

Content

I.	Motivation	2
II.	Implementation	3
	Part1:	3
	Part2:	5
III.	Result	9
	Part1:	9
	Part2:	10

Reference

1. Nachos Project1 Report
<https://github.com/taldehyde/os-project-1/blob/master/report.md>
2. OS::NachOS::HW1
<http://blog.terrynini.tw/tw/OS-NachOS-HW1/>
3. 向 NachOS 4.0 作業進發 (2)
<http://morris821028.github.io/2014/05/30/lesson/hw-nachos4-2/>

I. Motivation

Motivation and the problem analysis

Part1: 撰寫自己的 Sleep function，將該 Thread 進入休眠。

Part2: 完成排程處理，在原始的 NachOS 4.0 中，基礎排程是 **Round-Robin**。

本次嘗試加入 FCFS 先來先服務、SJF 最短工作優先、測試不同排程處理

What's your plan to deal with the problem

Part1: 定義新的 system call code 並且宣告 **void Sleep(int time)** 函數，並新增新的 exception 來處理 SC_Sleep system call

預計的工作如下：

1. 在 /code/userprog/system.h 中加入 Sleep(int time)
2. 找一下 assembly language
3. 增加新的 exception 來處理 SC_Sleep system call
4. 由於 kernel 存有 alarm，也就是說每隔固定一段間，就會呼叫 Alarm::CallBack()，因此，對於這個 alarm 來設計一個累加器 Bedroom::_current_interrupt 進行全局去記數，而 Alarm 的設定是每 100 ticks 就會產生一次 interrupt 並呼叫 Alarm::CallBack()
5. 當有程序呼叫 Sleep() 時，會呼叫 WaitUntil()，然後將其丟入 Bedroom 安眠
6. 在 CallBlack() 被呼叫時，去 Bedroom 檢查誰該醒來

Part2:

1. 在 Constructor 中決定要用哪一種類型的排程，並且宣告相對應的 compare function
2. 如果需要搶先設計，則在 Alarm::CallBack() 決定是否要呼叫 interrupt->YieldOnReturn() 查看是否有更需要優先的 process 要執行
3. 在 CPU Burst Time 的計算上，如果採用運行時估計，可以在利用執行 user program 的時間累加器進行計算
4. 分別在在每一個 kernel 宣告的地方 (Initialized()) 都給一個決定 scheduler 類型的參數
5. 希望可以下這樣的 command 來執行

```
$ cd code/threads
$ ./nachos FCFS
$ ./nachos SJF
$ ./nachos Priority
$ ./nachos RR
```

II. Implementation

Part1:

1. 定義 Sleep() 函數

```
31 #define SC_ThreadYield 10
32 #define SC_PrintInt 11
33
130 void PrintInt(int number); //my System Call
131 #endif /* IN_ASM */
```

Original Text

```
31 #define SC_ThreadYield 10
32 #define SC_PrintInt 11
33 #define SC_Sleep 12
34
131 void PrintInt(int number); //my System Call
132
133 void Sleep(int number);
134 #endif /* IN_ASM */
```

Changed Text

/code/userprog/syscall.h

2. 在 start.s 中修改 assembly language

.globl(c++ 寫的部分才能找得到) .ent Sleep(Sleep 的 entry 在這裡)

其餘部分大概跟其他 function 一樣，依樣畫葫蘆

```
139 .end PrintInt
140
141 /* dummy function to keep gcc happy */
142 .globl __main
143 .ent main
```

Original Text

```
139 .end PrintInt
140
141 .globl Sleep
142 .ent Sleep
143 Sleep:
144     addiu $2,$0,SC_Sleep
145     syscall
146     j $31
147     .end Sleep
148
149 /* dummy function to keep gcc happy */
150 .globl __main
151 .ent main
```

Changed Text

/code/test/start.s

3. 修改 exception.cc, 收到 SC_Sleep 這個 syscall 編號時要做的行為

```
66 cout << "Print integer:" << val << endl;
67 return;
68 /* case SC_Exec:
```

Original Text

```
66 cout << "Print integer:" << val << endl;
67 return;
68 case SC_Sleep:
69     val=kernel->machine->ReadRegister(4);
70     kernel->alarm->WaitUntil(val);
71     return;
72 /* case SC_Exec:
```

Changed Text

/code/userprog/exception.cc

4. 必須實作 kernel->alarm->WaitUntil(val); 設計累加器 Bedroom::_current_interrupt。當有程序呼叫 Sleep() 時，會呼叫 WaitUntil(), 然後將其丟入 Bedroom 安眠。

```
23 #include "timer.h"
24
25 // The following class defines a software alarm clock.
26 class Alarm : public CallBackObj {
27 public:
28     Alarm(bool doRandomYield); // Initialize the timer, and callback
29     // to "toCall" every time slice.
30     ~Alarm() { delete timer; }
31
32     void WaitUntil(int x); // suspend execution until time > now + x
33
34 private:
35     Timer *timer; // the hardware timer device
36     void CallBack(); // called when the hardware
```

Original Text

```
23 #include "timer.h"
24 #include "thread.h"
25 #include <list>
26
27 class Bedroom {
28 public:
29     Bedroom():_current_interrupt(0) {};
30     void PutToBed(Thread *t, int x);
31     bool MorningCall();
32     bool IsEmpty();
33 private:
34     class Bed {
35     public:
36         Bed(Thread* t, int x):
37             sleeper(t), when(x) {};
38         Thread* sleeper;
39         int when;
40     };
41
42     int _current_interrupt;
43     std::list<Bed> _beds;
44 };
45
46 // The following class defines a software alarm clock.
47 class Alarm : public CallBackObj {
48 public:
49     Alarm(bool doRandomYield); // Initialize the timer, and callback
50     // to "toCall" every time slice.
51     ~Alarm() { delete timer; }
52
53     void WaitUntil(int x); // suspend execution until time > now + x
54
55 private:
56     Timer *timer; // the hardware timer device
57     Bedroom *_bedroom;
58     void CallBack(); // called when the hardware
```

Changed Text

/code/threads/alarm.h

5. 在 CallBlack()被呼叫時，來去 Bedroom 檢查誰該醒來。WaitUntil 的定義

```

53 MachineStatus status = interrupt->getStatus();
54
55 if (status == IdleMode) { // is it time to quit?
56     if (!interrupt->AnyFutureInterrupts()) {
57         timer->Disable(); // turn off the timer
58     }
59 } else { // there's someone to preempt
60     interrupt->YieldOnReturn();
61 }
62 }

```

Original Text

```

53 MachineStatus status = interrupt->getStatus();
54 bool woken = _bedroom.MorningCall();
55
56 if (status == IdleMode && !woken && _bedroom.IsEmpty())
57 { // is it time to quit?
58     if (!interrupt->AnyFutureInterrupts()) {
59         timer->Disable(); // turn off the timer
60     }
61 } else { // there's someone to preempt
62     //interrupt->YieldOnReturn();
63     if(kernel->scheduler->getSchedulerType() == RR ||
64         kernel->scheduler->getSchedulerType() == Priority ) {
65         // interrupt->YieldOnReturn();
66         interrupt->YieldOnReturn();
67     }
68 }
69 }
70
71 void Alarm::WaitUntil(int x) {
72     IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
73     Thread* t = kernel->currentThread;
74     // burst time
75     int worktime = kernel->stats->userTicks - t->getStartTime();
76     t->setBurstTime(t->getBurstTime() + worktime);
77     t->setStartTime(kernel->stats->userTicks);
78     cout << "Alarm:WaitUntil go sleep" << endl;
79     _bedroom.PutToBed(t, x);
80     kernel->interrupt->SetLevel(oldLevel);
81 }
82
83 bool Bedroom::IsEmpty() {
84     return _beds.size() == 0;
85 }
86
87 void Bedroom::PutToBed(Thread*t, int x) {
88     ASSERT(kernel->interrupt->getLevel() == IntOff);
89     _beds.push_back(Bed(t, _current_interrupt + x));
90     t->Sleep(false);
91 }
92
93 bool Bedroom::MorningCall() {
94     bool woken = false;
95     _current_interrupt ++;
96     for(std::list<Bed>::iterator it = _beds.begin();
97         it != _beds.end(); ) {
98         if(_current_interrupt >= it->when) {
99             woken = true;
100             cout << "Bedroom::MorningCall Thread woken" << endl;
101             kernel->scheduler->ReadyToRun(it->sleeper);
102             it = _beds.erase(it);
103         } else {
104             it++;
105         }
106     }
107     return woken;
108 }

```

Changed Text

</code/threads/alarm.cc>

Part2:

1. 依照助教投影片指示先設計 main.cc，接著設計標頭檔

```
76  DEBUG(dbgThread, "Entering main");
77
78  kernel = new KernelType(argc, argv);
79  kernel->Initialize();
80
```

Original Text

```
76  DEBUG(dbgThread, "Entering main");
77  //
78  SchedulerType type = RR;
79  if(strcmp(argv[1], "FCFS") == 0) {
80      type = FIFO;
81  } else if (strcmp(argv[1], "SJF") == 0) {
82      type = SJF;
83  } else if (strcmp(argv[1], "PRIORITY") == 0) {
84      type = Priority;
85  } else if (strcmp(argv[1], "RR") == 0) {
86      type = RR;
87  }
88  //
89  kernel = new KernelType(argc, argv);
90  kernel->Initialize(type);
91
```

Changed Text

</code/threads/main.cc>

2. 為了實作多種不同的 scheduler，先定義 enum function 並讓程式可以藉由 [- SchedulerType]來切換排程模式，在 SchedulerType 中多增加 FIFO

```
20  enum SchedulerType {
21      RR,    // Round Robin
22      SJF,
23      Priority
24  };
25
26  class Scheduler {
27  public:
28      Scheduler();    // Initialize list of ready threads
29
30      ~Scheduler();    // De-allocate ready list
31
32      void ReadyToRun(Thread* thread);
33      Thread* FindNextToRun();    // Thread can be dispatched.
34      // Dequeue first thread on the ready
35      // list, if any, and return thread.
36      void Run(Thread* nextThread, bool finishing);
37      // Cause nextThread to start running
38      void CheckToBeDestroyed();    // Check if thread that had been
39      // running needs to be deleted
40      void Print();    // Print contents of ready list
41
42      // SelfTest for scheduler is implemented in class Thread
43
44  private:
```

Original Text

```
20  enum SchedulerType {
21      RR,    // Round Robin
22      SJF,
23      Priority,
24      FIFO
25  };
26
27  class Scheduler {
28  public:
29      Scheduler();
30      Scheduler(SchedulerType type);    //
31      // Initialize list of ready threads
32      ~Scheduler();    // De-allocate ready list
33
34      void ReadyToRun(Thread* thread);
35      Thread* FindNextToRun();    // Thread can be dispatched.
36      // Dequeue first thread on the ready
37      // list, if any, and return thread.
38      void Run(Thread* nextThread, bool finishing);
39      // Cause nextThread to start running
40      void CheckToBeDestroyed();    // Check if thread that had been
41      // running needs to be deleted
42      void Print();    // Print contents of ready list
43
44      // SelfTest for scheduler is implemented in class Thread
45
46      SchedulerType getSchedulerType() {return schedulerType;}
47      void setSchedulerType(SchedulerType t) {schedulerType = t;}
48
49  private:
```

Changed Text

</code/threads/scheduler.h>

3. 在 Constructor 中決定要用哪一種類型的排程，並且宣告相對應的 compare function

```
32  Scheduler::Scheduler()
33  {
34      schedulerType = type;
35      readyList = new List<Thread*>;
36      toBeDestroyed = NULL;
```

```
32  int SJFCompare(Thread *a, Thread *b) {
33      if(a->getBurstTime() == b->getBurstTime())
34          return 0;
35      return a->getBurstTime() > b->getBurstTime() ? 1 : -1;
36  }
37
38  int PriorityCompare(Thread *a, Thread *b) {
39      if(a->getPriority() == b->getPriority())
40          return 0;
41      return a->getPriority() > b->getPriority() ? 1 : -1;
42  }
43
44  int FIFOCompare(Thread *a, Thread *b) {
45      return 1;
46  }
47
48  Scheduler::Scheduler()
49  {
50      schedulerType = type;
51      readyList = new List<Thread*>;
52      toBeDestroyed = NULL;
53  }
54  Scheduler::Scheduler(SchedulerType type)
55  {
56      schedulerType = type;
57      switch(schedulerType) {
58      case RR:
59          readyList = new List<Thread*>;
60          break;
61      case SJF:
62          readyList = new SortedList<Thread*>(SJFCompare);
63          break;
64      case Priority:
65          readyList = new SortedList<Thread*>(PriorityCompare);
66          break;
67      case FCFS:
68          readyList = new SortedList<Thread*>(FIFOCompare);
69          break;
70      }
71      toBeDestroyed = NULL;
```

Original Text

Changed Text

</code/threads/scheduler.cc>

4. 在 Alarm::CallBack()決定是否要呼叫 interrupt->YieldOnReturn()查看是否有更需要優先的 process 要執行

```

49 void
50 Alarm::CallBack()
51 {
52     Interrupt *interrupt = kernel->interrupt;
53     MachineStatus status = interrupt->getStatus();
54
55     if (status == IdleMode) { // is it time to quit?
56         if (!interrupt->AnyFutureInterrupts()) {
57             timer->Disable(); // turn off the timer
58         }
59     } else { // there's someone to preempt
60         interrupt->YieldOnReturn();
61     }
62 }

```

```

49 void
50 Alarm::CallBack()
51 {
52     Interrupt *interrupt = kernel->interrupt;
53     MachineStatus status = interrupt->getStatus();
54     bool woken = _bedroom.MorningCall();
55
56     if (status == IdleMode && !woken && _bedroom.IsEmpty()) { // is it time to quit?
57         if (!interrupt->AnyFutureInterrupts()) {
58             timer->Disable(); // turn off the timer
59         }
60     } else { // there's someone to preempt
61         //interrupt->YieldOnReturn();
62         if (kernel->scheduler->getSchedulerType() == RR ||
63             kernel->scheduler->getSchedulerType() == Priority) {
64             interrupt->YieldOnReturn();
65         }
66     }
67 }
68
69

```

kernel->stats->userTicks 是執行 user program 的時間累加器

```

// ... (Original code for Alarm::WaitUntil and Bedroom::PutToBed)

```

```

71 void Alarm::WaitUntil(int x) {
72     intStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
73     Thread* t = kernel->currentThread;
74     // burst time
75     int workTime = kernel->stats->userTicks - t->getStartTime();
76     t->setBurstTime(t->getBurstTime() + workTime);
77     t->setStartTime(kernel->stats->userTicks);
78     cout << "Alarm::WaitUntil go sleep" << endl;
79     _bedroom.PutToBed(t, x);
80     kernel->interrupt->SetLevel(oldLevel);
81 }
82
83 bool Bedroom::IsEmpty() {
84     return _beds.size() == 0;
85 }
86
87 void Bedroom::PutToBed(Thread* t, int x) {
88     ASSERT(kernel->interrupt->getLevel() == IntOff);
89     _beds.push_back(Bed(t, _current_interrupt + x));
90     t->Sleep(false);
91 }
92
93 bool Bedroom::MorningCall() {
94     bool woken = false;
95     _current_interrupt++;
96     for (std::list<Bed>::iterator it = _beds.begin();
97          it != _beds.end(); ) {
98         if (_current_interrupt >= it->when) {
99             woken = true;
100             cout << "Bedroom::MorningCall Thread woken" << endl;
101             kernel->scheduler->ReadyToRun(it->sleeper);
102             it = _beds.erase(it);
103         } else {
104             it++;
105         }
106     }
107     return woken;
108 }

```

Original Text

Changed Text

</code/threads/alarm.cc>

5. 為了測試排程的結果，多增加一個 Thread::SchedulingTest()的函式，

Name	A	B	C	D
Priority	5	1	3	2
Burst Time	3	9	7	3

並設置各程式的 burst time 和 priority 特性，各個程式特性分別設置如上表，

```

436 SimpleThread(0);
437 }

```

```

438 SimpleThread(0);
439 }
440 // -----
441 // Test scheduler
442 // -----
443 void
444 threadBody() {
445     Thread *thread = kernel->currentThread;
446     while (thread->getBurstTime() > 0) {
447         thread->setBurstTime(thread->getBurstTime() - 1);
448         kernel->interrupt->OneTick();
449         printf("%s: remaining %d\n", kernel->currentThread->getName(), kernel->currentThread->getBurstTime());
450     }
451 }
452 void
453 Thread::SchedulingTest()
454 {
455     const int thread_num = 4;
456     char *name[thread_num] = {"A", "B", "C", "D"};
457     int thread_priority[thread_num] = {5, 1, 3, 2};
458     int thread_burst[thread_num] = {3, 9, 7, 3};
459
460     Thread *t;
461     for (int i = 0; i < thread_num; i++) {
462         t = new Thread(name[i]);
463         t->setPriority(thread_priority[i]);
464         t->setBurstTime(thread_burst[i]);
465         t->Fork((VoidFunctionPtr) threadBody, (void *)NULL);
466     }
467     kernel->currentThread->Yield();
468 }

```

Original Text

Changed Text

</code/threads/thread.cc>

6. 必須在 `class Thread` 增加 `setPriority()`, `setBurstTime()`, `SchedulingTest()` ... 等 method header。

```
109
110
111 private:
112     // some of the private data for this class is listed above
```

Original Text

```
109
110
111 void setBurstTime(int t) {burstTime = t;}
112 int getBurstTime() {return burstTime;}
113 void setStartTime(int t) {startTime = t;}
114 int getStartTime() {return startTime;}
115 void setPriority(int t) {execPriority = t;}
116 int getPriority() {return execPriority;}
117 static void SchedulingTest();
118 private:
119     // some of the private data for this class is listed above
120
121     // morris add
122     int burstTime; // predicted burst time
123     int startTime; // the start time of the thread
124     int execPriority; // the execute priority of the thread
```

Changed Text

[/code/threads/thread.h](#)

7. 將 `testcode` 加入到 `ThreadedKernel` 中。

```
46 //-----
47
48 void
49 ThreadedKernel::Initialize()
50 {
51     stats = new Statistics(); // collect statistics
52     interrupt = new Interrupt; // start up interrupt handling
53     scheduler = new Scheduler(); // initialize the ready queue
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111 currentThread->SelfTest(); // test thread switching
112
113 // test semaphore operation
114 semaphore = new Semaphore("test", 0);
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Original Text

```
46 //-----
47 void
48 ThreadedKernel::Initialize() {
49     Initialize(RR);
50 }
51
52 void
53 ThreadedKernel::Initialize(SchedulerType type)
54 {
55     stats = new Statistics(); // collect statistics
56     interrupt = new Interrupt; // start up interrupt handling
57     scheduler = new Scheduler(type); // initialize the ready queue
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111 currentThread->SelfTest(); // test thread switching
112 ThreadedKernel::SchedulingTest();
113 // test semaphore operation
114 semaphore = new Semaphore("test", 0);
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Changed Text

[/code/threads/ kernel.cc](#)

8. 後來發現 make 有錯誤，經過上網一查 ref[3] 向 NachOS 4.0 作業進發 (2)

<https://morris821028.github.io/2014/05/30/lesson/hw-nachos4-2/> 問題是發生於我們

在 /threads 下修改 main.cc 的關係，所以必須在每一個 kernel 宣告地方都宣告的地方 (Initialized()) 都給一個決定 scheduler 類型的參數。

```
18 #include "synchdisk.h"
19
20 class SynchDisk;
21 class UserProgKernel : public ThreadedKernel {
22 public:
23     UserProgKernel(int argc, char **argv);
24     ~UserProgKernel();    // deallocate the kernel
25
26     void Initialize();    // initialize the kernel
27
```

Original Text

```
18 #include "synchdisk.h"
19 #include "../threads/scheduler.h"
20
21 class SynchDisk;
22 class UserProgKernel : public ThreadedKernel {
23 public:
24     UserProgKernel(int argc, char **argv);
25     ~UserProgKernel();    // deallocate the kernel
26
27     void Initialize();    // initialize the kernel
28     void Initialize(SchedulerType type);
29
30
```

Changed Text

userprog/userkernel.h

```
50 // UserProgKernel::Initialize
51 // Initialize Nachos global data structures.
52 //-----
53
54 void
55 UserProgKernel::Initialize()
56 {
57     ThreadedKernel::Initialize(); // init multithreading
58
59     machine = new Machine(debugUserProg);

```

Original Text

```
50 // UserProgKernel::Initialize
51 // Initialize Nachos global data structures.
52 //-----
53
54 void
55 UserProgKernel::Initialize()
56 {
57     Initialize(RR);
58 }
59 void
60 UserProgKernel::Initialize(SchedulerType type)
61 {
62     ThreadedKernel::Initialize(type); // init multithreading
63
64     machine = new Machine(debugUserProg);

```

Changed Text

userprog/userkernel.cc

```
14
15 #include "userkernel.h"
16
17 class PostOfficeInput;
18 class PostOfficeOutput;
19
20 class NetKernel : public UserProgKernel {
21 public:
22     NetKernel(int argc, char **argv);
23     ~NetKernel();    // deallocate the kernel
24
25     void Initialize();    // initialize the kernel
26
27
```

Original Text

```
14
15 #include "userkernel.h"
16 #include "../threads/scheduler.h"
17
18 class PostOfficeInput;
19 class PostOfficeOutput;
20
21 class NetKernel : public UserProgKernel {
22 public:
23     NetKernel(int argc, char **argv);
24     ~NetKernel();    // deallocate the kernel
25
26     void Initialize();    // initialize the kernel
27     void Initialize(SchedulerType type);
28
29
```

Changed Text

network/netkernel.h

```
49 //-----
50
51 void
52 NetKernel::Initialize()
53 {
54     UserProgKernel::Initialize(); // init other kernel data structs
55
56     postOfficeIn = new PostOfficeInput(10);
57     postOfficeOut = new PostOfficeOutput(reliability, 10);

```

Original Text

```
49 //-----
50
51 void
52 NetKernel::Initialize() {
53     Initialize(RR);
54 }
55 void
56 NetKernel::Initialize(SchedulerType type)
57 {
58     UserProgKernel::Initialize(type); // init other kernel data structs
59
60     postOfficeIn = new PostOfficeInput(10);
61     postOfficeOut = new PostOfficeOutput(reliability, 10);

```

Changed Text

network/netkernel.cc

Part2:

對應剛剛 Thread::SchedulingTest()的函式，各個程式特性分別設置如下表，

Name	A	B	C	D
Priority	5	1	3	2
Burst Time	3	9	7	3

<pre> r08921005@r08921005-VirtualBox:~/nachos/2/code\$./threads/ nachos FCFS *** thread 0 looped 0 times *** thread 1 looped 0 times *** thread 0 looped 1 times *** thread 1 looped 1 times *** thread 0 looped 2 times *** thread 1 looped 2 times *** thread 0 looped 3 times *** thread 1 looped 3 times *** thread 0 looped 4 times *** thread 1 looped 4 times A: remaining 2 A: remaining 1 A: remaining 0 B: remaining 8 B: remaining 7 B: remaining 6 B: remaining 5 B: remaining 4 B: remaining 3 B: remaining 2 B: remaining 1 B: remaining 0 C: remaining 6 C: remaining 5 C: remaining 4 C: remaining 3 C: remaining 2 C: remaining 1 C: remaining 0 D: remaining 2 D: remaining 1 D: remaining 0 No threads ready or runnable, and no pending interrupts. Assuming the program completed. Machine halting! Ticks: total 2700, idle 160, system 2540, user 0 Disk I/O: reads 0, writes 0 Console I/O: reads 0, writes 0 Paging: faults 0 Network I/O: packets received 0, sent 0 </pre>	<pre> r08921005@r08921005-VirtualBox:~/nachos/2/code\$./threads/ nachos SJF *** thread 0 looped 0 times *** thread 1 looped 0 times *** thread 0 looped 1 times *** thread 1 looped 1 times *** thread 0 looped 2 times *** thread 1 looped 2 times *** thread 0 looped 3 times *** thread 1 looped 3 times *** thread 0 looped 4 times *** thread 1 looped 4 times A: remaining 2 A: remaining 1 A: remaining 0 D: remaining 2 D: remaining 1 D: remaining 0 C: remaining 6 C: remaining 5 C: remaining 4 C: remaining 3 C: remaining 2 C: remaining 1 C: remaining 0 B: remaining 8 B: remaining 7 B: remaining 6 B: remaining 5 B: remaining 4 B: remaining 3 B: remaining 2 B: remaining 1 B: remaining 0 No threads ready or runnable, and no pending interrupts. Assuming the program completed. Machine halting! Ticks: total 2700, idle 160, system 2540, user 0 Disk I/O: reads 0, writes 0 Console I/O: reads 0, writes 0 Paging: faults 0 Network I/O: packets received 0, sent 0 </pre>	<pre> r08921005@r08921005-VirtualBox:~/nachos/2/code\$./threads/ nachos PRIORITY *** thread 0 looped 0 times *** thread 1 looped 0 times *** thread 0 looped 1 times *** thread 1 looped 1 times *** thread 0 looped 2 times *** thread 1 looped 2 times *** thread 0 looped 3 times *** thread 1 looped 3 times *** thread 0 looped 4 times *** thread 1 looped 4 times B: remaining 8 B: remaining 7 B: remaining 6 B: remaining 5 B: remaining 4 B: remaining 3 B: remaining 2 B: remaining 1 B: remaining 0 D: remaining 2 D: remaining 1 D: remaining 0 C: remaining 6 C: remaining 5 C: remaining 4 C: remaining 3 C: remaining 2 C: remaining 1 C: remaining 0 A: remaining 2 A: remaining 1 A: remaining 0 No threads ready or runnable, and no pending interrupts. Assuming the program completed. Machine halting! Ticks: total 2800, idle 110, system 2690, user 0 Disk I/O: reads 0, writes 0 Console I/O: reads 0, writes 0 Paging: faults 0 Network I/O: packets received 0, sent 0 </pre>
./threads/nachos FCFS	./threads/nachos SJF	./threads/nachos PRIORITY

分別測試 FCFS、SJF、Priority，討論其結果

FCFS	確實是按照先來後到的順序，也就是 ABCD 的順序做執行
SJF	具備最長時間 CPU-Burst Time 的 B 程序確實是最後一個執行的， 然而同時是最短長度的兩個程序 A 與 D 是最先被執行的
Priority	確實是優先等地的順序，也就是 BDCA(1235)的順序做執行