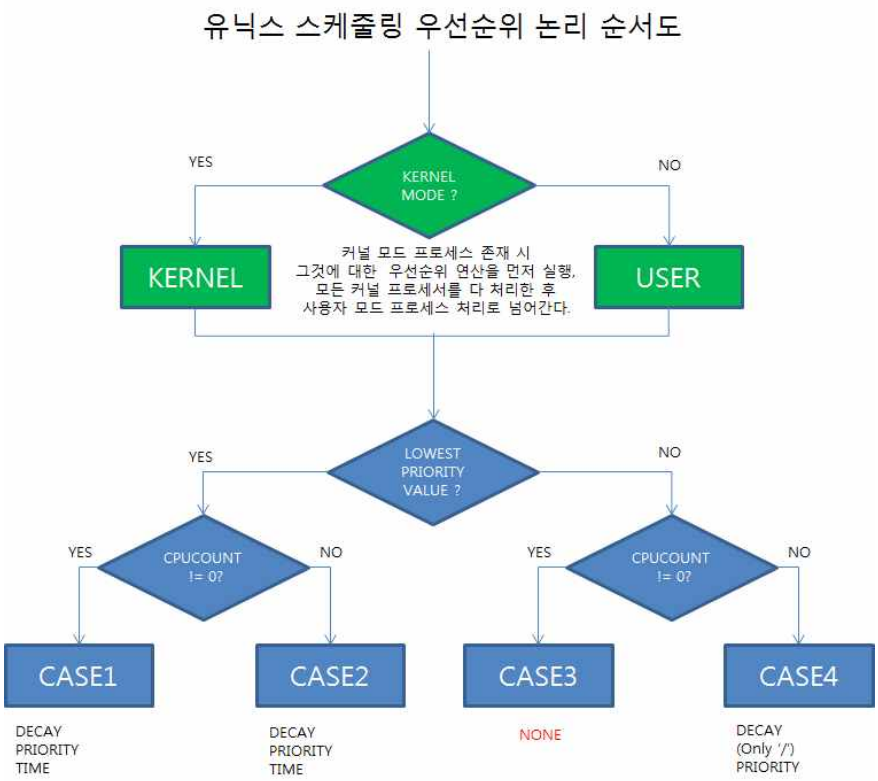


그림 2. UNIX SCHEDULING 논리 순서도

2_3. GUI 설계

LOGIC 구현의 시각화를 위해 JAVA Ecilpse 프로그램을 통해 GUI를 형성한다. GUI의 특성은 변화되는 값에 따라 나타나는 이미지의 연속적인 변경인데, 유닉스 스케줄링에 대해 그러한 부분을 찾아 그래픽 적인 요소를 가미해 기본적인 틀을 만든다. GUI를 구현했다고 가정했을 시의 프레임의 구성은 밑의 그림과 같다.

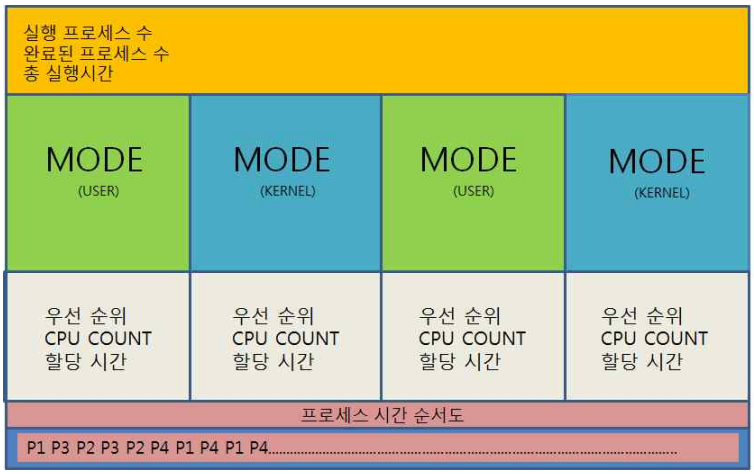


그림 3. GUI 환경을 통해 구현할 UNIX SCHEDULING FRAME 창

위에서 아래부터 크게 총 4개의 패널로 구성되어 있으며 작게는 다음과 같다.

상단 패널 – 3개의 라벨 (실행 프로세스 수, 완료된 프로세스 수, 총 실행시간)
중단 패널_상 – n개의 패널과 라벨 (실행 프로세스 수만큼 생성 후 패널에 라벨;아이콘/이미지 삽입)
중단 패널_하 – n개의 패널과 3*n개의 라벨 (각 프로세스의 속성을 나타내주는 3개의 라벨 각 패널마다 삽입)
하단 라벨 – 프로세스 시간 순서도 (제목)
하단 패널 – n개의 패널과 라벨 (총 실행 시간만큼의 이미지가 가로로 적재)

이 프레임 구성에서 유닉스 스케줄링을 고려한 이벤트 삽입은 다음 부분에서 이루어진다.

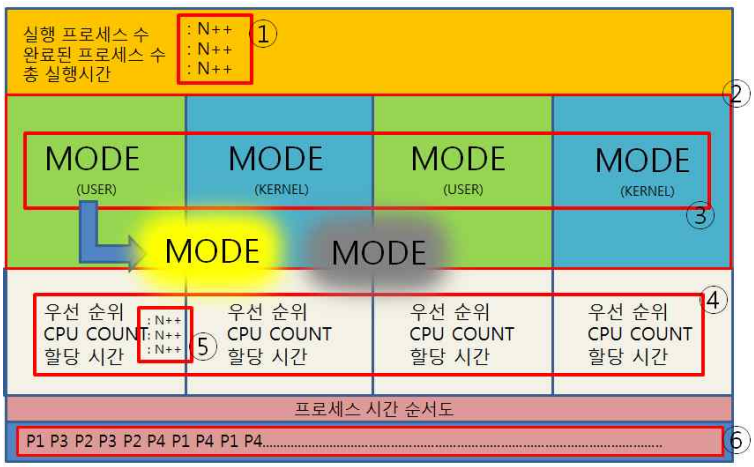


그림 4. GUI 환경을 통해 구현할 UNIX SCHEDULING EVENT

1. 실행 프로세스 수, 완료 프로세스 수, 총 실행시간은 프로그램의 연산이 이루어질 때마다 해당 값이 바뀐다.

2. 프로세스를 생성할 때 (혹은 생성된 프로세스를 나타낼 때) 그 값이 사용자 모드인지 혹은 커널 모드인지 이미지가 적용된다.

3.우선 순위 비교를 통해 실행되는 프로세스는 어떠한 프로세스인지 이미지를 통해 알 수 있게 해준다. 또한 실행이 완료된 프로세스도 이미지 변경을 통해 알 수 있게 한다.

4. 실행이 되는 중의 속성 값의 색을 변경하여 현재 실행하는 프로세스라는 점을 부각 시키고, 완료가 다된 프로세스는 더 이상 속성을 드러내지 않고 “실행 완료”로 라벨을 수정한다.

5. 실행될 때의 연산(CPU COUNT, PRIORITY, TIME 값)이 적용되어 어떻게 다시 저장이 되는지 보여준다. CPUCOUNT 값이 +60이 되는 것이 보여지고, 그 후 연산을 통해 바뀐 값을 적용시킨다.

6. 언제 어떤 프로세스가 실행되어 할당 시간의 일부를 소모했는지 간단한 이미지를 차례로 적재시킨다.

2_4. PROGRAMMING 설계

JAVA Ecilpse 프로그램을 통해 설계한 LOGIC과 GUI를 이용, 실행 한다.
UNIX SCHEDULING PROJECT에서 프로그래밍이 필요한 부분은 다음과 같다.

1. 프로세스 클래스 생성

2. 변수 생성
FOR, WHILE에 대한 변수. 입력 변수, 랜덤 함수 변수
완료 프로세스 카운팅 변수, 실행시간 카운팅 변수, 총 실행시간 적재 변수,
실행되는 프로세스 클래스 배열의 해당 NUMBER를 정하는 변수

3. 실행할 프로세스 수만큼 FOR문을 돌려 해당 부속 패널과 라벨 생성
화면 중단 그림에 대한 패널과 라벨
화면 중단 프로세스 번호에 대한 패널과 라벨
화면 중단 프로세스 속성에 대한 패널과 라벨 x3

4. FOR문을 돌려 클래스의 속성을 랜덤으로 지정 및 속성 중 모드에 대한 조건문을 통해 라벨에 이미지 적재

5. 최적의 우선순위를 구하기 위해 단계마다 각 클래스의 우선순위를 저장하는 배열 생성
배열에 내재된 값 탐색을 통해 최적의 우선순위 값을 가진 NUMBER를 TEMP 변수에 저장

6. 구현한 모든 프로세스가 끝날 때까지 매 단계마다 모든 프로세스에 대한 UNIX SCHEDULING 연산 구현. 그 중 CASE 1과 2는 같은 연산이므로 하나로 묶고 CASE1 || CASE2에서 실행될 때마다 화면 중단 그림과 프로세스에 속성에 대한 라벨 변경, 추가로 CASE4에 대해서도 변경 값 출력. 프로세스 시간 순서도에 대한 패널 및 라벨 추가 생성 (조건문을 통한 이미지 설정 포함) 적재

7. 모든 프로세스가 완성되면 WHILE문을 빠져나오고 프로그램 종료.

2_5. 전체 설계

유닉스 스케줄링의 개념을 시각적으로 구현하기 위해 LOGIC과 GUI의 설계를 실제 코딩 환경에 적응 되도록 PROGRAMMING을 통해 적절하게 맞춘다. 설계는 크게 LOGIC을 위한 클래스 생성이나 변수 생성 및 GUI환경을 조성하는 부분인 전반부와 갖춰진 조건에서 반복문과 조건문을 통한 논리 연산을 수행하는 후반부로 나누어 진다. 그러나 세부적으로 보면 이벤트를 수행하기 위해서 논리를 통해 액션을 만들어야 하기 때문에 전반부와 후반부 모두 두 분야의 설계가 혼용되어 있다. 전체 설계도는 다음과 같이 나타낼 수 있다.

1. 유닉스 스케쥴링 프로세스 클래스 생성
2. GUI와 LOGIC 구현을 위한 기본틀짜기 및 클래스(배열) 생성
3. 패널 틀 짜기
4. 패널(1~3) 내용 생성 (LOGIC 일부 구현; 최적우선순위 도출)
5. 패널4 내용 생성 및 주요 LOGIC을 구현

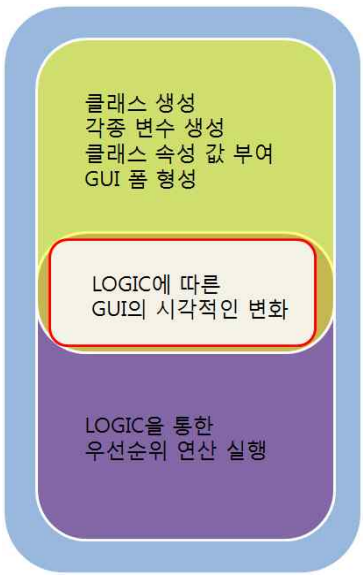
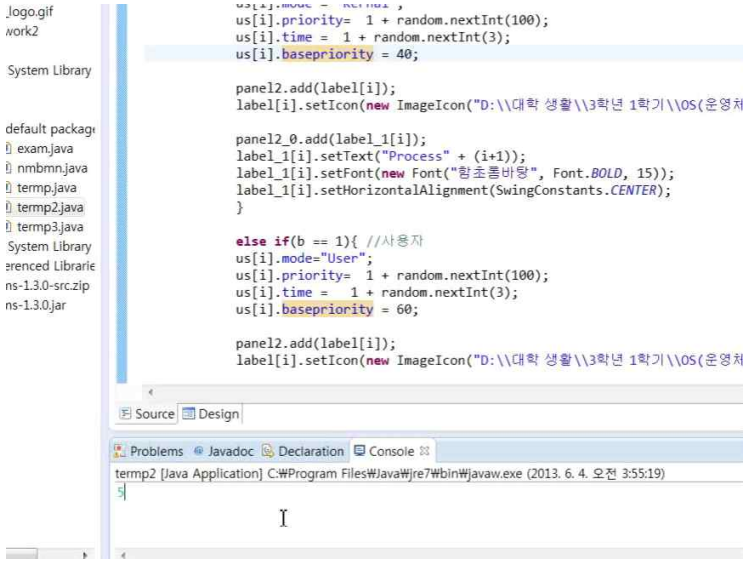


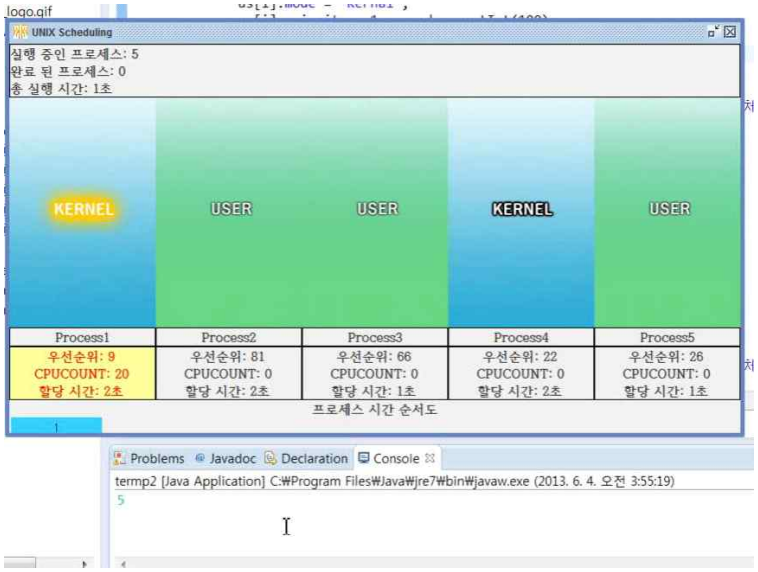
그림 5. JAVA를 통해 구현할 UNIX SCHEDULING의 전체 설계 순서

3. 프로그램 실행 및 캡쳐

다음은 소스코드를 통해 구현한 유닉스 스케줄링 프로그램의 실행 과정 캡쳐 화면이다.

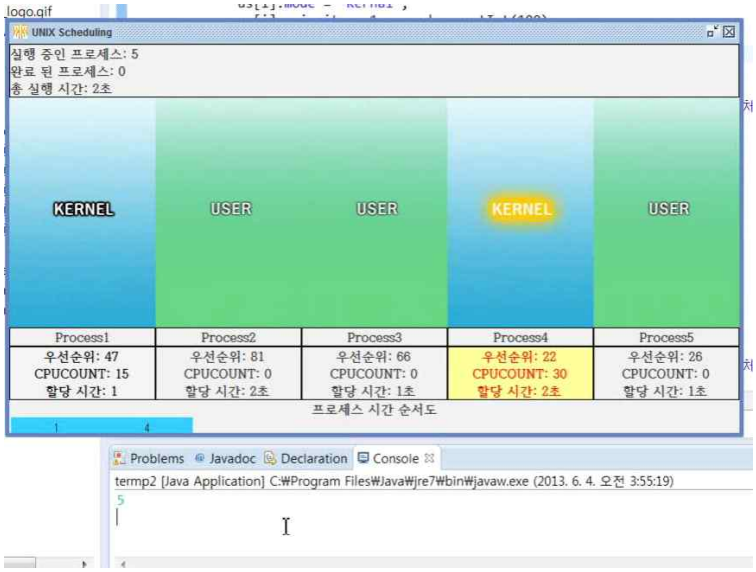


1. 구현 과정을 시뮬레이션 할 프로세수의 개수 (5)를 입력한다. 입력한 만큼 클래스와 패널, 라벨을 각 설계에 맞게 생성하여 기본적인 틀을 구성한다. 프로세스 클래스의 속성은 랜덤으로 그 값을 받아 후에 나타날 FRAME 에서 각각 다르게 나타난다.

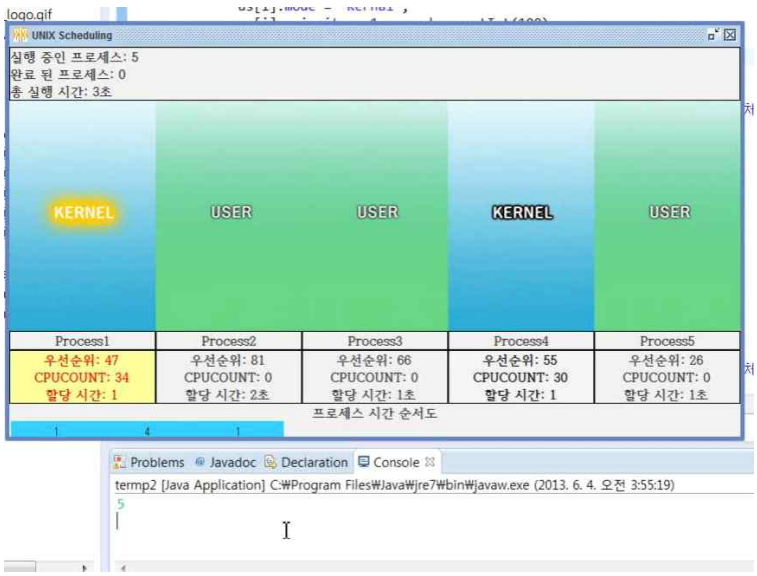


2. 설계된 코딩을 통해 새로운 FRAME 창에서 UNIX SCHEDULING을 구현한다. 프로세수의 개수 (5) 만큼의 속성 정보가 뜬다. 패널 상단에는 실행 시뮬레이션에 대한 프로세스 정보와 실행 시간이 업데이트 된다. 패널 중단에는 모드를 나타내주는 이미지가 있고 우선순위가 가장 낮은 프로세스에 대한 연산이 실행됨을 모드 이미지의 변경과 속성을 나타내주는 부분의 색 변화, CPU 카운트의 업데이트 및 후의 할당 시간과 우선순위 값의 변경,

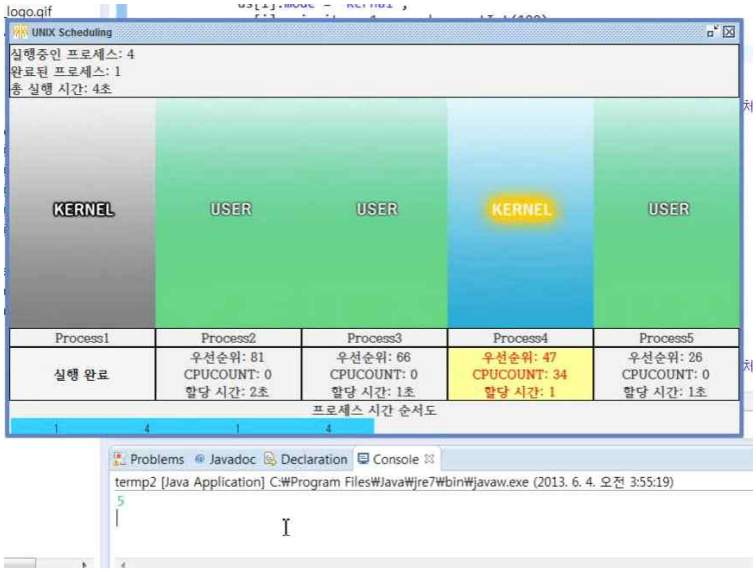
마지막으로 프로세스 시간 순서도의 적재되는 이미지를 통해 알 수 있다. 앞서 말했듯이 커널 모드와 사용자 모드에서 커널 모드 프로세스부터 실행시켜 주어야하기 때문에 아무리 사용자 모드 프로세스의 우선순위 값이 낮아도 커널 모드 프로세스끼리 우선순위를 비교하고 연산을 처리한다. Process1번은 40+15의 값으로 55가 된다.



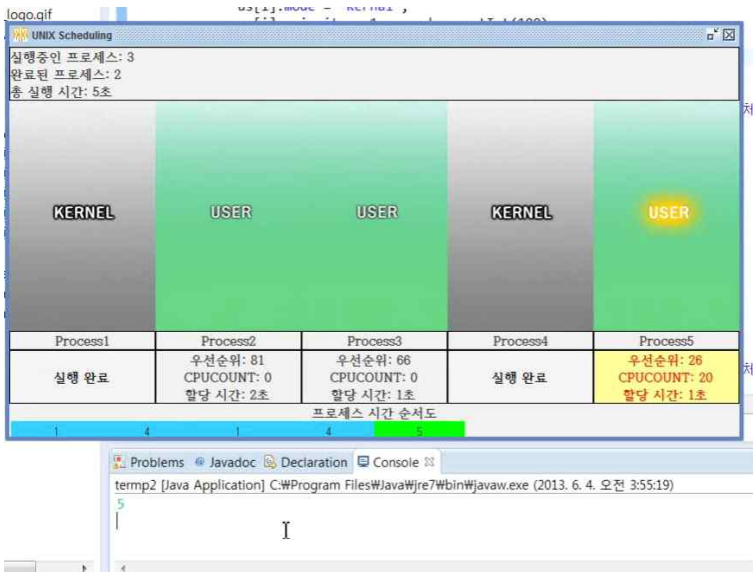
3. 이전 연산이 끝나고 다음으로 우선순위가 낮은 Process4번이 다음 연산을 실행한다. 이때 이전에 실행되었던 Process1은 이미 자체적으로 CASE4번- 실행되는 프로세스는 아니지만 이미 한번 이상 실행이 되었을 경우-에 속하여 CPUCOUNT에 대한 Decay 연산과 우선순위 재정립 연산을 실행한 상태이다. 따라서 CPUCOUNT는 30에서 15가 되었고 그림처럼 우선순위는 40+ 7= 47이 된다.



4. 다시 커널 모드 프로세스의 우선순위 비교에서 Process1이 값이 낮으므로 연산을 실행한다. FOR문이 차례대로 돌기 때문에 그림에서의 Process1 연산이 끝나는대로 Proess4의 CASE4번 경우의 연산이 실행되어 우선 순위는 55에서 47이 될 것이다. 시간이 지나면서 프로세스 시간 순서도 밑 패널에 해당 프로세스 모드의 색상을 띈 이미지가 재적이 된다.

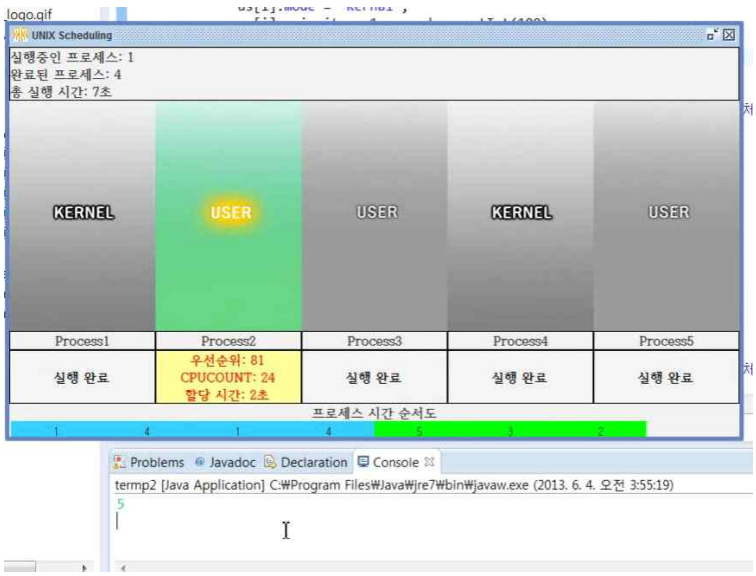


5. 프로세스 중 실행이 완료된 프로세스는 이미지의 변경과 함께 속성 칸에서 “실행 완료”라고 출력된다. 더불어서 상단 패널의 실행 중인 프로세스와 완료된 프로세스에 대한 카운팅이 동시에 이루어진다.

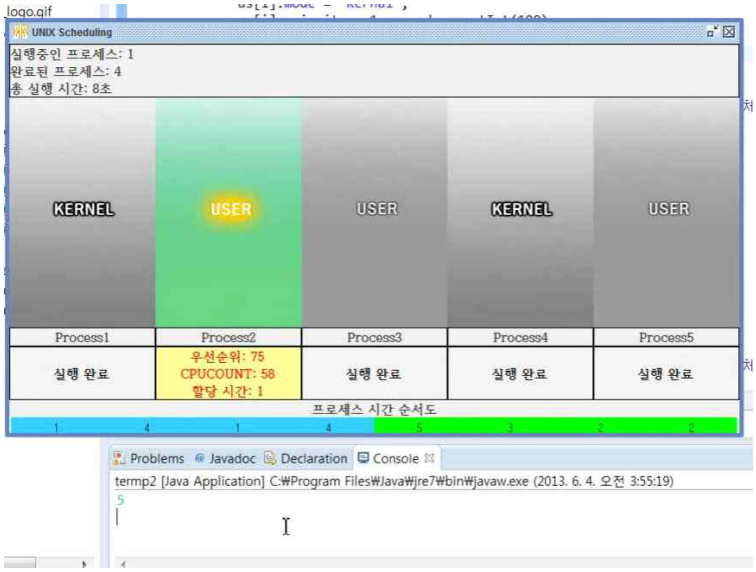


6. 모든 커널 모드 프로세스의 실행이 끝나면 비로소 사용자 모드 프로세스끼리의 우선순위 비교를 통한 연산이 시작된다. 세 개의 우선순위중 가장 낮은 Process5에 대한 연산을 실행한다. (이 부분도 프로그래밍으로 구현이 되어 있다; 소스코드 주석 참조) 최적의 우선순위를 찾으면 시간 순서도에는 이미지가 적재되는데 커널 모드의 프로세스와 다른 색상의 이미지로 배경을 채워넣어 UNIX SCHEDULING 기법에선 커널 모드 프로세스가 사용자 모드 프로세스보다 먼저 실행이 된다는 것을 시각적으로 보여준다.

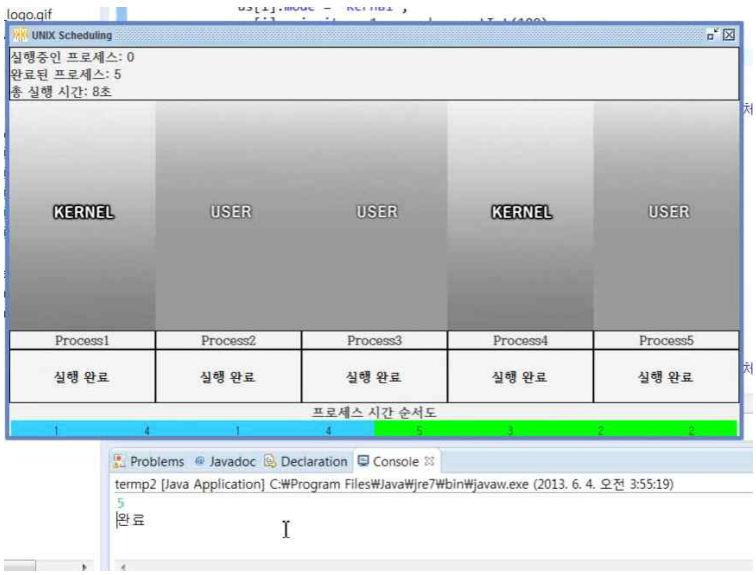
7. 사용자 모드 프로세스도 마찬가지로 실행이 끝나면 각 패널에서 변화된 이미지를 볼 수 있다. 이번 순서에서는 그 다음으로 우선순위가 낮은 Process3을 실행한다.



8. 프로세스가 하나만 남았을 경우엔 최적의 우선순위를 찾는 과정에서 할당 시간이 끝난 프로세스 클래스는 할당 시간이 남은 프로세스 클래스보다 값을 높게 받으므로 결국 마지막 남은 프로세스는 남은 할당 시간을 다 소모할 때까지 연산을 실행한다. 이번 경우엔 마지막으로 남은 Process2를 실행한다. 사용자 모드 프로세스의 기본 우선순위는 60이므로 값은 75가 될 것이다.



8. 그림에서 나타난 것처럼 프로세스 시간 순서도의 이미지는 참고로 연산이 끝나고 적재되는 것이 아니라 연산이 시작될 때 적재되어 프로그램을 실행한 사용자에게 알려준다. 위의 그림은 마지막 단계로써 Process2의 마지막 실행인데 CPUCOUNT가 60이 더해지면 90이 되어 45로 적재 될 것이고 우선순위는 60+23으로 83이 된 후 할당 시간을 모두 소비하여 클래스 속성의 MODE가 “FINISH”로 바뀐다. 또한 이미지에 대한 변경이 이루어진다.



9. 마지막 프로세스까지 할당 시간을 다 채우게 되면 “완료”라는 출력문과 함께 프로그램의 반복문을 빠져나가게 된다. FRAME 창에는 완성된 시간 순서도와 총 실행시간, 나머지 프로세스에 대한 상태 (실행 완료, 실행 중, 완료)를 보여준다.

4. 느낀 점

사실 말하자면 좀 난감했다. 학기 초에 기초적인 개념부분과 운영체제에 관한 알고리즘을 다루던 수업의 전반부에서 운영체제의 개념을 한번도 다뤄보지 않은 GUI로 구현해야한다는 사실은 내게 부담감을 주기 충분했다. 하지만 프로젝트를 완성시키고 글을 쓰는 지금에서 그때의 부담감을 이번 실습을 통해 얻은 성취감으로 덮을만큼 재미있고 의미 있는 시간이었다고 서술하고 싶다. 우선 이번이 아니었더라면 더 늦게 시도해보았을 그래픽 유저 인터페이스를 응용한 코딩에 대해 많은 시간과 고생이 따랐지만 그 덕분에 많은 지식을 얻어갈 수 있었다. 앞서 말했듯이 부족한 실력을 걱정했는데 분명 많은 시간이 걸렸고 설계하는 부분에서 그만큼의 애로사항이 따랐다. 하지만 끈기를 가지고 작업을 진행하니 하나하나 적용 되는 게 보이는 그래픽 때문에 프로젝트에 대한 흥미를 점차적으로 가질 수 있었고 또한 많은 동기 부여를 얻을 수 있었다.실습의 매력은 아무래도 많은 시간을 두고 열심히 한만큼 결과로 볼 수 있다는 것인데 이론을 배우는 것에 그치지 않고 그 개념을 설계를 통한 프로그래밍으로 볼 수 있었기 때문에 정말로 보람 있는 시간이었다고 말하고 싶다.

2009135044 나동희

APPENDIX_SOURCE CODE

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.*;
import java.util.Random;
import java.util.Scanner;
import javax.swing.JPanel;
import javax.swing.ImageIcon;
import javax.swing.BoxLayout;
import javax.swing.SwingConstants;
import javax.swing.border.LineBorder;

class US { //유닉스 스케쥴링 프로세스 클래스
public String mode //모드
public int priority=0; //우선순위
public int time=0; //시간
public int basepriority=0; //기본우선순위
public int cpucount=0; //CPU 카운팅
}


public class OS_2009135044 {
private static Scanner scan
public static void main(String[] args) {
JFrame.setDefaultLookAndFeelDecorated(true);
JFrame frame = new JFrame("UNIX Scheduling"); //프레임 제목
frame.setIconImage(Toolkit.getDefaultToolkit().getImage // 윈도우 창 로고 이미지
("C:₩₩Users₩₩₩uB098₩uB3D9₩uD76C₩₩workspace₩₩gui₩₩kut_logo.gif"));

//
//1. GUI와 LOGIC 구현을 위한 기본틀짜기 및 클래스(배열) 생성
//

Random random = new Random(); //속성 값을 랜덤으로 받기 위한 랜덤값

int a,b,i,j=0;
//a는 프로세스 호출할 수 입력
//b는 프로세스 속성 값에 대해 랜덤 함수를 이용한 임의의 값 부여
//i,j는 FOR문에 대한 변수들.
int finish_count=0; // 프로세스 완료되는 수 카운팅
int time_count=0; // 프로세스가 완료되기까지 걸리는 시간 카운팅
int time_sum=0; // 프로세스가 완료되기까지 걸리는 시간의 합

scan = new Scanner(System.in);
a = scan.nextInt(); //프로세스 호출할 수 입력받는다.

//입력 받은 해당 값만큼 각 항목에 배열을 생성한다
US[] us = new US[a]; //프로세스 클래스
JLabel [] label  = new JLabel[a]; // 프로세스 모드 이미지
JLabel [] label_1  = new JLabel[a];// 프로세스 번호
JPanel [] panel  = new JPanel[a]; // 각 프로세스 속성이 담길 패널
JLabel [][] panel3_inside_label = new JLabel[a][3]; // 각 프로세스 속성을 출력할 라벨

//배열 크기만큼 각 항목을 생성
for (i=0; i<us.lengthi++){
us[i] = new US();
label[i] = new JLabel();
label_1[i] = new JLabel(BorderLayout.CENTER);
label_1[i].setBorder(new LineBorder(new Color(0, 0, 0), 1));
panel[i] = new JPanel();
panel[i].setBorder(new LineBorder(new Color(0, 0, 0), 1));
}

//패널3 (각 프로세스 속성)에 대한 이차원 배열 설정
for (i=0; i<us.lengthi++){
for (j=0; j<3;j++)
panel3_inside_label[i][j] = new JLabel(BorderLayout.CENTER);
}


//
//2. 패널 틀 짜기
//


//그래픽 틀 생성 (콘텐트페인, 패널), 패널은 y축 정렬이 된다.
frame.getContentPane().setLayout(new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));

//패널 1 상단- 프로그램 정보 패널
JPanel panel1 = new JPanel(new BorderLayout());
frame.getContentPane().add(panel1, BorderLayout.CENTER);
panel1.setBorder(new LineBorder(new Color(0, 0, 0), 1));

//패널 2 중단-그림 패널
JPanel panel2 = new JPanel(new BorderLayout());
frame.getContentPane().add(panel2, BorderLayout.CENTER);

//패널 2_0 중단- 프로세스 번호 패널
JPanel panel2_0 = new JPanel(new BorderLayout());
frame.getContentPane().add(panel2_0, BorderLayout.CENTER);
```

```
//패널 3 중단- 프로세스 정보 패널
JPanel panel3 = new JPanel(new BorderLayout());
frame.getContentPane().add(panel3, BorderLayout.CENTER);

//하단 라벨 "프로세스 시간 순서도"
JLabel lblNewLabel_1 = new JLabel("프로세스 시간 순서도");
lblNewLabel_1.setAlignmentX(Component.CENTER_ALIGNMENT);
lblNewLabel_1.setFont(new Font("함초롬바탕", Font.BOLD, 15));
frame.getContentPane().add(lblNewLabel_1);

//패널 4 하단 - 프로세스 시간 순서도 패널
JPanel panel4 = new JPanel(new BorderLayout());
frame.getContentPane().add(panel4);
//패널 4_1 하단 - 프로세스 아이콘 시간별 생성 패널
JPanel panel4_1 = new JPanel(new BorderLayout());
panel4.add(panel4_1);

//윈도우 창의 축소 방지를 위한 라벨 생성
JLabel lblNewLabel = new JLabel("                              ");
frame.getContentPane().add(lblNewLabel);


//
//3. 패널(1~3)까지의 내용 생성 (LOGIC 일부 구현)
//


//상단 패널(panel 1번) 내용 생성 (GUI)
JLabel label5  = new JLabel();
JLabel label6  = new JLabel();
JLabel label7  = new JLabel();
panel1.setLayout(new GridLayout(3,0));
panel1.add(label5);
panel1.add(label6);
panel1.add(label7);
label5.setText("실행 중인 프로세스: "+Integer.toString(a) + "₩n");
label6.setText("완료 된 프로세스: "+Integer.toString(finish_count) + "₩n");
label7.setText("총 실행 시간: "+Integer.toString(time_count) + "₩n");
label5.setFont(new Font("함초롬바탕", Font.BOLD, 15));
label6.setFont(new Font("함초롬바탕", Font.BOLD, 15));
label7.setFont(new Font("함초롬바탕", Font.BOLD, 15));



//중단 패널(panel 2번, 2_0번, 3번) 내용 생성 (GUI, LOGIC)

for (i=0; i<us.lengthi++){ //초기 화면 생성 과정

//중단 그림과 프로세스 번호에 대한 레이어 설정
panel2.setLayout(new GridLayout(0,a));
panel2_0.setLayout(new GridLayout(0,a));

//커널 모드와 유저 모드를 랜덤으로 생성하기 위한 int형 변수 생성
b = random.nextInt(2);

//우선순위와 시간도 랜덤 값으로 받는다.
//어떤 모드냐에 따라 중단에 생성되는 이미지가 다르다.
//더불어 프로세스 번호도 출력.

if(b == 0){ //커널
us[i].mode = "Kernal"
us[i].priority=  1 + random.nextInt(100);
us[i].time =  1 + random.nextInt(3);
us[i].basepriority = 40;

panel2.add(label[i]);
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩2_k.jpg"));

panel2_0.add(label_1[i]);
label_1[i].setText("Process" + (i+1));
label_1[i].setFont(new Font("함초롬바탕", Font.BOLD, 15));
label_1[i].setHorizontalAlignment(SwingConstants.CENTER);
}

else if(b == 1){ //사용자
us[i].mode="User"
us[i].priority=  1 + random.nextInt(100);
us[i].time =  1 + random.nextInt(3);
us[i].basepriority = 60;

panel2.add(label[i]);
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩2_u.jpg"));

panel2_0.add(label_1[i]);
label_1[i].setText("Process" + (i+1));
label_1[i].setFont(new Font("함초롬바탕", Font.BOLD, 15));
label_1[i].setHorizontalAlignment(SwingConstants.CENTER);
}


//중단 패널(panel 3번) 내용 생성 (GUI)
panel3.setLayout(new GridLayout(0,a));
```

```
//중단 패널에 입력한 값만큼 패널을 만들고 그 안에 각 프로세스의 속성 값들을 출력한다.
panel3.add(panel[i]);
panel[i].setLayout(new BoxLayout(panel[i], BoxLayout.Y_AXIS));

panel[i].add(panel3_inside_label[i][0]);
panel[i].add(panel3_inside_label[i][1]);
panel[i].add(panel3_inside_label[i][2]);
panel3_inside_label[i][0].setText("우선순위: "+ Integer.toString(us[i].priority) + "₩n");
panel3_inside_label[i][1].setText("CPUCOUNT: "+ Integer.toString(us[i].cpucount) + "₩n");
panel3_inside_label[i][2].setText("할당 시간: "+ Integer.toString(us[i].time) + "초₩n");
//글씨체 및 문장 위치 설정

panel3_inside_label[i][0].setFont(new Font("함초롬바탕", Font.BOLD, 15));
panel3_inside_label[i][1].setFont(new Font("함초롬바탕", Font.BOLD, 15));
panel3_inside_label[i][2].setFont(new Font("함초롬바탕", Font.BOLD, 15));
panel3_inside_label[i][0].setAlignmentX(Component.CENTER_ALIGNMENT);
panel3_inside_label[i][1].setAlignmentX(Component.CENTER_ALIGNMENT);
panel3_inside_label[i][2].setAlignmentX(Component.CENTER_ALIGNMENT);
//하단 패널의 그래픽 효과를 위해 시간의 합을 구한다.
time_sum=time_sum+ us[i].time
}


//기타 GUI 설정, 사이즈 조절 금지
frame.setLocation(100,350);
frame.pack();
frame.setResizable(false);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


//
//4. 패널4 내용 생성 및 LOGIC을 구현
//   유닉스 스케쥴링 개념을 소스코드화
//   우선순위에 따른 경우를 통해 연산 실행
//   매 단계마다 달라지는 패널들의 구성요소를 통한 시각적인 효과 연출


// 커널 모드인 프로세스의 수 측정
int k=0;
for (i=0; i<a; i++){
if(us[i].mode == "Kernal") {
k++;
}
}

while (finish_count <= a){// 연산 시작

//모든 프로세스가 완료되면 WHILE문 탈출
if (finish_count ==a){
System.out.println("완료");
break
}
//매 시간마다 시간을 카운팅 하여 상단 패널 "총 실행 시간"에 값 출력
time_count++;
label7.setText("총 실행 시간: "+ Integer.toString(time_count) + "초₩n");

//우선 순위를 받을 인자 생성, 우선순위를 가능할 프로세스 개수만큼의 배열 생성
int temp=0;
int  [] order = new int [a];


// 커널 모드인 프로세스가 다 실행될 때까지의 우선순위 구하는 알고리즘
if (finish_count < k){
for (i=0; i<a; i++){
if(us[i].time != 0 && us[i].mode == "Kernal") { //할당 시간이 남은 동시에 '그것이 커널'이면
order[i] = us[i].priority //우선순위 배열에 값을 넣는다.
}
else //시간이 다된 프로세스나 커널이 아닌 프로세스는
order[i] = 1000; // 가지고있는 우선순위 값이 아닌 높은 값을 준다.
}

for (i=0; i<a; i++){ // 우선순위 배열의 최소값을 temp에 담는다.
if (order[temp] > order[i]){
temp=i;
}
}}

//모든 커널 모드 프로세스 실행 이후 유저 모드 프로세스들 사이에서 우선순위 구하는 알고리즘 (설명 생략)
else{

for (i=0; i<a; i++){
if(us[i].time != 0) {
order[i] = us[i].priority
}
else
order[i] = 1000;
}

for (i=0; i<a; i++){
if (order[temp] > order[i]){
temp=i;
```

```
}
}
}


for (i=0; i < a ; i++){ // 프로세스 수만큼 돌리기
if (i == temp && (us[i].cpucount==0 || us[i].cpucount!=0) && us[i].time !=0){
//경우[1]: 우선순위값이 가장 낮은 실행 대상 프로세스, CPUCOUNT=0
// 이미 한번 실행이 되었던 상태일 경우

//하단 패널(panel 4번) 내용 생성 (GUI)
//이 경우는 하단 패널에 대해 시간 순서도를 위한 이미지 추가에 대한 레이어 설정을 해준다.
//총 시간만큼 이미지가 채워지므로 앞서 구한 time_sum만큼 GridLayout 적용
panel4_1.setLayout(new GridLayout(0,time_sum));

JPanel panel4_k  = new JPanel();
JLabel panel_label_4_k  = new JLabel();
panel4_1.add(panel4_k);
panel4_k.add(panel_label_4_k);

// 논리 연산에 따른 속성 값과 그래픽 변화 출력

if (us[i].mode == "Kernal"){
//실행중의 이미지로 변경
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩0_k.jpg"));
//프로세스 번호와 함께 시간에 대한 모드 이미지 하단부 출력
panel4_k.setBackground(new Color(51, 204, 255));
panel_label_4_k.setText(Integer.toString(temp+1));
panel_label_4_k.setFont(new Font("굴림", Font.PLAIN, 12));
}

//유저 모드도 마찬가지로 경우를 적용시킨다.
else if (us[i].mode == "User"){
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩0_u.jpg"));
panel4_k.setBackground(new Color(0,255,0));
panel_label_4_k.setText(Integer.toString(temp+1));
panel_label_4_k.setFont(new Font("굴림", Font.PLAIN, 12));
}

//실행 프로세스에 대한 시각적인 효과 (배경색, 폰트색 변경)
panel[i].setBackground(new Color(255, 255, 153));
panel3_inside_label[i][1].setForeground(new Color(255, 0, 0));
panel3_inside_label[i][0].setForeground(new Color(255, 0, 0));
panel3_inside_label[i][2].setForeground(new Color(255, 0, 0));

//실행 프로세스에 대한 시각적인 효과 (CPU COUNT 값 증가)
for (int x=0;x<60;x++){
us[i].cpucount = us[i].cpucount+1;
panel3_inside_label[i][1].setText("CPUCOUNT: "+Integer.toString(us[i].cpucount) + "₩n");
try{
Thread.sleep(18);
}
catch(InterruptedException ignore){}
}

//유닉스 스케쥴링 논리 연산 실행
us[i].cpucount = us[i].cpucount/2;
us[i].priority = us[i].basepriority +us[i].cpucount/2;
us[i].time -= 1;
//실행 프로세스에 대한 시각적인 효과 (논리 연산 결과 값 출력)
panel3_inside_label[i][0].setText("우선순위: "+Integer.toString(us[i].priority) + "₩n");
panel3_inside_label[i][2].setText("할당 시간: "+Integer.toString(us[i].time) + "₩n");
panel3_inside_label[i][1].setText("CPUCOUNT: "+Integer.toString(us[i].cpucount) + "₩n");

//연산이 끝나면 실행 대기 상태의 이미지를 다시 출력한다.
if (us[i].mode == "Kernal"){
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩2_k.jpg"));
}
else if (us[i].mode == "User"){
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩2_u.jpg"));
}

//시간을 할당하는 이 연산의 경우 중, 프로세스가 완료되는 (시간이 0이 되는) 시점이 있다면
if (us[i].time == 0)
{
finish_count++; // 완료된 프로세스의 수를 카운팅

//패널 상단부 프로그램이 정보 업데이트
label5.setText("실행중인 프로세스: "+Integer.toString(a-finish_count) + "₩n");
label6.setText("완료된 프로세스: "+Integer.toString(finish_count) + "₩n");

//패널 중단부 프로세스에 속성에 대한 값 변경
panel3_inside_label[i][1].setText("실행 완료");
panel3_inside_label[i][0].setText(" ");
panel3_inside_label[i][2].setText(" ");


//실행이 완료된 프로세스에 대한 이미지 변경
if (us[i].mode == "Kernal")
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩4_k.jpg"));
else
label[i].setIcon(new ImageIcon("D:₩₩대학 생활₩₩3학년 1학기₩₩OS(운영체제)₩₩4_u.jpg"));
}
```

```
panel[i].setBackground(new Color(240, 240, 240));
panel3_inside_label[i][1].setForeground(new Color(0,0,0));
panel3_inside_label[i][0].setForeground(new Color(0,0,0));
panel3_inside_label[i][2].setForeground(new Color(0,0,0));
}

else if (i != temp && us[i].cpucount == 0 && us[i].time !=0){//2. 연산대상은 아니고, 실행도 안되었을 때

//변경 값이 없으므로 그대로 간다.
try{
Thread.sleep(1);
}
catch(InterruptedException ignore){}
}

else if(i != temp && us[i].cpucount != 0 && us[i].time !=0){//3. 연산대상은 아니지만, 이미 실행이 된 경우

us[i].cpucount = us[i].cpucount/2;//CPUCOUNT 값 증가 없이 1/2 연산 실행
panel3_inside_label[i][1].setText("CPUCOUNT: "+ Integer.toString(us[i].cpucount) + "₩n");

us[i].priority = us[i].basepriority +us[i].cpucount/2;
panel3_inside_label[i][0].setText("우선순위: "+ Integer.toString(us[i].priority) + "₩n");
try{
Thread.sleep(1);
}
catch(InterruptedException ignore){}

}
try{
Thread.sleep(1);
}
catch(InterruptedException ignore){}
}
} // 부분 FOR문이 끝나면 딜레이

try{
Thread.sleep(1);
}
catch(InterruptedException ignore){} //전체 FOR문이 끝나면 딜레이
}//연산 끝
} //main, class
```