Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт кибербезопасности и защиты информации

ЛАБОРАТОРНАЯ РАБОТА №2

«Планирование процессов.»
по дисциплине «Операционные системы»

Выполнил
студент гр. 4851003/10002 <подпись> Галкин К. К.

Руководитель
к. т. н. <подпись> Крундышев В.М.

Санкт-Петербург
2022

1. ЦЕЛЬ РАБОТЫ

Цель работы – изучение механизмов планирования процессов, разработка алгоритма приоритетного планирования и внедрение разработанного алгоритма в учебную операционную систему Pintos.

2. ХОД РАБОТЫ

Изначально, алгоритм планирования процессов работает по алгоритму Round Robin с квантом в 4 тика, без учета приоритета процесса и CPU burst. Касательно последнего, такого понятия в Pintos вообще не существует.

Давайте рассмотрим, как работает этап планирования процессов для следующего набора данных:

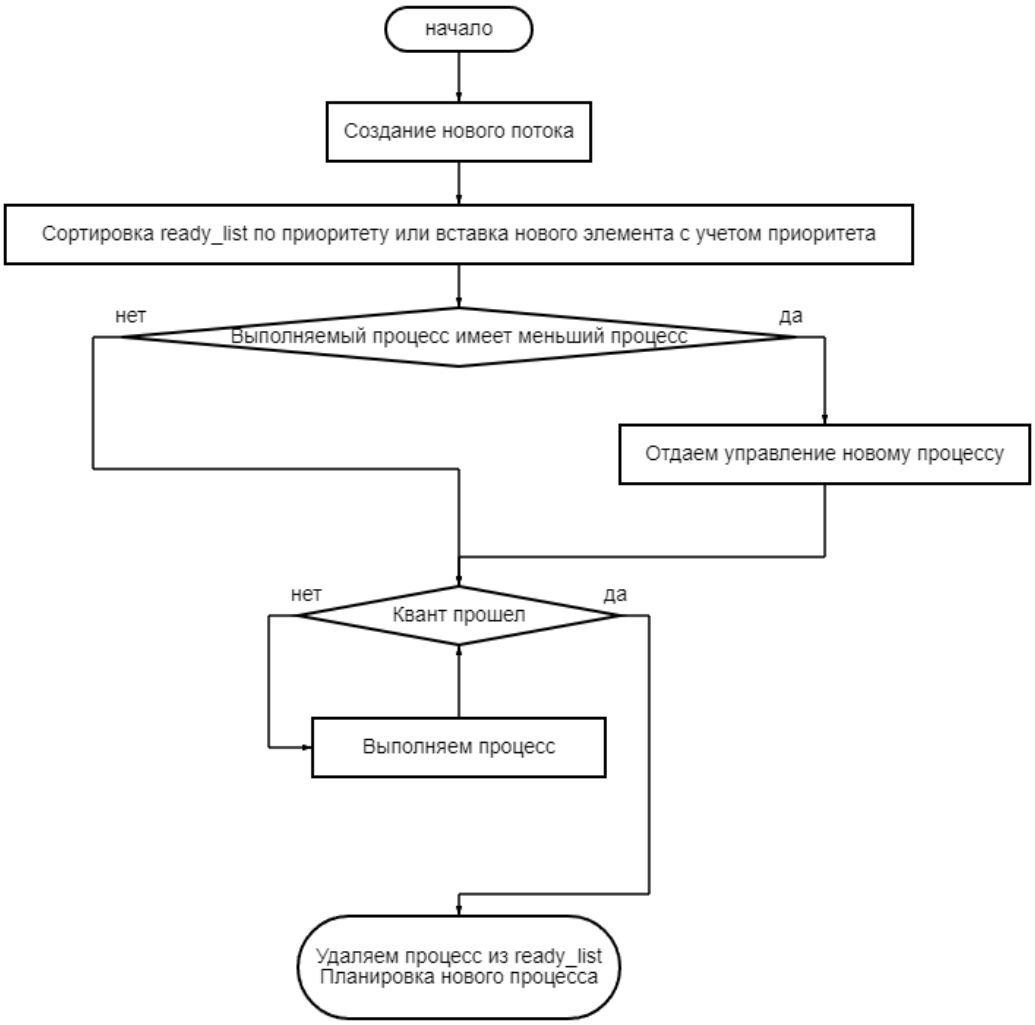
Процесс	Приоритет	CPU burst
Proc0	45	2
Proc1	44	2
Proc2	46	2
Proc3	33	3
Proc4	33	10

Построим диаграмму исполнения процессов для тех процессов, введенных ранее.

PR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	и	и	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	г	г	и	и	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	г	г	г	г	и	и	-	-	-	-	-	-	-	-	-	-	-	-	-
3	г	г	г	г	г	г	и	и	и	-	-	-	-	-	-	-	-	-	-
4	г	г	г	г	г	г	г	г	г	и	и	и	и	и	и	и	и	и	и

2.1. **Новый алгоритм планирования**

Суть нового алгоритма: используем сортировку по приоритетам. Чем больше приоритет, тем раньше этот поток встаёт во всех используемых списках. Так же не стоит забывать, что алгоритм должен не переключать процесс, если в списке процессов ожидающих выполнение нет процессов с таким же уровнем приоритета. А если такой поток есть, то должен использоваться алгоритм Round Robin, который уже реализован в ОС Pintos. Таким образом, блок-схема нового алгоритма будет выглядеть следующим образом:



Для реализации такого алгоритма в исходный код ОС были внесены следующие изменения:

- Threads.c

- Добавлена функция thread_priority_compare, которая берет два потока и сравнивает их приоритет.
- Список ready_list теперь сортируется функцией list_sort
- thread_yield теперь вызывается при создании потока

- Synch.c
  - Список потоков в структуре semaphore сортируется по приоритетам при работе функций sema_down и sema_up
  - В sema_up добавлен thread_yield, чтобы вытеснять процесс
  - Создана функция sema_priority_compare, которая нужна для сортировки списка в структуре semaphore_elem, предназначенной для работы мониторов. Эта функция берет список потоков из структуры semaphore и сравнивает приоритеты первых потоков из списка.
  - В cond_signal и cond_wait список cond.waiters сортируется

Таким образом, диаграмма выполнения процессов будет выглядеть так:

PR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	Г	Г	И	И	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	Г	Г	Г	Г	И	И	-	-	-	-	-	-	-	-	-	-	-	-	-
2	И	И	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	Г	Г	Г	Г	Г	Г	И	И	И	-	-	-	-	-	-	-	-	-	-
4	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И	И

2.2. **Новый алгоритм планирования**

Функция	Описание
tid_t thread_create (const char *name, int priority, thread_func *function, void *aux)	Создает поток с именем name, приоритетом priority, исполняемой функцией function.

void thread_block (void)	Изменение статуса потока на THREAD_BLOCKED и вызов планировщика.
void thread_unblock (struct thread *t)	Изменение статуса потока t на THREAD_READY, добавление потока в ready_list.
struct thread * thread_current (void)	Возвращает текущий поток
void thread_exit (void)	Изменение статуса текущего потока на THREAD_DYING, вызов планировщика, уничтожение потока
void thread_yield (void)	Прерывание текущего потока, возврат потока в список готовых к выполнению потока, вызов потока
struct thread * running_thread (void)	Возвращает запущенный поток
static bool is_thread (struct thread *t)	Проверяет, не равен поток NULL и не переполнен ли стек потока
static void init_thread (struct thread *t, const char *name, int priority)	Инициализация потока с именем name и приоритетом priority
static struct thread * next_thread_to_run (void)	Выбирает следующий поток для запуска из ready_list
void thread_schedule_tail (struct thread *prev)	Переключение между потоками. Обнуление пройденного значения тиков для кванта RR, убивается поток prev, освобождается его память
static void schedule (void)	Планировщик следующего потока

2.3.Механизм дарения приоритета

Для реализации алгоритма было решено модифицировать структуру потока, добавив несколько полей: замок, который разблокировку которого

приоритет ожидает, массив из “подаренных” приоритетов, на первом месте которого стоит изначальный приоритет. В структуре замка добавим поле is_donated, которая отвечает за состояние замка и приоритета держателя. Показывает, подарен ли приоритет для держателя замка или нет.

Соответственно, весь процесс “дарения” приоритета можно описать в двух функциях: lock_release и lock_acquire.

Рассмотрим подробнее каждую из них:

- Lock_acquire: происходит проверка того, что у замка есть владелец, если есть, тогда сохраняем текущий замок в переменную waiting_lock(donated_lock в старых версиях моего исходного кода). Далее проверяем, нужно ли вообще держателю замка давать приоритет, то есть приоритет держателя меньше, чем приоритет текущего потока. Если да, тогда заменяем приоритет держателя замка, добавляем подаренный приоритет в массив и увеличиваем его размер. После чего меняем проверяемый поток на поток-держатель замка. Так мы делаем, пока не найдем поток, который не находится в очереди на доступ в критическую секцию, то есть у кого waiting_lock не будет равен нулю. Таким образом, мы дойдем до такого замка(потока), который остановил всё планирование процессов, подарим ему приоритет и тем самым дадим ему процессорное время.
- Lock_release: возвращаем приоритет согласно следующим правилам: если больше нет потоков, которым необходим доступ к замку, тогда можно вернуть стартовые приоритет, то есть нулевой элемент массива donated_priorities, иначе – присваиваем максимально возможный приоритет из массива donated_priorities.
    - Дополнительное изменение в thread_set_priority: адаптация функции для работы с массивом подаренных приоритетов.

2.4.Результаты тестирования

```
Executing 'alarm-priority':
(alarm-priority) begin
(alarm-priority) Thread priority 30 woke up.
(alarm-priority) Thread priority 29 woke up.
(alarm-priority) Thread priority 28 woke up.
(alarm-priority) Thread priority 27 woke up.
(alarm-priority) Thread priority 26 woke up.
(alarm-priority) Thread priority 25 woke up.
(alarm-priority) Thread priority 24 woke up.
(alarm-priority) Thread priority 23 woke up.
(alarm-priority) Thread priority 22 woke up.
(alarm-priority) Thread priority 21 woke up.
(alarm-priority) end
Execution of 'alarm-priority' complete.
```

```
Executing 'priority-change':
(priority-change) begin
(priority-change) Creating a high-priority thread 2.
(priority-change) Thread 2 now lowering priority.
(priority-change) Thread 2 should have just lowered its priority.
(priority-change) Thread 2 exiting.
(priority-change) Thread 2 should have just exited.
(priority-change) end
```

```
Executing 'priority-fifo':
(priority-fifo) begin
(priority-fifo) 16 threads will iterate 16 times in the same order each time.
(priority-fifo) If the order varies then there is a bug.
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(priority-fifo) end
Execution of 'priority-fifo' complete.
```

```
Executing 'priority-preempt':
(priority-preempt) begin
(priority-preempt) Thread high-priority iteration 0
(priority-preempt) Thread high-priority iteration 1
(priority-preempt) Thread high-priority iteration 2
(priority-preempt) Thread high-priority iteration 3
(priority-preempt) Thread high-priority iteration 4
(priority-preempt) Thread high-priority done!
(priority-preempt) The high-priority thread should have already completed.
(priority-preempt) end
Execution of 'priority-preempt' complete.
Timer: 47 ticks
```

```
Executing 'priority-sema':
(priority-sema) begin
(priority-sema) Thread priority 30 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 29 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 28 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 27 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 26 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 25 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 24 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 23 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 22 woke up.
(priority-sema) Back in main thread.
(priority-sema) Thread priority 21 woke up.
(priority-sema) Back in main thread.
(priority-sema) end
Execution of 'priority-sema' complete.
```

```
Boot complete.
Executing 'priority-condvar':
(priority-condvar) begin
(priority-condvar) Thread priority 23 starting.
(priority-condvar) Thread priority 22 starting.
(priority-condvar) Thread priority 21 starting.
(priority-condvar) Thread priority 30 starting.
(priority-condvar) Thread priority 29 starting.
(priority-condvar) Thread priority 28 starting.
(priority-condvar) Thread priority 27 starting.
(priority-condvar) Thread priority 26 starting.
(priority-condvar) Thread priority 25 starting.
(priority-condvar) Thread priority 24 starting.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 30 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 29 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 28 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 27 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 26 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 25 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 24 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 23 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 22 woke up.
(priority-condvar) Signaling...
(priority-condvar) Thread priority 21 woke up.
(priority-condvar) end
Execution of 'priority-condvar' complete.
```

```
Executing 'priority-donate-one':
(priority-donate-one) begin
(priority-donate-one) This thread should have priority 32.  Actual priority: 32.
(priority-donate-one) This thread should have priority 33.  Actual priority: 33.
(priority-donate-one) acquire2: got the lock
(priority-donate-one) acquire2: done
(priority-donate-one) acquire1: got the lock
(priority-donate-one) acquire1: done
(priority-donate-one) acquire2, acquire1 must already have finished, in that order.
(priority-donate-one) This should be the last line before finishing this test.
(priority-donate-one) end
```

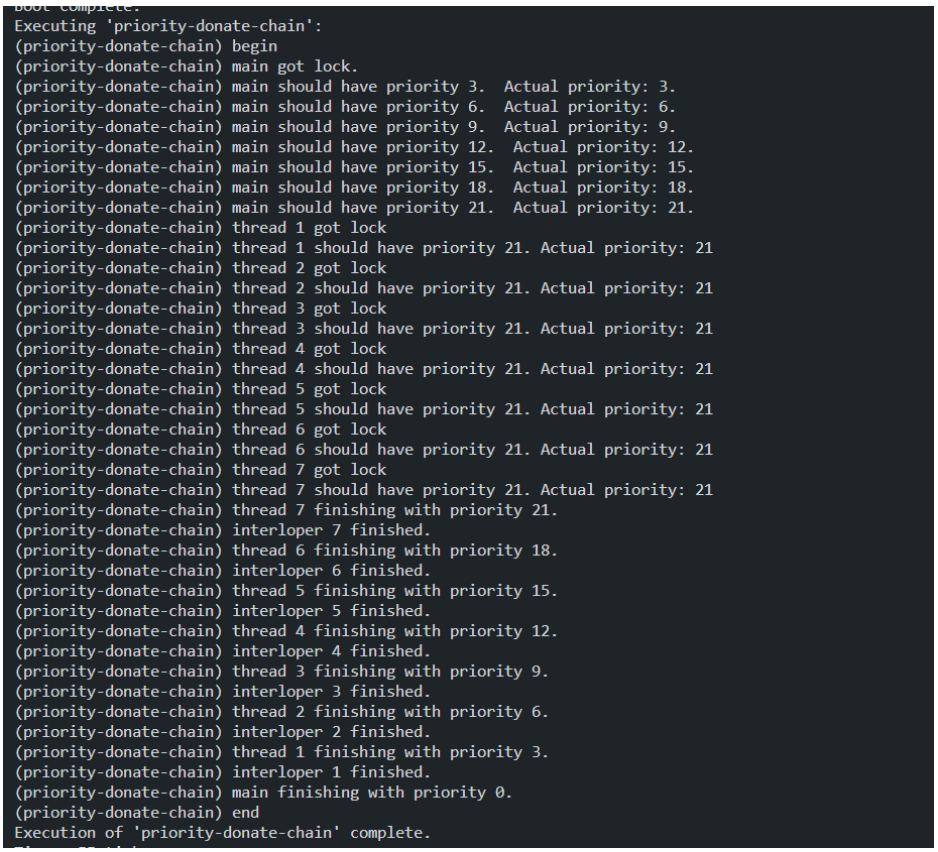
```
Executing 'priority-donate-lower':
(priority-donate-lower) begin
(priority-donate-lower) Main thread should have priority 41.  Actual priority: 41.
(priority-donate-lower) Lowering base priority...
(priority-donate-lower) Main thread should have priority 41.  Actual priority: 41.
(priority-donate-lower) acquire: got the lock
(priority-donate-lower) acquire: done
(priority-donate-lower) acquire must already have finished.
(priority-donate-lower) Main thread should have priority 21.  Actual priority: 21.
(priority-donate-lower) end
Execution of 'priority-donate-lower' complete.
```

```
Executing 'priority-donate-multiple':
(priority-donate-multiple) begin
(priority-donate-multiple) Main thread should have priority 32.  Actual priority: 32.
(priority-donate-multiple) Main thread should have priority 33.  Actual priority: 33.
(priority-donate-multiple) Thread b acquired lock b.
(priority-donate-multiple) Thread b finished.
(priority-donate-multiple) Thread b should have just finished.
(priority-donate-multiple) Main thread should have priority 32.  Actual priority: 32.
(priority-donate-multiple) Thread a acquired lock a.
(priority-donate-multiple) Thread a finished.
(priority-donate-multiple) Thread a should have just finished.
(priority-donate-multiple) Main thread should have priority 31.  Actual priority: 31.
(priority-donate-multiple) end
Execution of 'priority-donate-multiple' complete.
```

```
Executing 'priority-donate-multiple2':
(priority-donate-multiple2) begin
(priority-donate-multiple2) Main thread should have priority 34.  Actual priority: 34.
(priority-donate-multiple2) Main thread should have priority 36.  Actual priority: 36.
(priority-donate-multiple2) Main thread should have priority 36.  Actual priority: 36.
(priority-donate-multiple2) Thread b acquired lock b.
(priority-donate-multiple2) Thread b finished.
(priority-donate-multiple2) Thread a acquired lock a.
(priority-donate-multiple2) Thread a finished.
(priority-donate-multiple2) Thread c finished.
(priority-donate-multiple2) Threads b, a, c should have just finished, in that order.
(priority-donate-multiple2) Main thread should have priority 31.  Actual priority: 31.
(priority-donate-multiple2) end
```

```
Executing 'priority-donate-sema':
(priority-donate-sema) begin
(priority-donate-sema) Thread L acquired lock.
(priority-donate-sema) Thread L downed semaphore.
(priority-donate-sema) Thread H acquired lock.
(priority-donate-sema) Thread H finished.
(priority-donate-sema) Thread M finished.
(priority-donate-sema) Thread L finished.
(priority-donate-sema) Main thread finished.
(priority-donate-sema) end
Execution of 'priority-donate-sema' complete.
```

```
Executing 'priority-donate-nest':
(priority-donate-nest) begin
(priority-donate-nest) Low thread should have priority 32.  Actual priority: 32.
(priority-donate-nest) Low thread should have priority 33.  Actual priority: 33.
(priority-donate-nest) Medium thread should have priority 33.  Actual priority: 33.
(priority-donate-nest) Medium thread got the lock.
(priority-donate-nest) High thread got the lock.
(priority-donate-nest) High thread finished.
(priority-donate-nest) High thread should have just finished.
(priority-donate-nest) Middle thread finished.
(priority-donate-nest) Medium thread should just have finished.
(priority-donate-nest) Low thread should have priority 31.  Actual priority: 31.
(priority-donate-nest) end
Execution of 'priority-donate-nest' complete.
```



3. ВЫВОД

В ходе лабораторной работы был выявлен недостаток при планировании работы процессов в ОС Pintos. В итоге, был разработан новый алгоритм, учитывающий приоритет процессов, так же все примитивы синхронизации были адаптированы под новый алгоритм. Для корректной работы и исправления главного недостатка приоритетного планирования – голодания – был реализован алгоритм дарения приоритета процессу, освобождения замка которого ожидает поток с более высоким приоритетом.