**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

ФАИС

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

по дисциплине «**Операционные системы и среды**»

на тему: «**Планирование процессов**»

Выполнил: студент гр. ИП-32

Прокопенко А. Р.

Принял: преподаватель

Процкая М. А.

Дата сдачи отчета: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

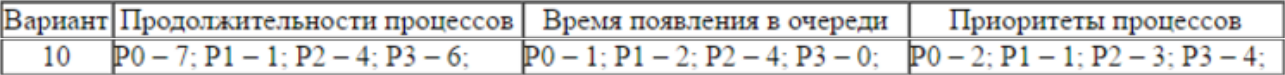
Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

**Цель**: изучить типовые алгоритмы планирования процессов.

**Задание**



1. **Не вытесняющие алгоритмы планирования процессов.**

Выполнить различные алгоритмы планирований – First-Come, First-Served (FCFS) (прямой и

обратный), Round Robin (RR), Shortest-Job-First (SJF) (не вытесняющий), Shortest-Job-First

(SJF) (не вытесняющий приоритетный) для данных приведенных в таблице 2.1 в

соответствии со своим вариантом (номер по журналу). Вычислить полное время выполнения

все процессов и каждого в отдельности, время ожидание для каждого процесса. Рассчитать

среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде

таблиц иллюстрирующих работу процессов.

1. **Вытесняющие алгоритмы планирования процессов.**

Выполнить различные алгоритмы планирований – Shortest-Job-First (SJF) (вытесняющий) и Shortest-Job-First (SJF) (приоритетный) для данных приведенных в таблице 2.1 в соответствии со своим вариантом. Вычислить полное время выполнения все процессов и каждого в отдельности, время ожидание для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц иллюстрирующих работу процессов.

1. **Программная реализация алгоритмов задания 1 и 2.**

Разработать программную реализацию алгоритмов задания 1 и 2.

**Выполнение**

1. **Не вытесняющие алгоритмы планирования процессов.**

First-Come, First-Served (FCFS) (прямой):



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Процессы | |  | Время ожидания | | | |  | Полное время выполнения Время появления в очереди | | | | | | | | | | |
| P0 |  |  |  | 5 |  |  |  |  |  | 7 |  |  |  |  |  | 1 |  |  |
| P1 |  |  |  | 11 |  |  |  |  |  | 1 |  |  |  |  |  | 2 |  |  |
| P2 |  |  |  | 10 |  |  |  |  |  | 4 |  |  |  |  |  | 4 |  |  |
| P3 |  |  |  | 0 |  |  |  |  |  | 6 |  |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Время | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| P0 |  | ГГГГГИИИИИИИ | | | | | | | | | | | |  |  |  |  |  |
| P1 |  |  | ГГГГГГГГГ | | | | | | | | | ГГИ | | |  |  |  |  |
| P2 |  |  |  |  | ГГГГГГГГГГИИИИ | | | | | | | | | | | | | |
| P3 | И | И | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |

«И» - исполняется, «Г» - в ожидании готовности, « » - завершен или не начат.

Полное время выполнения всех процессов: 7 + 1 + 4 + 6 = 18.

Среднее время выполнения процесса: 18 / 4 = 4.5.

Среднее время ожидания процесса: (5 + 11 + 10 + 0) / 4 = 6.5.

First-Come, First-Served (FCFS) (обратный):



|  |  |  |  |
| --- | --- | --- | --- |
| Процессы | Время ожидания | Полное время выполнения Время появления в очереди | |
| P0 | 3 | 7 | 2 |
| P1 | 3 | 1 | 1 |
| P2 | 0 | 4 | 0 |
| P3 | 8 | 6 | 4 |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Время | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| P0 |  |  | ГГГИИИИИИИ | | | | | | | | | |  |  |  |  |  |  |
| P1 |  | Г | Г | Г | И |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P3 |  |  |  |  | ГГГГГГГГИИИИИИ | | | | | | | | | | | | | |

«И» - исполняется, «Г» - в ожидании готовности, « » - завершен или не начат.

Полное время выполнения всех процессов: 7 + 1 + 4 + 6 = 18.

Среднее время выполнения процесса: 18 / 4 = 4.5.

Среднее время ожидания процесса: (3 + 3 + 0 + 8) / 4 = 3.5.

Round Robin (RR):



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Процессы | |  | Время ожидания | | | |  | Полное время выполнения Время появления в очереди | | | | | | | | | | | |
| P0 |  |  |  | 10 |  |  |  |  |  | 7 |  |  |  |  |  | 1 |  |  |  |
| P1 |  |  |  | 4 |  |  |  |  |  | 1 |  |  |  |  |  | 2 |  |  |  |
| P2 |  |  |  | 9 |  |  |  |  |  | 4 |  |  |  |  |  | 4 |  |  |  |
| P3 |  |  |  | 7 |  |  |  |  |  | 6 |  |  |  |  |  | 0 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Время | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |  |
| P0 |  | ГГИИИГГГГГГГИИИГИ | | | | | | | | | | | | | | | | |  |
| P1 |  |  | Г | Г | Г | Г | И |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 |  |  |  |  | ГГГИИИГГГГГГИ | | | | | | | | | | | | |  |  |
| P3 | ИИИГГГГГГГИИИ | | | | | | | | | | | | |  |  |  |  |  |  |

Пусть квант времени равняется 3.

«И» - исполняется, «Г» - в ожидании готовности, « » - завершен или не начат.

Полное время выполнения всех процессов: 7 + 1 + 4 + 6 = 18.

Среднее время выполнения процесса: 18 / 4 = 4.5.

Среднее время ожидания процесса: (10 + 4 + 9 + 7) / 4 = 7.5.

Shortest-Job-First (SJF) (не вытесняющий):



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Процессы | |  | Время ожидания | | | |  | Полное время выполнения Время появления в очереди | | | | | | | | | | | |
| P0 |  |  |  | 10 |  |  |  |  |  | 7 |  |  |  |  |  | 1 |  |  |  |
| P1 |  |  |  | 4 |  |  |  |  |  | 1 |  |  |  |  |  | 2 |  |  |  |
| P2 |  |  |  | 3 |  |  |  |  |  | 4 |  |  |  |  |  | 4 |  |  |  |
| P3 |  |  |  | 0 |  |  |  |  |  | 6 |  |  |  |  |  | 0 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Время | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |  |
| P0 |  | ГГГГГГГГГГИИИИИИИ | | | | | | | | | | | | | | | | |  |
| P1 |  |  | Г | Г | Г | Г | И |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 |  |  |  |  | Г | Г | Г | И | И | И | И |  |  |  |  |  |  |  |  |
| P3 | И | И | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |

«И» - исполняется, «Г» - в ожидании готовности, « » - завершен или не начат.

Полное время выполнения всех процессов: 7 + 1 + 4 + 6 = 18.

Среднее время выполнения процесса: 18 / 4 = 4.5.

Среднее время ожидания процесса: (10 + 4 + 3 + 0) / 4 = 4.25.

Shortest-Job-First (SJF) (не вытесняющий приоритетный):



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Процес |  | Время ожидания | | |  |  | Полное время | | |  | Время появления в | | |  | Приоритет | | |  |
| сы |  |  |  |  |  |  | выполнения | | |  |  | очереди |  |  |  |  |  |  |
| P0 |  |  | 6 |  |  |  |  | 7 |  |  |  | 1 |  |  |  | 2 |  |  |
| P1 |  |  | 4 |  |  |  |  | 1 |  |  |  | 2 |  |  |  | 1 |  |  |
| P2 |  |  | 10 |  |  |  |  | 4 |  |  |  | 4 |  |  |  | 3 |  |  |
| P3 |  |  | 0 |  |  |  |  | 6 |  |  |  | 0 |  |  |  | 4 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Время | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| P0 |  | ГГГГГГИИИИИИИ | | | | | | | | | | | | |  |  |  |  |
| P1 |  |  | Г | Г | Г | Г | И |  |  |  |  |  |  |  |  |  |  |  |
| P2 |  |  |  |  | ГГГГГГГГГГИИИИ | | | | | | | | | | | | | |
| P3 | И | И | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |

Будем считать, что большее значение приоритета соответстует меньшему приоритету, т.е. наиболее приоритетным в данном примере является P1, а наименее приоритетным — процесс P3.

«И» - исполняется, «Г» - в ожидании готовности, « » - завершен или не начат.

Полное время выполнения всех процессов: 7 + 1 + 4 + 6 = 18.

Среднее время выполнения процесса: 18 / 4 = 4.5.

Среднее время ожидания процесса: (10 + 6 + 4 + 0) / 4 = 5.

1. **Вытесняющие алгоритмы планирования процессов.**

Shortest-Job-First (SJF) (вытесняющий):



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Процессы | |  | Время ожидания | | | |  | Полное время выполнения | | | | | | Время появления в | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | очереди | | |  |  |
| P0 |  |  |  |  | 10 |  |  |  |  |  | 7 |  |  |  |  | 1 |  |  |  |
| P1 |  |  |  |  | 0 |  |  |  |  |  | 1 |  |  |  |  | 2 |  |  |  |
| P2 |  |  |  |  | 3 |  |  |  |  |  | 4 |  |  |  |  | 4 |  |  |  |
| P3 |  |  |  |  | 1 |  |  |  |  |  | 6 |  |  |  |  | 0 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Время | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |  |
| P0 |  | ГГГГГГГГГГИИИИИИИ | | | | | | | | | | | | | | | | |  |
| P1 |  |  | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 |  |  |  |  | Г | Г | Г | И | И | И | И |  |  |  |  |  |  |  |  |
| P3 | И | И | Г | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |

«И» - исполняется, «Г» - в ожидании готовности, « » - завершен или не начат.

Полное время выполнения всех процессов: 7 + 1 + 4 + 6 = 18.

Среднее время выполнения процесса: 18 / 4 = 4.5.

Среднее время ожидания процесса: (10 + 0 + 3 + 1) / 4 = 3.5.

Shortest-Job-First (SJF) (приоритетный):



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Процес | Время ожидания | Полное время | Время появления в | Приоритет |
| сы |  | выполнения | очереди |  |
| P0 | 1 | 7 | 1 | 2 |
| P1 | 0 | 1 | 2 | 1 |
| P2 | 5 | 4 | 4 | 3 |
| P3 | 12 | 6 | 0 | 4 |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Время | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| P0 |  | И | Г | И | И | И | И | И | И |  |  |  |  |  |  |  |  |  |
| P1 |  |  | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 |  |  |  |  | Г | Г | Г | Г | Г | И | И | И | И |  |  |  |  |  |
| P3 | ИГГГГГГГГГГГГИИИИИ | | | | | | | | | | | | | | | | | |

Будем считать, что большее значение приоритета соответстует меньшему приоритету, т.е. наиболее приоритетным в данном примере является P1, а наименее приоритетным — процесс P3.

«И» - исполняется, «Г» - в ожидании готовности, « » - завершен или не начат.

Полное время выполнения всех процессов: 7 + 1 + 4 + 6 = 18.

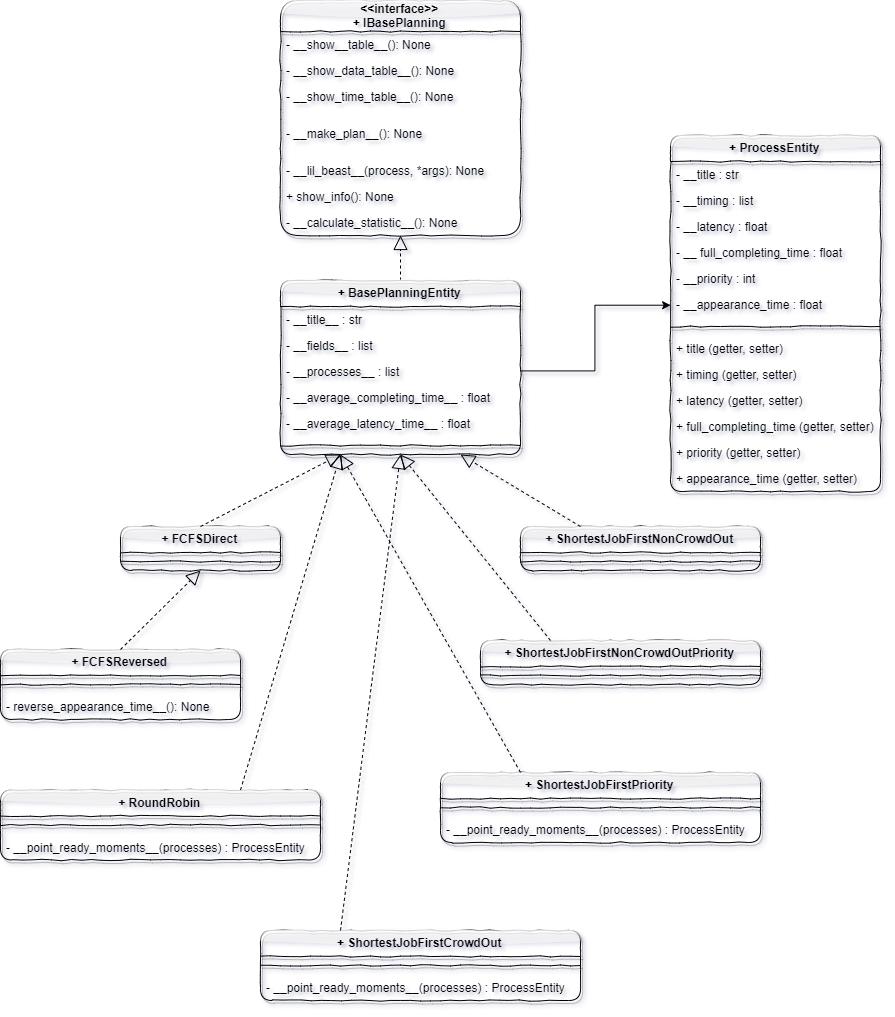
Среднее время выполнения процесса: 18 / 4 = 4.5.

Среднее время ожидания процесса: (12 + 0 + 5 + 1) / 4 = 4.5.

В данном случае самыми эффективным оказались First-Come, First-Served (FCFS) (обратный) и Shortest-Job-First (SJF) (вытесняющий).

1. **Программная реализация алгоритмов задания 1 и 2.**

Диаграмма классов программы:



Код программы на ЯП Python:

**IbasePlanning:**

import abc

class IBasePlanning(abc.ABC):

@abc.abstractmethod

def \_\_show\_tables\_\_(self) -> None:

pass

@abc.abstractmethod

def \_\_show\_data\_table\_\_(self) -> None:

pass

@abc.abstractmethod

def \_\_show\_time\_table\_\_(self) -> None:

pass

@abc.abstractmethod

def \_\_make\_plan\_\_(self) -> None:

pass

@abc.abstractmethod

def \_\_lil\_beast\_\_(self, process, \*args):

"""

This creature is needed to eat full\_completing\_time and provide timing with latency.

Wonderful and cruel it is a part of the great mechanism...

Laurent...

"""

pass

@abc.abstractmethod

def show\_info(self):

pass

@abc.abstractmethod

def \_\_calculate\_statistic\_\_(self):

pass

**BasePlanningEntity**:

from PlanningSystems.Interfaces.IBasePlanning import IBasePlanning

from prettytable import PrettyTable

class BasePlanningEntity(IBasePlanning):

\_\_title\_\_ = None

\_\_fields\_\_ = ['Processes', 'Latency', 'Full completing time', 'Appearance time', 'Priority']

def \_\_init\_\_(self, processes: list):

self.\_\_processes\_\_ = sorted(processes, key=lambda x: x.title)

self.\_\_whole\_time\_\_ = sum(process.full\_completing\_time for process in self.\_\_processes\_\_)

self.\_\_average\_completing\_time\_\_ = None

self.\_\_average\_latency\_time\_\_ = None

def \_\_show\_tables\_\_(self) -> None:

self.\_\_make\_plan\_\_()

self.\_\_show\_data\_table\_\_()

self.\_\_show\_time\_table\_\_()

def \_\_show\_data\_table\_\_(self) -> None:

table = PrettyTable()

table.title = self.\_\_title\_\_

table.field\_names = self.\_\_fields\_\_

data\_rows = [

(process.title, process.latency, process.full\_completing\_time, process.appearance\_time, process.priority)

for process in self.\_\_processes\_\_

]

table.add\_rows(data\_rows)

print(table)

def \_\_show\_time\_table\_\_(self) -> None:

table = PrettyTable()

table.title = self.\_\_title\_\_

table.field\_names = ['Time'] + [str(i) for i in range(1, self.\_\_whole\_time\_\_ + 1)]

data\_rows = [[process.title] + process.timing for process in self.\_\_processes\_\_]

table.add\_rows(data\_rows)

print(table)

def \_\_make\_plan\_\_(self) -> None:

for process in self.\_\_processes\_\_:

process.timing = [" " for i in range(self.\_\_whole\_time\_\_)]

def show\_info(self):

self.\_\_show\_tables\_\_()

self.\_\_calculate\_statistic\_\_()

print(f'Full completing time: {self.\_\_whole\_time\_\_}')

print(f'Average completing time: {self.\_\_average\_completing\_time\_\_}')

print(f'Average latency time: {self.\_\_average\_latency\_time\_\_}')

print('-' \* 50, '\n\n')

def \_\_calculate\_statistic\_\_(self):

self.\_\_average\_completing\_time\_\_ = sum([process.full\_completing\_time for process in self.\_\_processes\_\_]) \

/ len(self.\_\_processes\_\_)

self.\_\_average\_latency\_time\_\_ = sum([process.latency for process in self.\_\_processes\_\_])\

/ len(self.\_\_processes\_\_)

def \_\_lil\_beast\_\_(self, process, \*args):

pass

**ProcessEntity**:

class ProcessEntity:

def \_\_init\_\_(self, title, latency, full\_completing\_time, priority, appearance\_time, timing=None):

self.\_\_title = title

self.\_\_timing = timing

self.\_\_latency = latency

self.\_\_full\_completing\_time = full\_completing\_time

self.\_\_priority = priority

self.\_\_appearance\_time = appearance\_time

@property

def title(self):

return self.\_\_title

@property

def latency(self):

return self.\_\_latency

@property

def full\_completing\_time(self):

return self.\_\_full\_completing\_time

@property

def priority(self):

return self.\_\_priority

@property

def appearance\_time(self):

return self.\_\_appearance\_time

@property

def timing(self):

return self.\_\_timing

@title.setter

def title(self, title):

self.\_\_title = title

@latency.setter

def latency(self, latency):

self.\_\_latency = latency

@full\_completing\_time.setter

def full\_completing\_time(self, full\_completing\_time):

self.\_\_full\_completing\_time = full\_completing\_time

@priority.setter

def priority(self, priority):

self.\_\_priority = priority

@appearance\_time.setter

def appearance\_time(self, appearance\_time):

self.\_\_appearance\_time = appearance\_time

@timing.setter

def timing(self, timing):

self.\_\_timing = timing

**FCFSDirect**:

from Entities.BasePlanningEntity import BasePlanningEntity

import copy

class FCFSDirect(BasePlanningEntity):

\_\_title\_\_ = "FIRST-COME, FIRST-SERVED, direct"

def \_\_init\_\_(self, processes: list):

super().\_\_init\_\_(processes)

def \_\_make\_plan\_\_(self) -> None:

super().\_\_make\_plan\_\_()

* sorted\_processes - objects which will lose their full completing time,
* but they will have latency and timing

sorted\_processes = sorted(copy.deepcopy(self.\_\_processes\_\_), key=lambda x: x.appearance\_time)

start = min(process.appearance\_time for process in sorted\_processes)

for i, process in enumerate(sorted\_processes, start=0):

sorted\_processes[i] = self.\_\_lil\_beast\_\_(process, start, start + process.full\_completing\_time)

start += process.full\_completing\_time

sorted\_processes = sorted(sorted\_processes, key=lambda x: x.title)

for i in range(len(sorted\_processes)):

self.\_\_processes\_\_[i].timing = sorted\_processes[i].timing

self.\_\_processes\_\_[i].latency = sorted\_processes[i].latency

def \_\_lil\_beast\_\_(self, process, \*args):

"""

See description in an abstract class file.

:param process: the process itself.

:param args: start and end of timing (in MAIN FLOW)

:return: sophisticated lil beast itself...

"""

start, end = args

process.timing[process.appearance\_time:end] = ["Г" for i in range(process.appearance\_time, end)]

process.timing[start:end] = ['И' for i in range(start, end)]

process.latency = process.timing.count('Г')

return process

**FCFSReversed**:

from PlanningSystems.FCFSDirect import FCFSDirect

class FCFSReversed(FCFSDirect):

\_\_title\_\_ = "FIRST-COME, FIRST-SERVED, reversed"

def \_\_init\_\_(self, processes: list):

super().\_\_init\_\_(processes)

self.\_\_reverse\_appearance\_time\_\_()

def \_\_reverse\_appearance\_time\_\_(self):

self.\_\_processes\_\_ = sorted(self.\_\_processes\_\_, key=lambda x: x.appearance\_time)

appearance\_time = [process.appearance\_time for process in self.\_\_processes\_\_]

appearance\_time.reverse()

for i in range(len(self.\_\_processes\_\_)):

self.\_\_processes\_\_[i].appearance\_time = appearance\_time[i]

self.\_\_processes\_\_ = sorted(self.\_\_processes\_\_, key=lambda x: x.title)

**RoundRobin**:

from Entities.BasePlanningEntity import BasePlanningEntity

import copy

class RoundRobin(BasePlanningEntity):

\_\_title\_\_ = "ROUND ROBIN (RR)"

portion = 3

def \_\_init\_\_(self, processes: list):

super().\_\_init\_\_(processes)

def \_\_make\_plan\_\_(self) -> None:

super().\_\_make\_plan\_\_()

sorted\_processes = sorted(copy.deepcopy(self.\_\_processes\_\_), key=lambda x: x.appearance\_time)

start = min(process.appearance\_time for process in sorted\_processes)

while start < self.\_\_whole\_time\_\_:

for i in range(0, len(sorted\_processes)):

sorted\_processes[i], start = self.\_\_lil\_beast\_\_(sorted\_processes[i], start)

sorted\_processes = self.\_\_point\_ready\_moments\_\_(sorted\_processes)

sorted\_processes = sorted(sorted\_processes, key=lambda x: x.title)

for i in range(len(sorted\_processes)):

self.\_\_processes\_\_[i].timing = sorted\_processes[i].timing

self.\_\_processes\_\_[i].latency = sorted\_processes[i].latency

def \_\_lil\_beast\_\_(self, process, \*args):

"""

See description in an abstract class file.

:param process: the process itself.

:param args: current time.

:return: process, time.

"""

time = args[0]

process.timing[time:time + min(self.portion, process.full\_completing\_time)] \

= ['И' for i in range(min(self.portion, process.full\_completing\_time))]

time += min(self.portion, process.full\_completing\_time)

process.full\_completing\_time -= min(self.portion, process.full\_completing\_time)

return process, time

@staticmethod

def \_\_point\_ready\_moments\_\_(processes):

"""

It should be used after \_\_lil\_beast\_\_.

:param processes: processes

:return: modified processes

"""

for i, process in enumerate(processes, start=0):

start = process.appearance\_time

end = ''.join(process.timing).rfind('И')

for j in range(start, end + 1):

if process.timing[j] != 'И':

process.timing[j] = 'Г'

process.latency += 1

return processes

**ShortestJobFirstCrowdOut**:

from Entities.BasePlanningEntity import BasePlanningEntity

import copy

class ShortestJobFirstCrowdOut(BasePlanningEntity):

\_\_title\_\_ = "Shortest Job First Crowd Out"

def \_\_init\_\_(self, processes: list):

super().\_\_init\_\_(processes)

def \_\_make\_plan\_\_(self) -> None:

super().\_\_make\_plan\_\_()

processes\_copy = copy.deepcopy(self.\_\_processes\_\_)

time = 0

while time < self.\_\_whole\_time\_\_:

active\_processes = [

process for process in processes\_copy

if

process.full\_completing\_time != 0 and process.appearance\_time <= time

]

active\_processes.sort(key=lambda x: x.full\_completing\_time)

if len(active\_processes) > 0:

active\_processes[0] = self.\_\_lil\_beast\_\_(active\_processes[0], time)

time += 1

self.\_\_point\_ready\_moments\_\_(processes\_copy)

for i, process in enumerate(processes\_copy, start=0):

self.\_\_processes\_\_[i].latency = process.latency

self.\_\_processes\_\_[i].timing = process.timing

def \_\_lil\_beast\_\_(self, process, \*args):

"""

Lil beast itself.

:param process: process

:param args: time

:return: updated process

"""

time = args[0]

process.timing[time:time + 1] = ['И' for i in range(time, time + 1)]

process.full\_completing\_time -= 1

return process

@staticmethod

def \_\_point\_ready\_moments\_\_(processes):

"""

It should be used after \_\_lil\_beast\_\_.

:param processes: processes

:return: modified processes

"""

for i, process in enumerate(processes, start=0):

start = process.appearance\_time

end = ''.join(process.timing).rfind('И')

for j in range(start, end + 1):

if process.timing[j] != 'И':

process.timing[j] = 'Г'

process.latency += 1

return processes

**ShortestJobFirstNonCrowdOut**:

from Entities.BasePlanningEntity import BasePlanningEntity

import copy

class ShortestJobFirstNonCrowdOut(BasePlanningEntity):

\_\_title\_\_ = "Shortest Job First Non Crowd Out"

def \_\_init\_\_(self, processes: list):

super().\_\_init\_\_(processes)

def \_\_make\_plan\_\_(self) -> None:

super().\_\_make\_plan\_\_()

processes\_copy = copy.deepcopy(self.\_\_processes\_\_)

time = 0

while time < self.\_\_whole\_time\_\_:

active\_processes = [

process for process in processes\_copy

if

process.full\_completing\_time != 0 and process.appearance\_time <= time

]

active\_processes.sort(key=lambda x: x.full\_completing\_time)

if len(active\_processes) > 0:

active\_processes[0], time = self.\_\_lil\_beast\_\_(active\_processes[0], time)

else:

time += 1

for i, process in enumerate(processes\_copy, start=0):

self.\_\_processes\_\_[i].latency = process.latency

self.\_\_processes\_\_[i].timing = process.timing

def \_\_lil\_beast\_\_(self, process, \*args):

"""

Lil beast itself.

:param process: process

:param args: time

:return: updated process, time

"""

time = args[0]

process.timing[process.appearance\_time:time + process.full\_completing\_time] = \

['Г' for i in range(process.appearance\_time, time + process.full\_completing\_time)]

process.timing[time:time + process.full\_completing\_time] = \

['И' for i in range(time, time + process.full\_completing\_time)]

process.latency = process.timing.count('Г')

time += process.full\_completing\_time

process.full\_completing\_time = 0

return process, time

**ShortestJobFirstNonCrowdOutPriority**:

from Entities.BasePlanningEntity import BasePlanningEntity

import copy

class ShortestJobFirstNonCrowdOutPriority(BasePlanningEntity):

\_\_title\_\_ = "Shortest Job First Non Crowd Out Priority"

def \_\_init\_\_(self, processes: list):

super().\_\_init\_\_(processes)

def \_\_make\_plan\_\_(self) -> None:

super().\_\_make\_plan\_\_()

processes\_copy = copy.deepcopy(self.\_\_processes\_\_)

time = 0

while time < self.\_\_whole\_time\_\_:

active\_processes = [

process for process in processes\_copy

if

process.full\_completing\_time != 0 and process.appearance\_time <= time

]

active\_processes.sort(key=lambda x: x.priority)

if len(active\_processes) > 0:

active\_processes[0], time = self.\_\_lil\_beast\_\_(active\_processes[0], time)

else:

time += 1

for i, process in enumerate(processes\_copy, start=0):

self.\_\_processes\_\_[i].latency = process.latency

self.\_\_processes\_\_[i].timing = process.timing

def \_\_lil\_beast\_\_(self, process, \*args):

"""

Lil beast itself.

:param process: process

:param args: time

:return: updated process, time

"""

time = args[0]

process.timing[process.appearance\_time:time + process.full\_completing\_time] = \

['Г' for i in range(process.appearance\_time, time + process.full\_completing\_time)]

process.timing[time:time + process.full\_completing\_time] = \

['И' for i in range(time, time + process.full\_completing\_time)]

process.latency = process.timing.count('Г')

time += process.full\_completing\_time

process.full\_completing\_time = 0

return process, time

**ShortestJobFirstPriority**:

import copy

from Entities.BasePlanningEntity import BasePlanningEntity

class ShortestJobFirstPriority(BasePlanningEntity):

\_\_title\_\_ = "Shortest Job First Priority"

def \_\_init\_\_(self, processes: list):

super().\_\_init\_\_(processes)

def \_\_make\_plan\_\_(self) -> None:

super().\_\_make\_plan\_\_()

processes\_copy = copy.deepcopy(self.\_\_processes\_\_)

time = 0

while time < self.\_\_whole\_time\_\_:

active\_processes = [

process for process in processes\_copy

if

process.full\_completing\_time != 0 and process.appearance\_time <= time

]

active\_processes.sort(key=lambda x: x.priority)

if len(active\_processes) > 0:

active\_processes[0] = self.\_\_lil\_beast\_\_(active\_processes[0], time)

time += 1

self.\_\_point\_ready\_moments\_\_(processes\_copy)

for i, process in enumerate(processes\_copy, start=0):

self.\_\_processes\_\_[i].latency = process.latency

self.\_\_processes\_\_[i].timing = process.timing

def \_\_lil\_beast\_\_(self, process, \*args):

"""

Lil beast itself.

:param process: process

:param args: time

:return: updated process

"""

time = args[0]

process.timing[time:time + 1] = ['И' for i in range(time, time + 1)]

process.full\_completing\_time -= 1

return process

@staticmethod

def \_\_point\_ready\_moments\_\_(processes):

"""

It should be used after \_\_lil\_beast\_\_.

:param processes: processes

:return: modified processes

"""

for i, process in enumerate(processes, start=0):

start = process.appearance\_time

end = ''.join(process.timing).rfind('И')

for j in range(start, end + 1):

if process.timing[j] != 'И':

process.timing[j] = 'Г'

process.latency += 1

return processes

**main**:

from PlanningSystems.FCFSDirect import FCFSDirect

from PlanningSystems.FCFSReversed import FCFSReversed

from PlanningSystems.RoundRobin import RoundRobin

from PlanningSystems.ShortestJobFirstCrowdOut import ShortestJobFirstCrowdOut

from PlanningSystems.ShortestJobFirstNonCrowdOutPriority import ShortestJobFirstNonCrowdOutPriority

from PlanningSystems.ShortestJobFirstNonCrowdOut import ShortestJobFirstNonCrowdOut

from PlanningSystems.ShortestJobFirstPriority import ShortestJobFirstPriority

from Entities.ProcessEntity import ProcessEntity

import copy

def main():

processes = [

ProcessEntity("P0", 0, 7, 2, 1),

ProcessEntity("P1", 0, 1, 1, 2),

ProcessEntity("P2", 0, 4, 3, 4),

ProcessEntity("P3", 0, 6, 4, 0),

]

FCFSDirect(copy.deepcopy(processes)).show\_info()

FCFSReversed(copy.deepcopy(processes)).show\_info()

RoundRobin(copy.deepcopy(processes)).show\_info()

ShortestJobFirstNonCrowdOut(copy.deepcopy(processes)).show\_info()

ShortestJobFirstNonCrowdOutPriority(copy.deepcopy(processes)).show\_info()

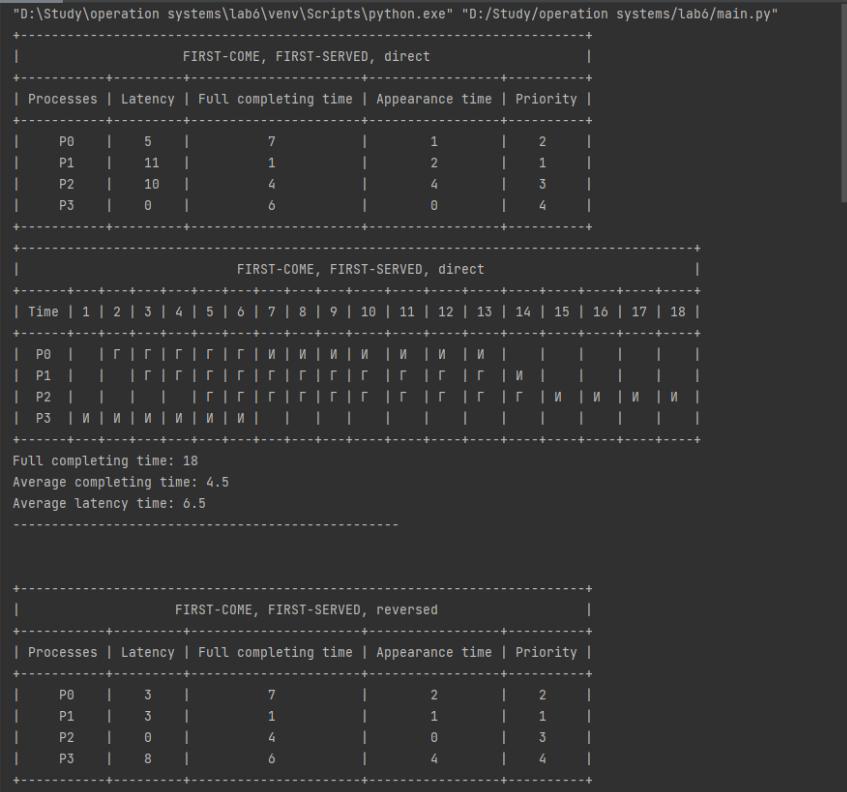
ShortestJobFirstCrowdOut(copy.deepcopy(processes)).show\_info()

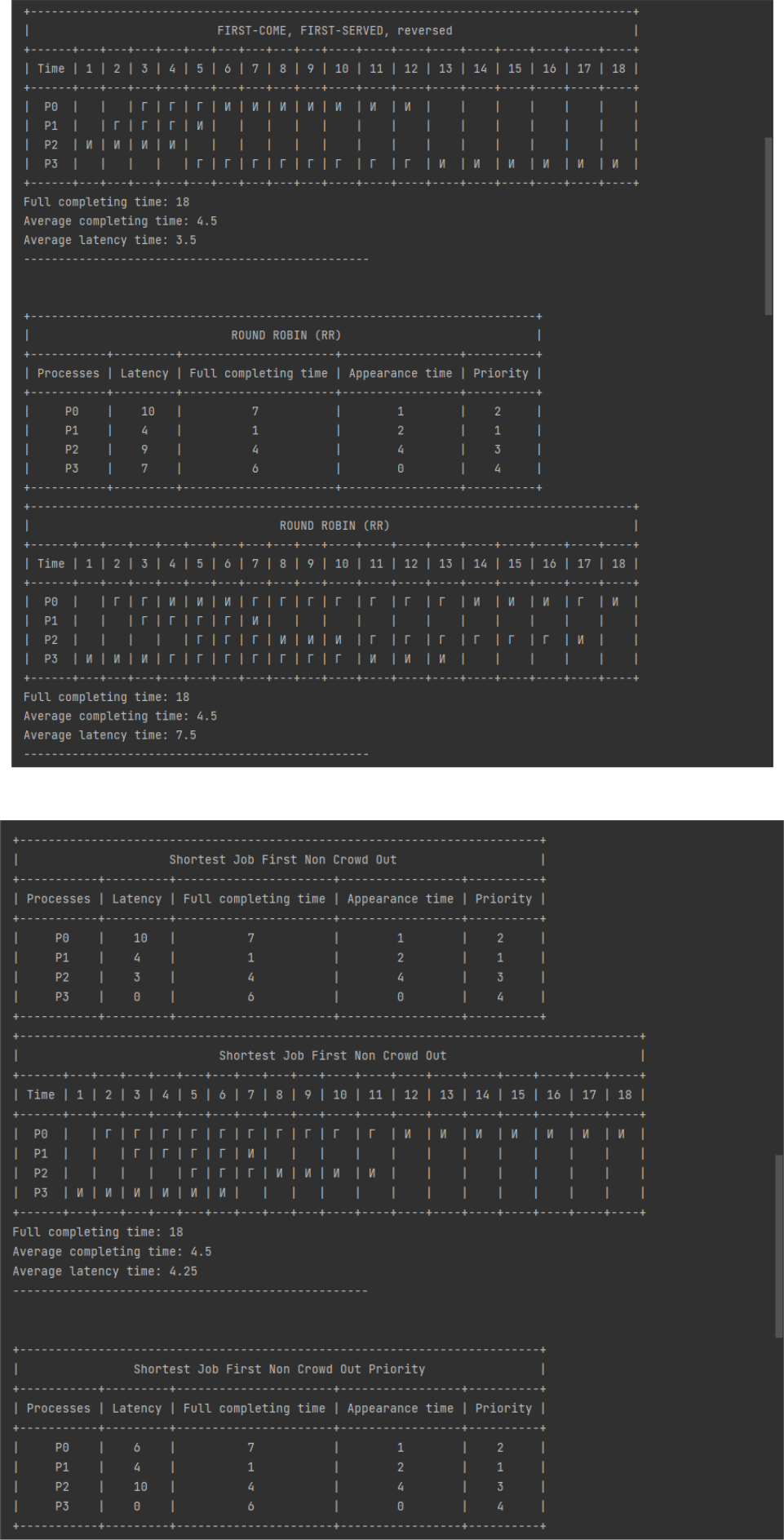
ShortestJobFirstPriority(copy.deepcopy(processes)).show\_info()

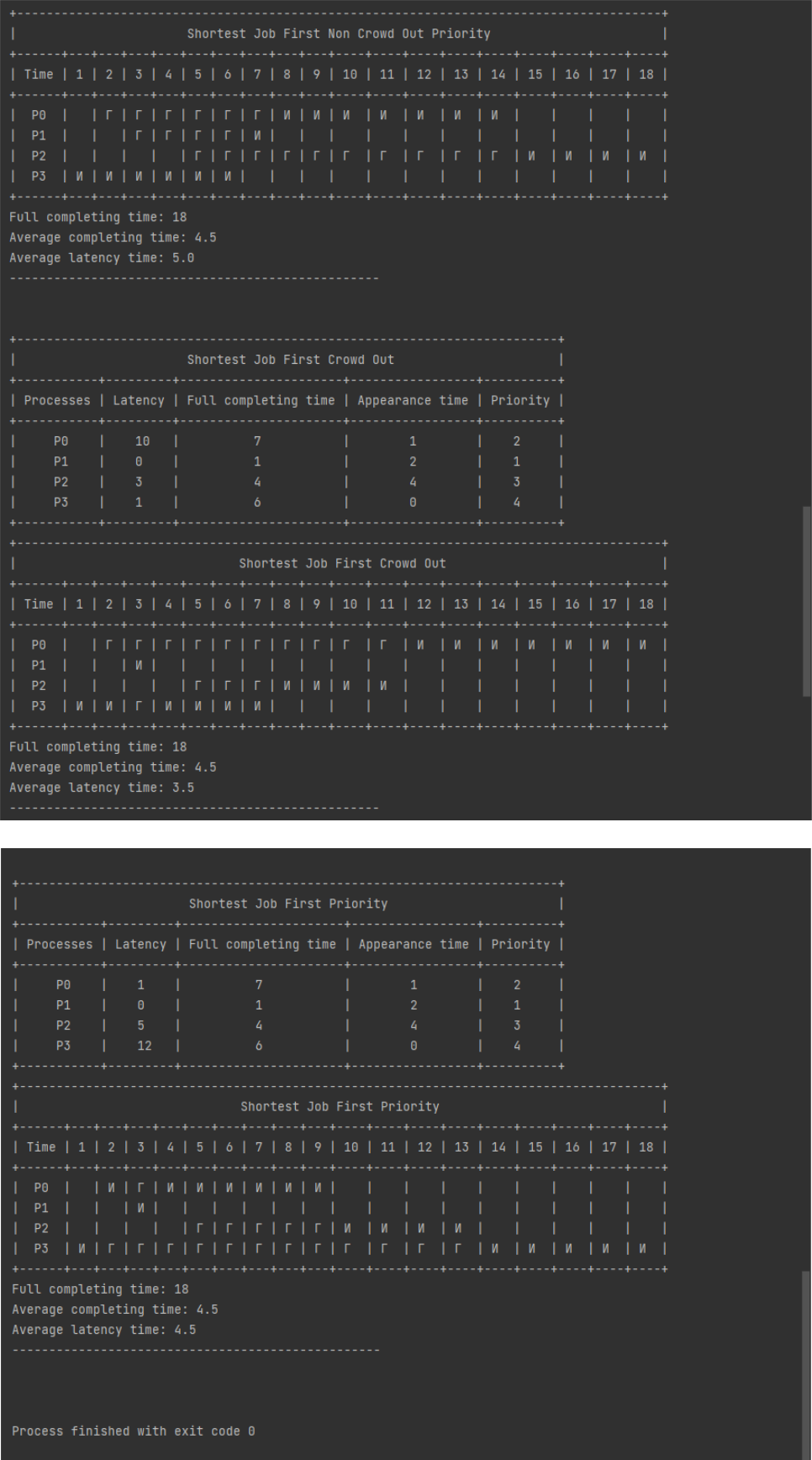
if \_\_name\_\_ == "\_\_main\_\_":

main()

**Результат работы программы**:







**Вывод**: в результате выполнения лабораторной работы был изучены типовые

алгоритмы планирования.