ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

Кафедра ВС

КУРСОВАЯ РАБОТА

По предмету: «Моделирование»

На тему: «Модель функционирования многоядерной системы»

Выполнил:

Студент группы МГ-172

Суходоева Н.Н.

Проверил:

д.т.н., профессор

Родионов А.С.

Новосибирск 2018

Постановка задачи

Построить модель работы многоядерной системы, содержащей 4096 блоков памяти и 1024 ядер. В систему на выполнение поступают задачи, определяемые временем счета, количеством требуемых блоков памяти и ядер. Ядра различны и имеет порядковый номер. При выделении ядер и памяти предпочтение отдается нумерации ядер и блоков подряд. Исследовать среднее время ожидания в очереди.

Граф событий представлен на рисунке 1.

1

2

3

4

5

τ

tполомки

tвосст

tоч

Рисунок 1 – граф событий

Ход решения

Исходные данные для моделирование задаются:

|  |
| --- |
|  |

Календарь событий создается:

|  |
| --- |
|  |

Событие поломки определенно в блоке:

|  |
| --- |
|  |

Событие восстановление определенно в блоке:

|  |
| --- |
|  |

Выделение требуемых ядер для задачи происходит в блоке:

|  |
| --- |
|  |

Выделение требуемых блоков памяти для задачи происходит в блоке:

|  |
| --- |
|  |

Подсчет времени ожидания задачи в очереди, считается от момента поступления задачи в очередь до ее выхода из нее:

|  |
| --- |
|  |

Вывод

Среднее время ожидания в очереди

Листинг программы

|  |
| --- |
| import random  import time  import pprint  queue = []  # начальные данные  NUM\_MEMORY\_BLOCK = 4096  NUM\_CORES = 1024  CORE\_RESERVE = 20  FAILURE\_PERSENT = 0.3  memory\_blocks = []  cores = []  cores\_reserve = []  calendar = []  # создание задачи  def get\_task():  ter = random.randint(1, 10) # время выполнения  v = random.randint(1, 100) # память  n = random.randint(4, 256) # ядра  return ter, v, n  # работа с памятью  # занятие памяти  def get\_memory\_blocks(num):  flag = False  position = 0  for n, \_ in enumerate(memory\_blocks):  sum = 0  for j in range(num):  sum += memory\_blocks[n + j]  if sum == 0:  for j in range(num):  memory\_blocks[n + j] = 1  flag = True  position = n  break  return flag, position, num  # освобождение памяти  def clean\_memory\_blocks(position, num):  for i in range(num):  memory\_blocks[position + i] = 0  # работа с ядрами  # занятие памяти  def get\_cores(num):  flag = False  position = 0  for n, \_ in enumerate(cores):  sum = 0  for j in range(num):  if n + j < len(cores):  sum += cores[n + j]  else:  return flag, position, num  if sum == 0:  for j in range(num):  cores[n + j] = 1  flag = True  position = n  break  return flag, position, num  # освобождение памяти  def clean\_cores(position, num):  for i in range(num):  cores[position + i] = 0  # выход из строя  def failure\_cores():  fail = random.random()  if fail <= FAILURE\_PERSENT:  fail\_count = -1  if NUM\_CORES \* fail >= 1:  fail\_count = random.randint(1, int(NUM\_CORES \* fail))  fail\_position = random.randint(0, NUM\_CORES)  return True, fail\_count, fail\_position  else:  return False, -1, -1  def failure\_memory():  fail = random.random()  if fail <= FAILURE\_PERSENT:  fail\_count = -1  if NUM\_MEMORY\_BLOCK \* fail >= 1:  fail\_count = random.randint(1, int(NUM\_MEMORY\_BLOCK \* fail))  fail\_position = random.randint(0, NUM\_MEMORY\_BLOCK)  return True, fail\_count, fail\_position  return False, -1, -1  # убийство ресурсов  def drop\_cores(position, num):  for i in range(num):  if position + i < len(cores):  cores[position + i] = 999  else:  print("Закончились ядра")  exit(-1)  def drop\_memory(position, num):  for i in range(num):  if position + i < len(memory\_blocks):  memory\_blocks[position + i] = 999  else:  print("Закончились блоки памяти")  exit(-1)  if \_\_name\_\_ == "\_\_main\_\_":  # начальные 2 задачи  queue.append(get\_task())  queue.append(get\_task())  # заполнение начальных массивов  for \_ in range(NUM\_CORES):  cores.append(0)  for \_ in range(NUM\_MEMORY\_BLOCK):  memory\_blocks.append(0)  for \_ in range(CORE\_RESERVE):  cores\_reserve.append(0)  pos = 0  time\_machine = 0  while True:  if not queue:  break  # получение задачи  ter, v, n = queue.pop(0)  # отработка задачи  flag\_cores, position\_cores, \_ = get\_cores(n)  if not flag\_cores:  print("Нет доступных ядер")  break  flag\_memory, position\_memory, \_ = get\_memory\_blocks(v)  if not flag\_memory:  print("Нет доступной памяти")  break  calendar.append((time\_machine, position\_cores, n, position\_memory, v))  time\_machine += ter  clean\_cores(position\_cores, n)  clean\_memory\_blocks(position\_memory, v)  # выход из строя  fail\_core\_flag, fail\_core\_count, fail\_core\_position = failure\_cores()  fail\_memory\_flag, fail\_memory\_count, fail\_memory\_position = failure\_memory()  if fail\_core\_flag:  drop\_cores(fail\_core\_position, fail\_core\_count)  if fail\_memory\_flag:  drop\_memory(fail\_memory\_position, fail\_memory\_count)  queue.append(get\_task())  pos += 1  pprint.pprint(calendar)  time.sleep(1)  print(calendar) |