

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»

—
Институт компьютерных наук и кибербезопасности
Высшая школа кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА №2

ПЛАНИРОВАНИЕ ПРОЦЕССОВ

по дисциплине «Операционные системы»

Выполнил
студенты гр. 5131001/30002

Мишенев Н. С.

<подпись>

Руководитель
программист

Огнёв Р. А.

<подпись>

Санкт-Петербург
2024г.

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ	3
ХОД РАБОТЫ	4
1. Диаграммы исполнения процессов.	4
1. Исходный алгоритм.	4
2. Модифицированный алгоритм.....	4
2. Сравнительный анализ диаграмм.....	4
3. Блок-схема нового разработанного алгоритма планирования.....	5
4. Описание функций, которые применяются для организации планирования: их назначение, описание аргументов, возвращаемого значения, связей друг с другом.	5
5. Исходные коды системы планирования Pintos с внесенными модификациями и комментариями.	6
> list.h + list.c	7
> Thread.h	8
> Thread.c	8
> Synch.h.....	10
> Synch.c	10
6. Описание тестовых задач: имя теста и описание действий, которые в нем совершаются; полученные при запуске результаты, анализ полученных результатов.....	14
ВЫВОД.....	17

ЦЕЛЬ РАБОТЫ

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

ХОД РАБОТЫ

1. Диаграммы исполнения процессов.

1. Исходный алгоритм.

Исходный алгоритм планирования процессов в ОС Pintos, не учитывал приоритет процессов. Диаграмма исполнения процессов была составлена для 4-го варианта задания.

4. Алгоритм: First Come First Served (FCFS).

Процесс:	CPU burst:	Приоритет процесса:
Proc0	4	2
Proc1	4	27
Proc2	4	60
Proc3	3	30
Proc4	7	31

Рис. 1. Вариант задания.

Диаграмма процессов (До модификации)

PID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Proc0	И	И	И	И																		
Proc1	Г	Г	Г	Г	И	И	И	И														
Proc2	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И										
Proc3	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И							
Proc4	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И

Рис. 2. Диаграмма исполнения процессов.

2. Модифицированный алгоритм.

После модификации алгоритма планировщик выбирает из списка готовых процессов, процесс с наивысшим приоритетом, поэтому вид диаграммы изменился:

Диаграмма процессов (После модификации)

PID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Proc0	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И
Proc1	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И				
Proc2	И	И	И	И																		
Proc3	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И								
Proc4	Г	Г	Г	Г	И	И	И	И	И	И	И											

Рис. 3. Новая диаграмма исполнения процессов.

2. Сравнительный анализ диаграмм.

Как можно заметить, из-за изменения алгоритма планирования изменился порядок исполнения процессов. Процессы с высшим приоритетом исполняются раньше, чем процессы с низшим приоритетом.

3. Блок-схема нового разработанного алгоритма планирования.

Для нового алгоритма планирования была создана соответствующая ему блок-схема:

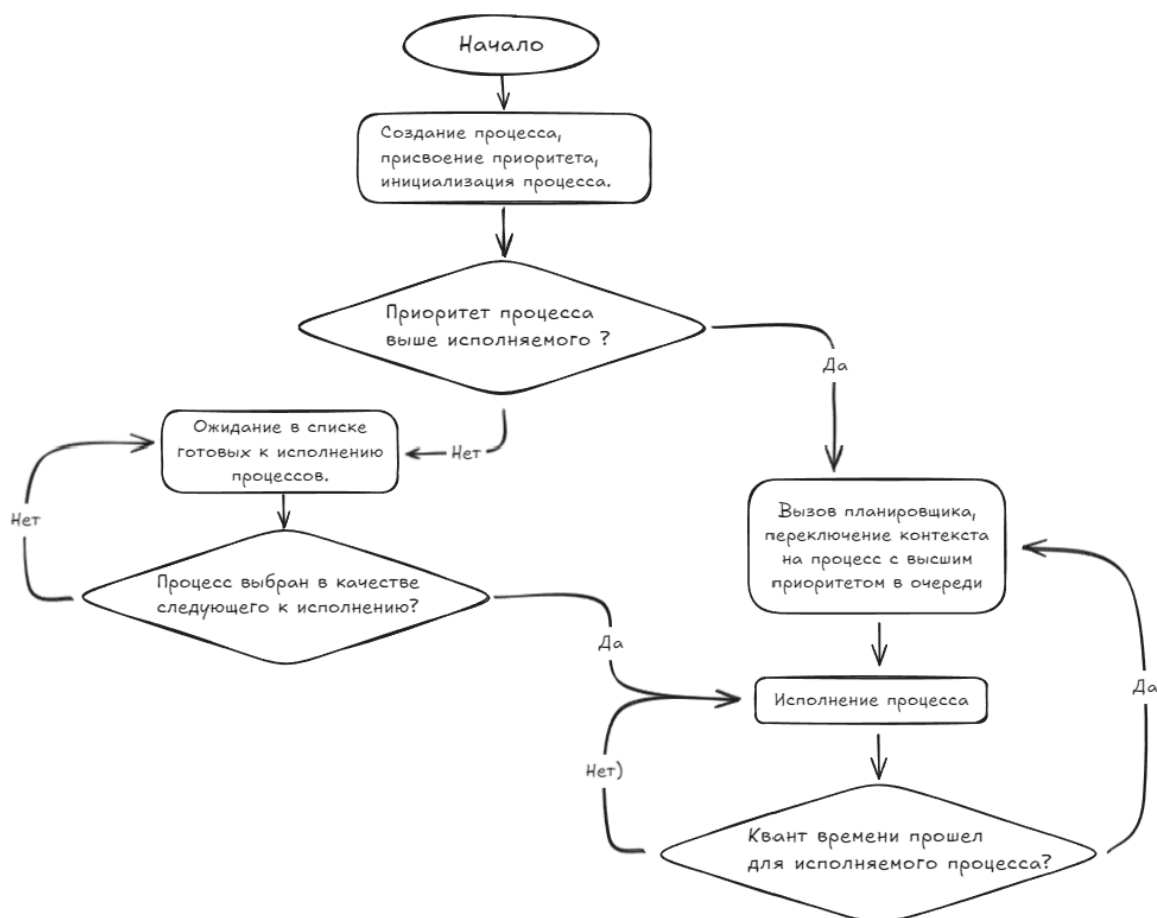


Рис. 4. Блок-схема для нового алгоритма планирования.

4. Описание функций, которые применяются для организации планирования: их назначение, описание аргументов, возвращаемого значения, связей друг с другом.

Были рассмотрены файлы *thread.c* и *synch.c*, и функции отвечающие за организацию планирования в них. Далее была составлена таблица, которая также включает в себя функции, разработанные в процессе выполнения лабораторной работы.

Функции, принимающие участие в планировании процессов

Прототип функции	Описание
<code>void thread_yield(void);</code>	Функция передает процесс планировщику, который помещает его в конец списка список процессов готовых к выполнению
<code>void schedule(void);</code>	Функция планировщика, которая выбирает следующий процесс для запуска с помощью функции <i>next_thread_to_run()</i> и осуществляет переключение контекста
<code>void thread_block(void);</code>	Функция изменяет статус текущего процесса на THREAD_BLOCKED и вызывает планировщик
<code>void thread_unblock(struct thread*);</code>	Функция изменяет статус текущего процесса на THREAD_READY и помещает его в список процессов готовых к выполнению
<code>bool compare_thread_priority(struct list_elem a, struct list_elem b, void *aux);</code>	Функция, которая сравнивает приоритеты элементов списка как процессов. Возвращает TRUE если приоритет <i>a</i> меньше приоритета <i>b</i> . Используется в функции <i>list_pop_max()</i> ;
<code>bool compare_semaphore_priority(struct list_elem a, struct list_elem b, void *aux);</code>	Функция, которая сравнивает приоритеты элементов списка как элементов семафора. Возвращает TRUE если приоритет <i>a</i> меньше приоритета <i>b</i> . Используется в функции <i>list_pop_max()</i> ;
<code>void sema_up(struct semaphore* sema);</code>	Производит разблокировку процесса из списка ожидания
<code>void sema_down(struct semaphore* sema);</code>	Усыпляет процесс, если КС занята и добавляет его в список ожидания
<code>void lock_acquire(struct lock*);</code>	Назначает текущий поток владельцем замка, если свободен, иначе блокирует его
<code>void lock_release(struct lock*);</code>	Замок освобождается процессом, и его владельцем назначается следующий процесс
<code>void cond_wait(struct condition* , struct lock*);</code>	Монитор блокирует процесс, пока выполняется некоторое условие и назначает процесс владельцем замка
<code>void cond_signal(struct condition* , struct lock*);</code>	Освобождает замок и пробуждает следующий процесс

Таблица 1. Функции, принимающие участие в планировании процессов.

5. Исходные коды системы планирования Pintos с внесенными модификациями и комментариями.

Для реализации очереди процессов в ОС Pintos используется описанный в самой ОС двусвязный список, поэтому в реализацию списка были внесены новые функции, позволяющие отыскивать и удалять максимальный элемент, а также находить определенный элемент.

Рассмотрим эти изменения.

> list.h + list.c

```
187  /// LAB 2 S
188  /* Created list functions. */
189  struct list_elem *list_pop_max (struct list *, list_less_func *);
190
191  struct list_elem *list_pop_exact (struct list *, list_exact_func *, void *aux);
192  struct list_elem *list_exact (struct list *, list_exact_func *, void *aux);
193  /// LAB 2 E
```

Рис. 5. Прототипы созданных функций в файле list.h

В заголовочном файле были описаны прототипы созданных функций, рассмотрим их назначение и реализацию:

list_pop_max(...) - находит и удаляет из списка максимальный элемент, определенный переданной функцией.

```
267  /// LAB 2 S
268  /* Removes the element with highest parameter decided by FUNC from LIST and returns it.
269     Undefined behavior if LIST is empty before removal.*/
270  struct list_elem *
271  list_pop_max(struct list *list, list_less_func *less)
272  {
273      struct list_elem *highest_p = list_max(list, less, NULL);
274      list_remove(highest_p);
275      return highest_p; /// addition
276  }
```

Рис. 6. Реализация функции list_pop_max() в файле list.c

list_exact(...) - возвращает первый элемент списка, который подходит под условие переданной функции.

```
278  /* Returns the element in LIST which fits to FUNC's condition with
279     given auxiliary data AUX. If there is more than one element,
280     returns the one that appears earlier in the list. If
281     the list is empty, returns NULL. */
282  struct list_elem *
283  list_exact (struct list *list, list_exact_func *func, void *aux)
284  {
285      struct list_elem *elem;
286      for (elem = list_begin (list) ; elem != list_end(list); elem = list_next(elem))
287      {
288          if (func (elem, aux))
289              return elem;
290      }
291
292      return NULL; /// addition
293  }
```

Рис. 7. Реализация функции list_exact() в файле list.c

list_pop_exact(...) - удаляет из списка первый элемент, который подходит под условие переданной функции.

```
295  /* Finds the element that fits the condition in given FUNC in LIST and pops
296     it. Also, returning popped element.*/
297  struct list_elem *
298  list_pop_exact(struct list *list, list_exact_func *func, void* aux)
299  {
300      struct list_elem *exact_element = list_exact(list, func, aux);
301      if (exact_element != NULL)
302      {
303          list_remove(exact_element);
304      }
305      return exact_element; ///! addition
306  }
307  ///! LAB 2 E
```

Рис. 7. Реализация функции *list_pop_exact()* в файле *list.c*

> Thread.h

В структуру процесса были внесены следующие изменения:

```
96  ///! LAB 2 S
97      int old_priority;                /* Original priority of process. */
98      struct list donors_list;         /* Contains priority donors. */
99      struct list_elem donors_elem;    /* List element for donators list. */
100     struct lock *lock_waiter;        /* Pointer to lock. */
101  ///! LAB 2 E
```

Рис. 5. Изменения в структуре процесса.

Были добавлены поля для сохранения изначального приоритета процесса, для сохранения списка “доноров” приоритета, а также был добавлен указатель на замок, который ожидает данный процесс.

Также в этот файл был добавлен прототип функции *compare_thread_priority(...)*, которая была описана в таблице выше.

> Thread.c

В функцию *thread_create(..)* было добавлено учитьвание приоритета процесса при его последующей планировке:


```

209  ///! LAB 2 S
210  /* Add to run queue. */
211  thread_unblock (t);
212  if (priority > thread_current()->priority) {
213      thread_yield(); ///! change
214  }
215  ///! LAB 2 E

```

Рис. 6. Изменения в функции `thread_create()`.

Была изменена функция `thread_set_priority()` так, чтобы она учитывала старый приоритет процесса, который записан в `old_priority` поле процесса.

```

344  ///! LAB 2 S
345  /* Sets the current thread's priority to NEW_PRIORITY.
346   * If thread didn't donated his priority, then NEW_PRIORITY
347   * is just set, otherwise it is set in OLD_PRIORITY field */
348  void
349  thread_set_priority (int new_priority)
350  {
351      ASSERT (!intr_context ());
352      enum intr_level old_level = intr_disable ();
353      struct thread *curr = thread_current();
354      bool need_to_yield = (curr->old_priority > new_priority) ? true : false;
355
356      if (list_empty(&curr->donors_list)) {
357          curr->priority = curr->old_priority = new_priority;
358      } else {
359          curr->old_priority = new_priority;
360      }
361
362      if (need_to_yield) {
363          thread_yield();
364      }
365
366      intr_set_level(old_level);
367  }
368  ///! LAB 2 E

```

Рис. 7. Изменения в функции `thread_set_priority()`.

Изменения затронули и функцию `init_thread()`, в которую была добавлена инициализация добавленных ранее новых полей:

```

493      ///!LAB 2 S
494      t->old_priority = priority;
495      t->lock_waiter = NULL;
496      list_init(&t->donors_list);
497      ///!LAB 2 E

```

Рис. 7. Изменения в функции *init_thread()*.

Изменена функция планировщика *next_thread_to_run()* таким образом, чтобы она выбирала процесс с наивысшим приоритетом из списка процессов готовых к исполнению. Правильный выбор осуществляется с помощью определенной ранее функции *list_pop_max()*:

```

514  ///! LAB 2 S
515  /* Compares priority of A and B given threads. Returns TRUE if
516  | priority of A less then B priority. */
517  bool
518  compare_thread_priority (struct list_elem *a, struct list_elem *b, void *aux UNUSED)
519  {
520      return (list_entry(a, struct thread, elem)->priority <
521              list_entry(b, struct thread, elem)->priority); ///! addition
522  }
523
524  /* Chooses and returns the next thread to be scheduled. Should
525  | return a thread from the run queue, unless the run queue is
526  | empty. (If the running thread can continue running, then it
527  | will be in the run queue.) If the run queue is empty, return
528  | idle_thread. */
529  static struct thread *
530  next_thread_to_run (void)
531  {
532      if (list_empty (&ready_list))
533          return idle_thread;
534      else
535          return list_entry (list_pop_max(&ready_list, &compare_thread_priority), struct thread, elem);
536  }
537  ///! LAB 2 E

```

Рис. 8. Изменения в функции *next_thread_to_run()*;

> Synch.h

В заголовочный файл был добавлен только прототип функции *compare_semaphore_priority()*, которая также была описана в таблице выше.

> Synch.c

Были внесены изменения в функцию “V” семафора, для того чтобы пробуждать процесс из списка ждущих с наивысшим приоритетом, а также учитывать его приоритет при планировании:

```
104  ///! LAB 2 S
105  /* Up or "V" operation on a semaphore. Increments SEMA's value
106  and wakes up one thread of those waiting for SEMA, if any.
107
108  This function may be called from an interrupt handler. */
109  void
110  sema_up (struct semaphore *sema)
111  {
112      ASSERT (sema != NULL);
113      enum intr_level old_level = intr_disable ();
114      struct thread *next_thread = NULL;
115
116      if (!list_empty (&sema->waiters)) {
117          next_thread = list_entry (list_pop_max(&sema->waiters, &compare_thread_priority),
118                                  struct thread, elem);
119          thread_unblock (next_thread); ///! change
120      }
121
122      sema->value++;
123      intr_set_level (old_level);
124
125      if (next_thread && next_thread->priority > thread_current()->priority) {
126          thread_yield(); ///! change
127      }
128  }
129  ///! LAB 2 E
```

Рис. 9. Изменения в функции семафора `sema_up()`.

Для создания механизма распределения приоритета было разработано несколько функций. Рассмотрим назначение этих функций и их реализации:

`void priority_donate(...)` - функция, которая осуществляет передачу приоритета, если целевой процесс в этом нуждается, а также добавляет указатель на отправителя приоритета в список доноров целевого процесса.

```
192  ///! LAB 2 S
193  /* Function checks, does RECEIVER needs more priority to continue
194  executing, donates priority and adds SENDER do donors_list if he donates priority
195  and if APPEND flag is true*/
196  void priority_donate(struct thread* sender, struct thread *receiver, bool append)
197  {
198      if (receiver->priority < sender->priority) {
199          receiver->priority = sender->priority;
200      }
201
202      if (append) {
203          list_push_back(&receiver->donors_list, &sender->donors_elem);
```

Рис. 10. Реализация функции *priority_donate()*.

void find_lock(...) - функция-фильтр, которая используется при итерации по списку с помощью функции ***find_exact(...)*** описанной выше. Возвращает элемент, указатель на замок, в котором совпадает с переданным в функцию.

```
207  /* Function checks if elem A has LOCK lock in lock_waiter field. */
208  bool find_lock(struct list_elem *a, struct lock *lock) {
209      return list_entry(a, struct thread, donors_elem)->lock_waiter == lock;
210  }
```

Рис. 11. Реализация функции *find_lock()*.

void revert_priority_donation(...) - функция, которая находит процесс донор в списке доноров, удаляет указатель на него из списка и возвращает процессу донору свой изначальный приоритет.

```
215  void revert_priority_donation(struct thread* receiver, struct lock *lock)
216  {
217      receiver->priority = receiver->old_priority;
218      struct list_elem *result;
219
220      do
221      {
222          result = list_pop_exact(&receiver->donors_list, &find_lock, lock);
223      } while (result != NULL);
224
225      struct list_elem *elem = list_begin(&receiver->donors_list);
226      struct thread *sender;
227
228      while (elem != list_end(&receiver->donors_list))
229      {
230          sender = list_entry(elem, struct thread, donors_elem);
231          if (sender->priority > receiver->priority)
232              receiver->priority = sender->priority;
233
234          elem = list_next(elem);
235      }
```

Рис. 12. Реализация функции *revert_priority_donate()*.

Для того чтобы использовать систему пожертвования приоритетов, были внесены изменения в функции ***lock_acquire()*** и ***lock_release()***. В функции ***lock_acquire()*** происходит пожертвование приоритета процессу, занимающему критическую секцию до тех пор, пока он не освободит замок.

```

247 lock_acquire (struct lock *lock)
248 {
249     ASSERT (lock != NULL);
250     ASSERT (!intr_context ());
251     ASSERT (!lock_held_by_current_thread (lock));
252
253     if (lock->holder != NULL)
254     {
255         priority_donate(thread_current(), lock->holder, true);
256         thread_current()->lock_waiter = lock;
257
258         struct thread *curr = thread_current();
259         struct thread *holder = lock->holder;
260
261         while (holder->lock_waiter)
262         {
263             curr = holder;
264             holder = holder->lock_waiter->holder;
265             priority_donate(curr, holder, false);
266         }
267     }
268
269     sema_down (&lock->semaphore);
270     thread_current()->lock_waiter = NULL;
271     lock->holder = thread_current ();
272 }

```

Рис. 13. Изменения в функции *lock_acquire()*.

Так же в функцию *lock_release()* был добавлен вызов функции, отменяющей пожертвование приоритета, принцип работы которой был описан выше.

```

301 void
302 lock_release (struct lock *lock)
303 {
304     ASSERT (lock != NULL);
305     ASSERT (lock_held_by_current_thread (lock));
306
307     revert_priority_donation(thread_current(), lock);
308
309     lock->holder = NULL;
310     sema_up (&lock->semaphore);
311 }
312 /// LAB 2 E

```

Рис. 14. Изменения в функции *lock_release()*.

Так же были внесены изменения в реализацию монитора. В описание элемента семафора было добавлено поле с приоритетом, которое, при инициализации монитора, устанавливалось равным приоритету процесса.

```

325 /// LAB 2 S
326 /* One semaphore in a list. */
Nikita Mishenev, October 23rd, 2024 4:16 AM | 2 authors (You and one other)
327 struct semaphore_elem
328 {
329     struct list_elem elem;           /* List element. */
330     struct semaphore semaphore;    /* This semaphore. */
331     int priority;                   /* This semaphore's element priority */
332 };
333 /// LAB 2 E

```

Рис. 15. Изменения в описании элемента `semaphore_elem`.

```

368 cond_wait (struct condition *cond, struct lock *lock)
369 {
370     struct semaphore_elem waiter;
371
372     ASSERT (cond != NULL);
373     ASSERT (lock != NULL);
374     ASSERT (!intr_context ());
375     ASSERT (lock_held_by_current_thread (lock));
376
377     waiter.priority = thread_current()->priority; /// addition
378
379     sema_init (&waiter.semaphore, 0);

```

Рис. 16. Изменения в функции `cond_wait()`.

Изменения были также внесены и в функцию `cond_wait()`, такие, чтобы пробуждался семафор с наивысшим приоритетом процесса, ожидающего его разблокировку.

```

403 cond_signal (struct condition *cond, struct lock *lock UNUSED)
404 {
405     ASSERT (cond != NULL);
406     ASSERT (lock != NULL);
407     ASSERT (!intr_context ());
408     ASSERT (lock_held_by_current_thread (lock));
409
410     if (!list_empty (&cond->waiters))
411         sema_up (&list_entry (list_pop_max(&cond->waiters, &compare_semaphore_priority),
412             struct semaphore_elem, elem)->semaphore); /// change
413 }
414 /// LAB 2 E

```

Рис. 17. Изменения функции `cond_signal()`

6. Описание тестовых задач: имя теста и описание действий, которые в нем совершаются; полученные при запуске результаты, анализ полученных результатов.

alarm-priority - проверка на корректность порядка пробуждения процессов с учетом приоритетов.

priority-change - проверка корректности порядка пробуждения процессов при изменении приоритета.

priority-fifo - проверка на корректность используемого алгоритма планирования (Round Robin).

priority-preempt - проверка на то, что поток с большим приоритетом выполняется в приоритетном режиме.

priority-sema - проверка на корректность порядка пробуждения процессов на семафоре.

priority-condvar - проверка на корректность порядка пробуждения процессов на мониторе.

priority-donate-one - проверка на корректность порядка передачи замка в соответствии с приоритетами созданных основным потоком процессов

priority-donate-lower - проверка на корректность последовательности действий. Понижение приоритета основного процесса не должно иметь эффект до завершения передачи приоритета.

priority-donate-multiple - проверка на корректность последовательности передачи приоритетов. Основной процесс должен в правильном порядке вернуть приоритеты процессам-донорам.

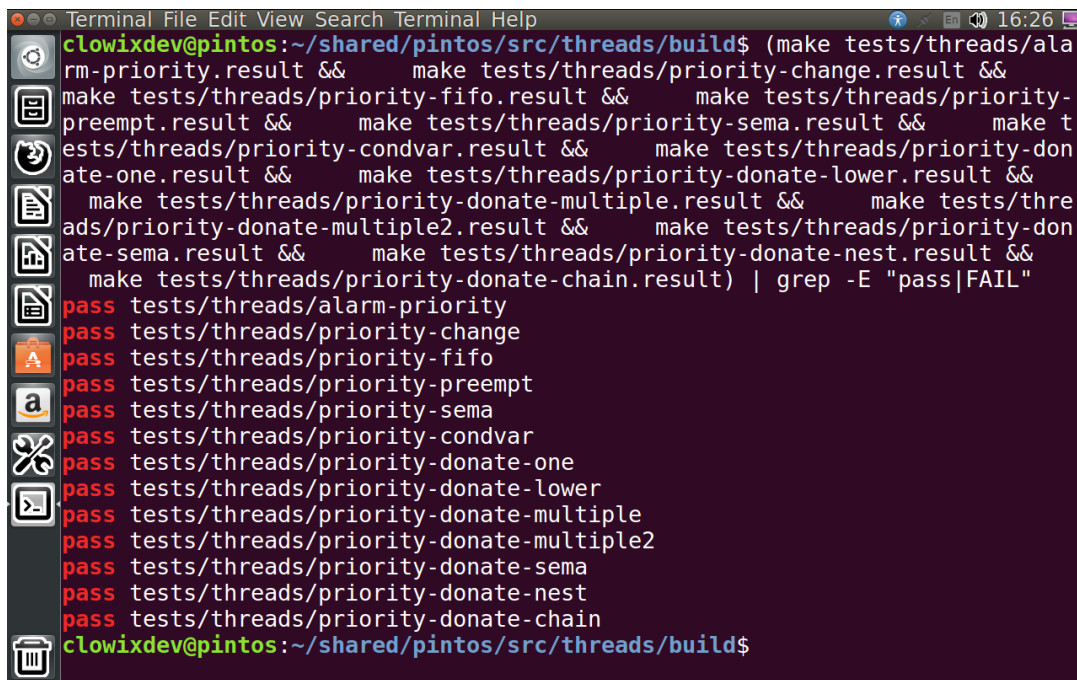
priority-donate-multiple2 - отличается от предыдущего теста, порядком возврата приоритетов.

priority-donate-sema - проверка механизма пожертвования приоритетов на семафоре.

priority-donate-nest - проверка множественного глубокого наследования на семафорах и замках

priority-donate-chain - проверка сохранения приоритетов при многократном последовательном пожертвовании, вырождающемся в “цепь”

При запуске тестов локально на виртуальной машине, все тесты проходятся успешно.



```
Terminal File Edit View Search Terminal Help
clowixdev@pintos:~/shared/pintos/src/threads/build$ (make tests/threads/alarm-priority.result && make tests/threads/priority-change.result &&
make tests/threads/priority-fifo.result && make tests/threads/priority-preempt.result && make tests/threads/priority-sema.result && make t
ests/threads/priority-condvar.result && make tests/threads/priority-donate-one.result && make tests/threads/priority-donate-lower.result &&
make tests/threads/priority-donate-multiple.result && make tests/threads/priority-donate-multiple2.result && make tests/threads/priority-donate-sema.result &&
make tests/threads/priority-donate-nest.result && make tests/threads/priority-donate-chain.result) | grep -E "pass|FAIL"
pass tests/threads/alarm-priority
pass tests/threads/priority-change
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/priority-condvar
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-lower
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-chain
clowixdev@pintos:~/shared/pintos/src/threads/build$
```

Рис. 18. Результаты запуска необходимых тестов.

ВЫВОД

В ходе работы был изучен механизм планирования процессов, также был разработан алгоритм приоритетного планирования и внедрён в учебную операционную систему Pintos.