DKU 단국대학교 DANKOOK UNIVERSITY

REPORT

운영체제

과 목 : 운영체제
이 름 : 김민수
학 번 : 32197083
과제정보: 스케줄링 시뮬레이터
제출일자: 240409

운영체제 - RoundRobin 구현하기 (1)

요구사항 정의

운영체제 과제로 스케줄링 알고리즘을 구현하는 과제가 주어졌다. 정의된 요구사항은 다음과 같았다.

1. 프로세스의 생성과 관리
    a. 프로세스 모델링
        i. 파이썬의 클래스를 이용
        ii. 프로세스는 실행 시간, 상태, 우선순위 등의 프로세스의 기본적인 형태를 가지고 있다
2. 스케줄러의 구현
    a. 프로세스 큐를 사용하여 관리
    b. 현재 실행할 프로세스를 결정하는 스케줄러 구현
    c. 지정된 시간 동안에만 실행 및 시간 종료 시 다음 프로세스로 교체
3. 메인 실행 루프 및 프로세스 생성
    a. 생성한 모든 프로세스가 완료될 때까지 시뮬레이션 루프 실행(while)
    b. 각 루프에서는 스케줄러가 다음 프로세스를 선택하고 실행 상태 업데이트 (Ready - Running - Finish)

구현하고자 하는 스케줄링 알고리즘은 라운드 로빈(Round Robin)이다. 라운드 로빈은 프로세스들 사이에 우선순위를 두지 않고, 여러 프로세스가 CPU 사용시간을 일정하게 할당하는 방식이다. 과정은 다음과 같다.

1. 프로세스를 생성하여 준비 큐에 append, 타임 슬라이스는 2로 설정
2. 주어진 준비큐에서 프로세스를 하나씩 꺼내면서 실행(CPU 할당, state 변경)
3. 할당된 프로세스는 주어진 실행시간에서 타임슬라이스만큼 실행시간을 빼기
4. 프로세스의 상태변화 진행 및 준비 큐 append 진행
5. 2-4를 반복작업 진행
6. 만약 프로세스의 실행 시간이 0 이하로 내려가면 준비큐에 append x

구현

먼저 클래스이다. 요구사항에 맞춰서 클래스로 만들어서 진행을 하였으며 상태값은 Enum으로 지정하였다. 프로세스의 경우 프로세스의 상태변화를 위한 함수들과 실행시간 변경을 위한 함수, 그래프로 작성을 위한 함수로 구성되어있다. 그래프 생성을 위한 시간 측정을 위해 상태 변화의 경우 time.sleep(0.1)로 딜레이를 넣어줬다.

```
class ProcessState(Enum):
    RUNNING = "Running"
    READY = "Ready"
    FINISHED = "Finished"


class Process:
    def __init__(self, processId, executionTime, priorityNum)
        self.processId = processId  # 프로세스 식별자
        self.executionTime = executionTime  # 실행 시간
        self.priorityNum = priorityNum  # 우선순위
        self.state = ProcessState.READY  # 초기 상태는 준비(Read

    def changeStateRunning(self):
        self.state = ProcessState.RUNNING
        time.sleep(0.1)
```

```
    def changeStateReady(self):
        self.state = ProcessState.READY
        time.sleep(0.1)


    def changeStateFinished(self):
        self.state = ProcessState.FINISHED
        time.sleep(0.1)


    def changeExecuteTime(self, timeslice):
        self.executionTime = self.executionTime - timeslice

    def record_state(self, time):
        process_state_records.append((self.processId, self.st
```

다음은 프로세스 생성 함수이다. 준비 큐에 각 프로세스를 생성해서 넣어주기위해 for 반복문으로 작성되었다. 이때 실행시간의 차이를 두기 위해 i가 짝수일때와 홀수인 경우로 나눠서 실행시간의 변화를 주었다.

```
# 프로세스 리스트 생성
def makeProcessors(ReadyQueue):
    for i in range(1, 10, 1):
        if (i % 2 == 0):
            execTime = i * i
        else:
            execTime = i
        process = Process(processId= i, executionTime=execTim
```

```
            ReadyQueue.append(process)
```

다음은 라운드 로빈 프로세스 스케줄링의 구현이다. 구현은 간단하다.

1. while 반복문을 통해 readyQueue에서 프로세스가 다 소모될때까지 반복한다.
2. 라운드 로빈에 맞게 로직 구현
    a. 이때 상태 및 성능을 확인할 수 있도록 상태변화가 일어나는 순간에 해당 내용을 record_state를 사용하여 상태를 저장

```
def startRoundRobin(readyQueue):
    timeSlice = 2  # 2,10 등의 사용자 지정 가능
    time_count = 0 # 최종 시간 확인을 위한 변수
    while readyQueue:
        process = readyQueue.popleft()
        process.changeStateRunning()
        process.record_state(time_count)
        time_count += timeSlice
        if process.executionTime <= timeSlice:
            process.executionTime = 0
            process.changeStateFinished()
            process.record_state(time_count)
        else:
            process.changeExecuteTime(timeSlice)
            process.changeStateReady()
            process.record_state(time_count)
            readyQueue.append(process)
```

다음은 성능 및 동작을 확인할 수 있도록 그래프를 만드는 과정이다. matplotlib를 사용해서 구현하였으며, x축은 시간을, y축은 각 프로세스의 ID를 나타내도록 하였다. 또한 state마다의 색을 다르게 배정하여 그래프를 볼 시 좀 더 직관적으로 보이도록 하였다.

```
def plot_process_states(size_process):
    # 각 프로세스의 실행 상태를 그래프로 표현
    # time의 경우 각 라운드로빈의 timeslice 시간 사용
    for processId, state, time in process_state_records:
        plt.plot([time, time + 2], [processId, processId],
                 color='blue'
                 if state == 'Running' else 'green' if state =
                 linewidth=3)

    plt.xlabel('Time')
    plt.ylabel('Process ID')
    plt.title('Process Execution States')
    plt.yticks(range(1, size_process + 1), [f'P{i}' for i in
    plt.grid(axis='x')
    plt.legend()
    plt.show()
```

전체 코드

```
'''
OS -1
파이썬을 이용하여 프로세스 스케줄링 및 멀티 태스킹의 기본 시뮬레이션

기본 요구사항

1. 프로세스의 생성과 관리
2. 스케줄러의 구현
3. 메인 실행 루프 및 프로세스 생성

구현 가이드

1. 프로세스 모델링
    - 파이썬 클래스를 사용하여 프로세스 모델링
```

```
    - 프로세스는 실행 시간, 상태, 우선순위 등의 속성을 가짐

2. 스케줄러 구현
    - 프로세스 큐를 관리
    - 현재 실행할 프로세스를 결정하는 스케줄러 구현
    - 지정된 시간 동안만 실행 및 시간 종료 시 다음 프로세스로 교체

3. 시뮬레이션 루프
    - 모든 프로세스가 완료될 때까지 시뮬레이션 루프 실행
    - 각 루프에서는 스케줄러가 다음 프로세스를 선택하고 실행 상태 업데이트



라운드 로빈
프로세스들 사이에 우선순위를 두지않고, 여러 프로세스가 CPU 사용시간을 일정하
1. 준비큐 : 프로세스들이 들어있는 큐
2. 타임 슬라이스 : CPU가 사용할 수 있는 시간, 해당 타임 슬라이스는 2로 지
3. CPU 할당 : 현재 실행중인 타임 슬라이스가 끝나면, CPU는 준비큐에서 다음
4. 1-3 반복
'''
import random
import time
from enum import Enum
from collections import deque
import matplotlib.pyplot as plt

process_state_records = []

class ProcessState(Enum):
    RUNNING = "Running"
    READY = "Ready"
    FINISHED = "Finished"


class Process:
    def __init__(self, processId, executionTime, priorityNum)
        self.processId = processId  # 프로세스 식별자
        self.executionTime = executionTime  # 실행 시간
        self.priorityNum = priorityNum  # 우선순위
```

```
        self.state = ProcessState.READY  # 초기 상태는 준비(Ready

    def changeStateRunning(self):
        self.state = ProcessState.RUNNING
        time.sleep(0.1)


    def changeStateReady(self):
        self.state = ProcessState.READY
        time.sleep(0.1)


    def changeStateFinished(self):
        self.state = ProcessState.FINISHED
        time.sleep(0.1)


    def changeExecuteTime(self, timeslice):
        self.executionTime = self.executionTime - timeslice

    def record_state(self, time):
        process_state_records.append((self.processId, self.st


# 프로세스 리스트 생성
def makeProcessors(ReadyQueue):
    for i in range(1, 10, 1):
        if (i % 2 == 0):
            execTime = i * i
        else:
            execTime = i
        process = Process(processId= i, executionTime=execTim
        ReadyQueue.append(process)

# 라운드 로빈 함수
def startRoundRobin(readyQueue):
    timeSlice = 2
    time_count = 0
```

```
    while readyQueue:
        process = readyQueue.popleft()
        process.changeStateRunning()
        process.record_state(time_count)
        time_count += timeSlice
        if process.executionTime <= timeSlice:
            process.executionTime = 0
            process.changeStateFinished()
            process.record_state(time_count)
        else:
            process.changeExecuteTime(timeSlice)
            process.changeStateReady()
            process.record_state(time_count)
            readyQueue.append(process)

# 그래프 생성
def plot_process_states(size_process):
    # 각 프로세스의 실행 상태를 그래프로 표현
    for processId, state, time in process_state_records:
        plt.plot([time, time + 2], [processId, processId],
                 color='blue'
                 if state == 'Running' else 'green' if state
                 linewidth=3)

    plt.xlabel('Time')
    plt.ylabel('Process ID')
    plt.title('Process Execution States')
    plt.yticks(range(1, size_process + 1), [f'P{i}' for i in
    plt.grid(axis='x')
    plt.legend()
    plt.show()

# 시뮬레이션 동작 실행
def startRoundRobinWorking():
    print("Processor simulation is start now")  # Press ⌘F8 t
    # ready, finish 큐 구현
    readyQueue = deque([])
    #
```

```
    makeProcessors(readyQueue)
    print("check about processor")
    size_process = len(readyQueue)

    for i in readyQueue:
        print(i.processId, i.executionTime)

    print("We make processes and we start round robin schedul
    startRoundRobin(readyQueue)
    plot_process_states(size_process)



# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    startRoundRobinWorking()
# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

실행 결과

시뮬레이션 환경

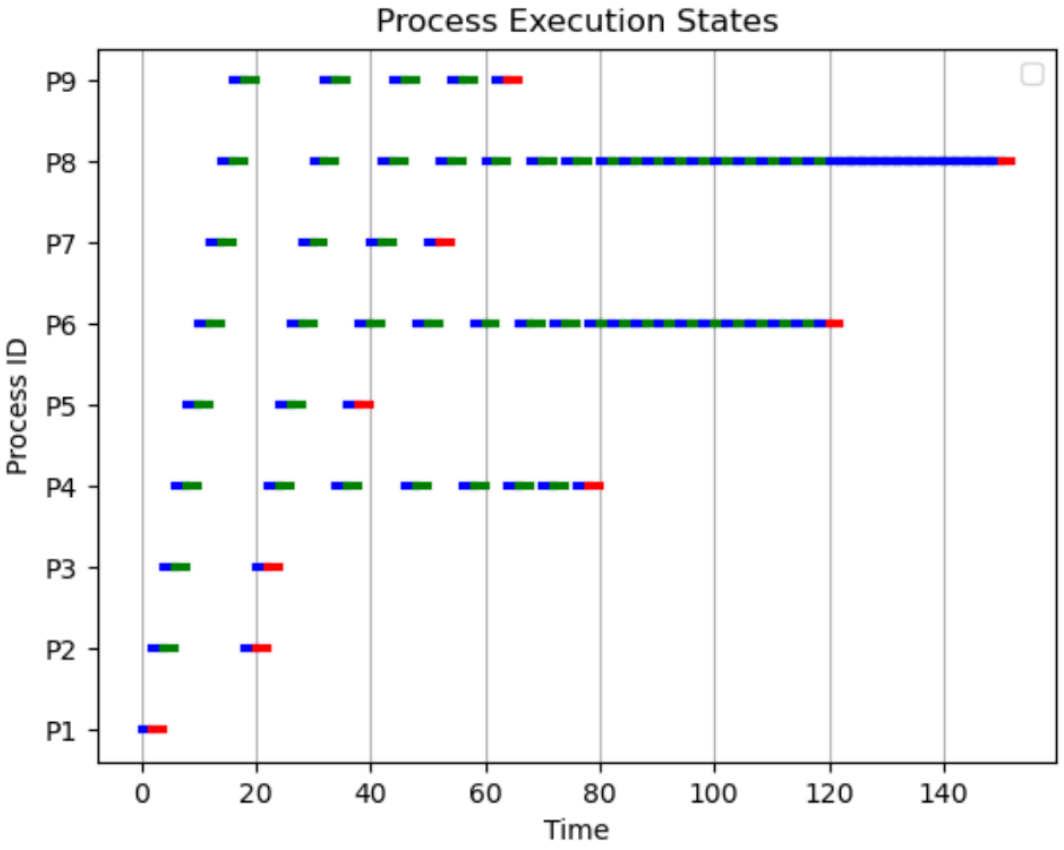
- **프로세스 수:** 9개
- **스케줄링 알고리즘:** 라운드로빈
- TimeSlice : 2, 10

각 프로세스는 주어진 2와 10의 시간동안만 동작하고 다음 문맥교환을 가지게 된다. 이때 파란색은 실행, 그리고 초록색은 대기, 빨간색은 종료를 의미한다. 종료의 경우에도 표기를 위해 2초의 결과값을 가지고 진행이 된다. 그리고 초록색으로 표기 된 대기의 상태는 뒤의 파란색으로 변하기 전까지 표시가 되어있지않아도 지속해서 대기상태로 되어있는 상태이다.

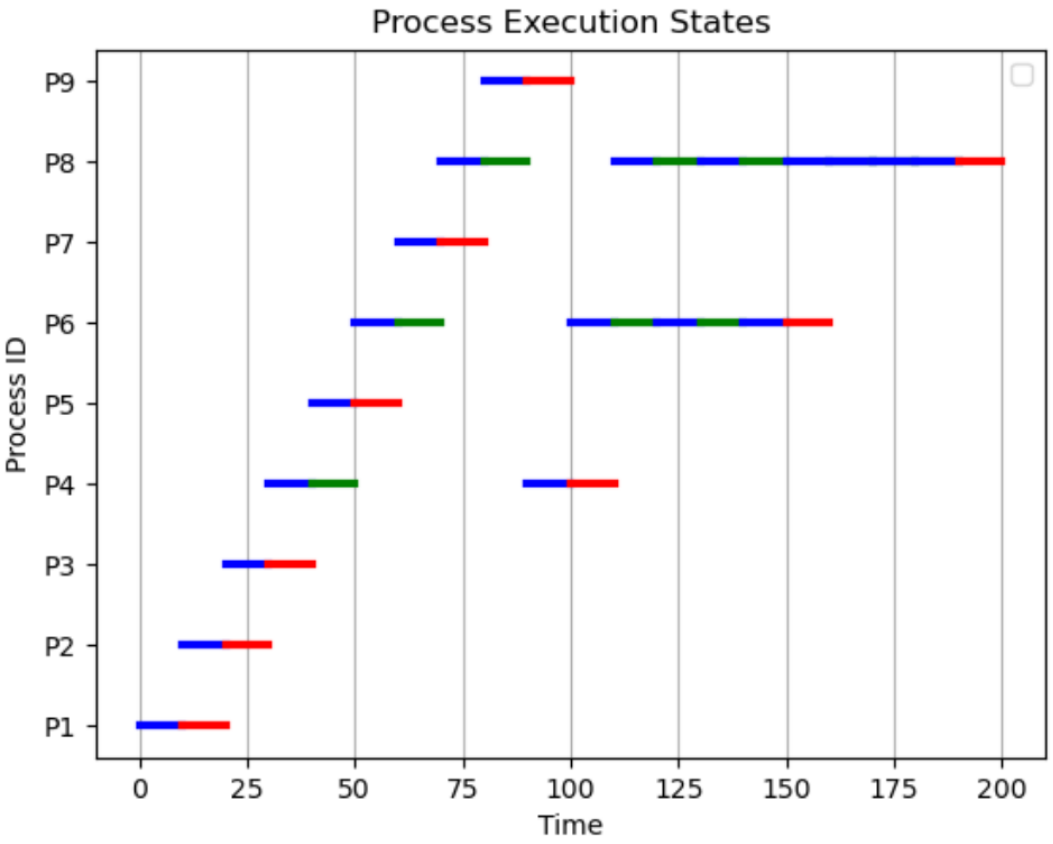
라운드로빈 스케줄링 알고리즘은 결론적으로 모든 프로세스에게 동일한 실행 기회를 제공한다. 다만 CPU를 할당 받을 수 있는 TimeSlice를 짧게 설정하면 모든 프로세스가 자주 실행될 수 있지만, 긴 실행 시간을 가진 프로세스는 지속하여 벌어지는 문맥교환으로 인해 마지

막까지 남게된다. 그러나 TimeSlice를 길게 설정하면 긴 실행 시간을 가진 프로세스가 완료될 수 있지만, 짧은 실행 시간을 가진 프로세스를 실행 후 종료까지 자원이 낭비될 수 있다.

- 2



- 10



실제로 2와 10을 timeslice로 잡은 결과 그래프이다. 2의 경우 설정된 timeslice가 작아서 잦은 문맥교환(컨텍스트 스위칭 오버헤드)을 진행하지만 최종적으로 모든 프로세스가 종료가 되는 시간은 150 이전에 끝나게 된다. 다만 10으로 TimeSlice를 잡으면 문맥교환은 2인 경우에 비해 적게 발생하지만 최종적으로 끝나는 시간은 175와 200 사이에서 종료가 된다. 이때 시간 할당량이 10인 경우에 P9의 경우를 보자 p9의 경우 실행 및 종료시간이 다른 프로세스에 비해 짧은 상황이다. 하지만 많은 프로세스가 존재를 하고 이때 시간의 할당량도 길게 가져갔기때문에 대기에서 실행까지 긴 대기 시간이 발생하는 것을 볼 수 있었다.

그렇기에 라운드 로빈의 경우 문맥교환의 비용과 종료 시간의 최적화를 위해 timeSlice를 지정하는 것이 해당 스케줄링 알고리즘의 큰 관건이 될 것이라 판단한다.