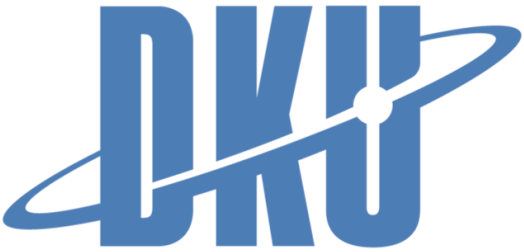
Report

운영체제 3분반

1번 과제: 멀티태스킹 운영체제 시뮬레이터



이름	이창현
학번	32193605
학과	소프트웨어학과
과목	운영체제 3분반
교수명	김하연
날짜	2024-4-16
이메일	32193605@dankook.ac.kr
전화	01085319558

프로젝트 설계

파이썬을 이용해서 프로세스 스케줄링과 멀티태스킹 시뮬레이션 프로젝트를 설계했다.
클래스로는 실행할 가상의 프로세스를 생성하는 프로세스 클래스와 생성한 프로세스를 실행하고 스케줄링 할 스케줄러 클래스를 설계했다.
프로젝트의 기본 요구사항에 맞게 프로세스 클래스에서는 프로세스 생성 및 관리를 한다.
이때 사용자가 정의한 간단한 태스크로는 1번 반복할 때마다 1씩 더하는 계산작업을 반복하는 함수와 파일 다운로드 시뮬레이션을 함수로 정의한다.
이 때 각 프로세스는 고유한 id와 상태(ready, running, complete)를 가지고 우선순위를 가진다.
스케줄러 클래스에서는 Round Robin 스케줄링 알고리즘을 사용하여 CPU시간(time_slice)을 인자로 받아 프로세스 간 실행시간을 배분하여 일정시간 동안 각 프로세스를 실행(running 상태)하고 배분된 시간이 다 지나고 프로세스가 완료되지 않았다면 큐에 다시 넣고 다음 프로세스를 실행한다. 이 때 프로세스는 ready상태이다. 만약 프로세스가 완료되었다면 complete 상태가 된다.
만약 프로세스가 진행 중에 keyboard interrupt가 발생한다면 프로세스는 즉시 ready상태가 되고 입력이 끝날 때까지 ready상태를 유지한다. 입력이 끝나면 프로세스를 이어서 진행한다.
이러한 프로세스의 실행 상태와 과정을 텍스트 기반으로 시각화 한다.

주요 구현 사항

```
import threading
import time
import queue


class Process:
    def __init__(self, id, task, priority, size=None):
        self.id = id
        self.task = task
        self.priority = priority
        self.size = size
        self.execution_count = 0
        self.state = "ready"
```

threading, time, queue 라이브러리를 사용한다.
process 클래스의 생성자로 process클래스는 고유한 id와 어떤 작업을 수행할지 그리고 우선순위를 인자로 받는다. 파일 다운로드 작업을 수행할 때는 인자로 파일의 크기(다운로드 시간)을 인자로 받는다. 그리고 처음 생성 될 때 프로세스는 실행횟수가 0회이고 프로세스의 state는 대기상태이기 때문에 ready로 초기화한다.

```
def __lt__(self, other):
    if self.execution_count == other.execution_count:
        return self.priority < other.priority
    else:
        return self.execution_count < other.execution_count
```

프로세스가 우선순위에 따라 실행될 수 있도록 한다. 이 때 실행횟수를 고려하여 우선순위가 가장 높은 프로세스가 연속하여 실행되는 것을 방지한다.(만약 중간에 새로운 프로세스를 추가하는 기능이 추가된다면 이 함수는 수정이 필요함 -> 현재는 그런 기능이 없다고 가정했음)

```
def million_loop(self, time_slice):
    count = 0
    total = 0
    start_time = time.time()
    try:
        with open(f"{self.id}.txt", 'r') as file:
            total = int(file.read())
    except FileNotFoundError:
        with open(f"{self.id}.txt", 'w+') as file:
            file.write(str(total))
    while True:
        if scheduler.paused:  # 일시 중지 상태 확인
            print(f"process{self.id} paused")
            print(f"process{self.id} state: ready")
            while scheduler.paused:
                time.sleep(0.1)  # 일시 중지 상태가 해제될 때까지 대기
            print(f"process{self.id} resumed")
            print(f"process{self.id} state: running")
            self.state = "running"
            start_time = time.time()  # 시간을 재설정하여 정확한 시간을 유지

        if total + count > 10000000:
            self.state = "complete"
            print("Repeating complete!!")
            print(f"process{self.id} state: complete")
            break
        count += 1
        if time.time() >= start_time+time_slice:
            total = total + count
            self.state = "ready"
            print(f"{total}times repeats")
            print(f"CPU time over - process{self.id} state: ready")
            with open(f"{self.id}.txt", 'w') as file:
                file.write(str(total))
            break
```

사용자가 정의한 작업 함수 중 하나인 million_loop

time_slice(CPU time)를 인자로 받아 1000만번 반복하는 while문을 통해 어떠한 계산 작업이 진행되고 있음을 가정하였다.(실제로 팩토리얼 계산이나 복잡한 계산을 수행하려 했으나 math 라이브러리를 사용하면 ms 단위로 작업이 끝나버리게 되고, 직접 반복문을 통해 구현하는 것은 단순 반복 계산과 다를 것이 없다고 생각하여 가장 간단한 반복 계산 함수를 설계하게 되었음)

파이썬에서 함수를 실행 중에 중단하는 것은 직접적으로 지원되지 않는다. 파이썬은 함수가 시작되면 해당 함수가 완료되거나 예외가 발생할 때까지 실행을 계속하지만 Round Robin 알고리즘에서는 함수 중단이 필연적이다. 따라서 이를 구현하기 위해 try except를 이용해서 txt파일을 작성해서 함수가 중단될 때 txt파일에 진행상황을 기록하고 다시 실행될 때 이 파일을 읽어와 중단된 시점부터 이어서 프로세스가 진행되는 것으로 구현하였다. 이

함수가 호출되면 프로세스 state를 running으로 바꾸고 time_slice 시간만큼 프로세스를 실행하고 프로세스가 완료되지 않았다면 프로세스 state를 ready로 바꾸고 프로세스가 완료되었다면 프로세스 state를 complete로 바꾼다.
이 때 keyboard interrupt가 발생하면 즉시 프로세스 state를 ready로 바꾸고 프로세스를 일시중지 한다. 이 후 입력이 끝나면 다시 프로세스 state를 running으로 바꾸고 이어서 실행한다.

```python
def file_download_simulation(self, size, time_slice):
    print(f"process{self.id} state: running")
    elapsed_time = 0
    total = 0
    start_time = time.time()
    try:
        with open(f"{self.id}.txt", 'r') as file:
            total = float(file.read())
    except FileNotFoundError:
        with open(f"{self.id}.txt", 'w+') as file:
            file.write(str(total))
    while True:
        if scheduler.paused:  # 일시 중지 상태 확인
            print(f"process{self.id} paused")
            print(f"process{self.id} state: ready")
            while scheduler.paused:
                time.sleep(0.1)  # 일시 중지 상태가 해제될 때까지 대기
            print(f"process{self.id} resumed")
            print(f"process{self.id} state: running")
            self.state = "running"
            start_time = time.time()  # 시간을 재설정하여 정확한 시간을 유지

        if (total + elapsed_time) / size > 1:
            self.state = 'complete'
            print("Download complete!!")
            print(f"process{self.id} state: complete")
            break
        elapsed_time = time.time() - start_time
        if elapsed_time >= time_slice:
            total = total + elapsed_time
            self.state = "ready"
            print(f"{total/size*100}% downloading...")
            print(f"CPU time over - process{self.id} state: ready")
            with open(f"{self.id}.txt", 'w') as file:
                file.write(str(total))
            break
```

사용자가 정의한 작업 함수 중 하나인 file_download_simultaion
size(파일의 크기 -> 이 프로젝트에서는 파일 다운로드 시간)과 time_slice를 인자로 받아 파일이 다운로드 되고 있는 상황을 가정하였다.
time 라이브러리를 활용하여 파일 다운로드 진행상황을 나타낸다.
million_loop함수와 마찬가지로 Round Robin 알고리즘에 필요한 프로세스 흐름이 구현되어 있다.

```python
class Scheduler:
    def __init__(self, time_slice):
        self.process_queue = queue.PriorityQueue()
        self.time_slice = time_slice
        self.time = 0
        self.paused = False

    def pause(self):
        self.paused = True

    def resume(self):
        self.paused = False

    def add_process(self, process):
        self.process_queue.put(process)

    def start(self):
        while not self.process_queue.empty():
            while self.paused:  # 일시 중지 상태일 때는 대기
                time.sleep(0.1)
            process = self.process_queue.get()
            if process.state != "complete":
                if process.task == 'million_loop':
                    process.million_loop(self.time_slice)
                elif process.task == 'file_download_simulation':
                    process.file_download_simulation(
                        process.size, self.time_slice)
                process.execution_count += 1
                if process.state != "complete":
                    self.add_process(process)
```

스케줄러 클래스에서는 time_slice를 인자로 받고, pause, resume함수를 이용해서 keyboard interrupt 발생 시에 paused를 True와 False로 바꿔준다. add_process함수가 호출되면 queue 라이브러리를 이용해서 우선순위 큐를 사용해서 우선순위를 기반으로 프로세스를 큐에 넣는다. start 함수가 호출되면 프로세스 큐가 비어 있지 않다면 큐에서 프로세스를 가져오고 프로세스 state가 complete가 아니라면 해당 task맞는 함수를 호출해서 작업을 수행하고 실행횟수를 1회 더한다. 그리고 프로세스의 state가 complete가 아니라면 다시 큐에 넣어준다.

```python
def listen_for_input(scheduler):
    while True:
        input("Press the enter key to pause the process\n")

        # 스케줄러가 이미 일시 중지 상태인 경우 재개
        if scheduler.paused:
            process.state = 'running'
            scheduler.resume()
        else:
            scheduler.pause()
            process.state = 'ready'
            print("Keyboard interrupt!!\n")
            input("Press the enter key to resume the process")
            scheduler.resume()
```

keyboard interrupt가 발생하는 상황을 가정한 함수

enter키를 입력하면 keyboard interrupt가 발생했다고 가정하고 즉시 프로세스를 중단하고 입력을 받는다. 다시 enter키를 입력하면 입력이 끝났다고 가정하고 중단된 프로세스를 이어서 실행한다.

```python
if __name__ == "__main__":
    time_slice = 1
    scheduler = Scheduler(time_slice)
    processes = [
        Process(0, task='million_loop', priority=3),
        Process(1, task='file_download_simulation', priority=1, size=10.5),
        Process(2, task='million_loop', priority=2),
        Process(3, task='file_download_simulation', priority=2, size=7)
    ]

    for process in processes:
        scheduler.add_process(process)

    input_thread = threading.Thread(target=listen_for_input, args=(scheduler,))
    input_thread.start()

    scheduler_thread = threading.Thread(target=scheduler.start)
    scheduler_thread.start()
    scheduler_thread.join()
```

time_slice를 1로 초기화해주고 4개의 프로세스를 직접 선언하여 해당 프로그램을 실행했다.

실행 결과 스크린샷

```
Press the enter key to pause the process
process1 state: running
9.529231843494234% downloading...
CPU time over - process1 state: ready
process2 state: running
1425892times repeats
CPU time over - process2 state: ready
process3 state: running
14.2970187323434% downloading...
CPU time over - process3 state: ready
process0 state: running

Keyboard interrupt!!
process0 paused

process0 state: ready

Press the enter key to resume the process
```

프로세스는 정의한 우선순위에 따른 순서로 실행된다.

프로세스 실행되고 enter키를 입력하면 state가 ready로 바뀌고 실행이 중단된다.

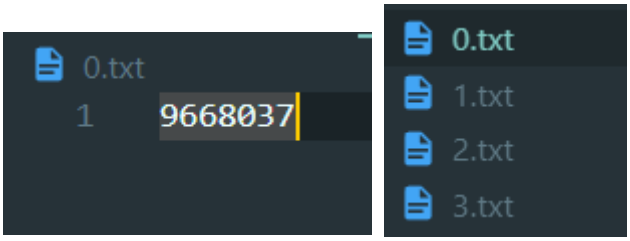
```
process0 state: ready

Press the enter key to resume the process
Press the enter key to pause the process
process0 resumed
process0 state: running
2772174times repeats
CPU time over - process0 state: ready
process1 state: running
19.057991391136532% downloading...
CPU time over - process1 state: ready
process2 state: running
2866787times repeats
CPU time over - process2 state: ready
process3 state: running
28.58326094491141% downloading...
CPU time over - process3 state: ready
process0 state: running
4074378times repeats
```

다시 enter키를 입력하면 state가 running으로 바뀌고 이어서 실행된다.

```
process3 state: running
Download complete!!
process3 state: complete
process0 state: running
Repeating complete!!
process0 state: complete
process1 state: running
76.23018991379512% downloading...
CPU time over - process1 state: ready
process2 state: running
Repeating complete!!
process2 state: complete
process1 state: running
85.75972602480934% downloading...
CPU time over - process1 state: ready
process1 state: running
95.2861854008266% downloading...
CPU time over - process1 state: ready
process1 state: running
Download complete!!
process1 state: complete
```

작업이 완료되어 프로세스 state가 complete가 되면 완료된 프로세스는 더 이상 실행되지 않는다.



프로세스의 진행상황을 기록하는 txt파일이 생성된다.