به نام خدا

نام استاد : الهام علیقارداش

اعضای گروه : صادق رنجبر و مهرداد عابدی

پروژه درس سیستم عامل ترم 961

پیاده سازی الگوریتم ها SJF وFSCF و اولویتی غیر انحصاری در Nachos

پوشه اجرایی پروژه در همین فایل با نام OsProject موجود میباشد . برای اجرا و دیدن خروجی کافیست دستور make را در این دایرکتوری اجرا کنید . که سه فایل FCSF و SJF و NPP ایجاد میشود که هرکدام الگوریتم خود را برروی یک تست کیس اجرا میکنند . نحوه تغییر و مشاهده تست کیس نیست در ادامه ذکر خواهد شد .

گزارش پروژه :

همانطور که در صورت پروژه ذکر شده بود برای پیاده سازی الگوریتم ها کافیست فایل scheduler.cc را تغییر داد که دارای سه تابع ()ReadyToRun و ()FindNextToRun و ()ShouldISwitch میباشد . توضیحات این توابع در صورت پروژه موجود میباشد.

تابع ()ReadyToRun : کد های تغییر داده شده در این تابع در شکل زیر قابل مشاهده میباشد.

```
void
Scheduler::ReadyToRun (Thread *thread)
{
    DEBUG('t', "Putting thread %s on ready list.\n", thread->getName());

    thread->setStatus(READY);
    switch(policy)
    {

      case SCHED_MLQ:
      case SCHED_PRIO_NP: // Nonpreemptive Priority scheduling

        readyList->SortedInsert(thread,thread->MAX_PRIORITY - thread->getPriority());

        break;

      case SCHED_PRIO_P:
      case SCHED_RR:
      case SCHED_FCFS: // FCFS algorithm
        readyList->Append(thread);
        break;

      case SCHED_SJF: // SJF scheduling algorithm
        readyList->SortedInsert(thread,thread->getCBT());
        break;

      default:
        readyList->Append(thread);
        break;

    }
}
```

Fig-1

تابع **FindNextToRun()** : کد های این تابع نیز در Fig-2 آمده است.

```
Thread *
Scheduler::FindNextToRun ()
{
    Thread * next_to_run;
    switch(policy)
    {
     case SCHED_MLQ:
     case SCHED_PRIO_NP:// Nonpreemptive Priority scheduling
       next_to_run = readyList->Remove();
       break;

     case SCHED_PRIO_P:
     case SCHED_RR:
     case SCHED_FCFS:// FCFS algorithm
       next_to_run = readyList->Remove();
        break;



    case SCHED_SJF: // SJF scheduling algorithm
         next_to_run = readyList->Remove();
         break;

     default:
       next_to_run = readyList->Remove();
       break;
    }
    return ( next_to_run );
}
```

Fig-2

تابع ShouldISwitch() : کد های این تابع به شرح زیر است .

```
bool
Scheduler::ShouldISwitch ( Thread  * oldThread, Thread * newThread )
{
   bool doSwitch;
   switch(policy)
   {

    case SCHED_MLQ:
    case SCHED_PRIO_NP:// Nonpreemptive Priority scheduling
      if(oldThread->getPriority() < newThread->getPriority())
        doSwitch = true;
      else
        doSwitch = false;
      break;



    case SCHED_PRIO_P:
    case SCHED_RR:

    case SCHED_FCFS: // FCSF scheduling algorithm
      doSwitch = false;
      break;

    case SCHED_SJF: // SJF scheduling algorithm
      doSwitch = false;
      break;

    default:
      doSwitch = false;
      break;
   }
   return doSwitch;
}
```

Fig-3

الگوریتم FCSF : برای پیاده سازی این الگوریتم همانطور که میدانیم تمامی فرایند ها به ترتیب ورودشان اجرا شده . بنابراین زمانی که یک فرایند درخواست اجرا داشت کافیست آنرا به انتهای صف اماده ها اضافه کرد . همانطور که در Fig-1 میبینیم در شرط های switch قسمت SCHED_FCSFS مربوط به این الگوریتم میباشد . در بدنه این شرط ما باید

فرایندی که به این تابع به عنوان آرگومان ارسال شده است را به انتهای صف فرایند های اماده اضافه کنیم . متود append را بر روی readyList که صف فرایند های آماده میباشد اجرا کرده و آن فرایند را به آن پاس میدهیم .

در تابع FindNextToRun() که وظیفه تصمیم گیری بر این را دارد که پس از اجرای یک فرایند کدام فرایند حاضر در لیست اماده ها پردازنده را در اختیار بگیرد و اجرا کند کافیست از سر صف یک فرایند را برداشته و برای اجرا به پردازنده دهیم . که این کار با متود Remove() بر روی readyList انجام میشود که این کار در Fig-2 در قسمت شرط الگوریتم FCSF خط 107 اضافه میشود و آن فرایند را در نهایت برمیگرداند تا آن اجرا شود.

برای تابع shoudISwitch() که زمانی یک فرایند جدید در حین انجام یک فرایند رسید تصمیم میگیرد آیا باید پردازنده از فرایند در حال اجرا گرفته شده و به فرایند جدید اختصاص داده شود یا خیر که این کار با متغیر doSwitch مشخص میشود . چون الگوریتم FCSF انحصاری است یک فرایند هنگامی که پردازنده را در اختیار میگیرد نمیتوان از آن پردازنده را گرفت بنابراین به سادگی در قسمت شرط FCSF مقدار doSwitch رو برابر False قرار میدهیم .

خروجی الگوریتم FCSF :

برای دیدن خروجی این الگوریتم پس از make کردن سورس کد کافیست دستور FCSF/. را اجرا کنید. مقدار ها AV و CBT و Priority فرایند ها در فایل test.0.cc در پوشه thread قابل مشاهده میباشد . برای تغییر میتوان متغیر های ارایه ایه startTime و burstTime و priority را تغییر داد . هرکدام از فرایند ها نظیر به نظیر میباشند . اگر قصد اضافه کردن تعداد فرایند به این ارایه هارا دارید میبایست مقدار numThreads را نیز تغییر دهید .

پس از اجرای این فرایند خروجی برای تست کیس مشخص شده به صورت زیر میباشد:

```
##########################################
 Sadegh Ranjbar has created this files
##########################################
Starting at Ticks: total 0
Queuing threads.
Queuing thread P1  at Time 0, priority 1
P1, Starting Burst of  7Ticks: total 0
Queuing thread P2 at Time 1, priority 2
```

```
P1, Still  6to go Ticks: total 1
Queuing thread P3 at Time 2, priority 3
Queuing thread P4 at Time 2, priority 4
P1, Still  5to go Ticks: total 2
P1, Still  4to go Ticks: total 3
P1, Still  3to go Ticks: total 4
P1, Still  2to go Ticks: total 5
P1, Still  1to go Ticks: total 6
P1, Still  0to go Ticks: total 7
P1, Done with burst Ticks: total 7
P2, Starting Burst of  6Ticks: total 7
Queuing thread P 5at Time 8, priority 5
P2, Still  5to go Ticks: total 8
P2, Still  4to go Ticks: total 9
P2, Still  3to go Ticks: total 10
P2, Still  2to go Ticks: total 11
P2, Still  1to go Ticks: total 12
P2, Still  0to go Ticks: total 13
P2, Done with burst Ticks: total 13
P3, Starting Burst of  1Ticks: total 13
P3, Still  0to go Ticks: total 14
P3, Done with burst Ticks: total 14
P4, Starting Burst of  3Ticks: total 14
P4, Still  2to go Ticks: total 15
P4, Still  1to go Ticks: total 16
```

```
P4, Still  0to go Ticks: total 17
P4, Done with burst Ticks: total 17
P5, Starting Burst of  4Ticks: total 17
P5, Still  3to go Ticks: total 18
P5, Still  2to go Ticks: total 19
P5, Still  1to go Ticks: total 20
P5, Still  0to go Ticks: total 21
P5, Done with burst Ticks: total 21
No threads ready or runnable, and no pending
interrupts.
Assuming the program completed.
Machine halting!


Ticks: total 21, idle 0, system 21, 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

Cleaning up...



Test Case
      //FCSF scheduling algorithm
        //Process   Burst Time(CBT)   Priority    Start
Time(A.V)
```

```
        //P1          7                   1                   0
        //P2          6                   2                   1
        //P3          1                   3                   2
        //P4          3                   4                   2
        //P5          4                   5                   8
-----------------------------------------------------------//
---------------
```

```
#define UserTick    0.5 // adv
#define SystemTick  1   // adv
#define RotationTime    500
#define SeekTime    500     //
#define ConsoleTime     100 //
#define NetworkTime     100
#define TimerTicks   40     //
```

Fig-4

نتیجه تست کیس بالا با الگوریتم FCSF به این صورت از که فرایند ها p1 تا p5 به ترتیب اجرا شده و درنهایت 21 واحد زمانی طول میکشد تا الگوریتم به پایان برسد .

نکته مهم : در سورس کد اصلی که دانلود شده واحد های زمانی 10 به 10 زیاد میشد. این مقدار را در فایل thread.h تغییر داده ام .

مقدار SystemTick را از 10 به 1 تغییر داده شده است . همچنین در سورس اصلی برای تعویض فرایند و اضافه کردن فرایند به صف اماده یک واحد زمانی اشغال میکند . برای اینکه بیشتر شبیه به تمرین های حل شده در کلاس باشد من این قسمت را حذف کرده و تنها برای انجام فرایند زمان میگیرد .

الگوریتم SJF :

همانطور که میدانیم زمانی که یک فرایند جدید میرسد باید در صف فرایند های اماده بعد از فرآیندی که CBT آن فرایند از CBT فرایند ورودی بیشتر و فرایندی که قبل از این فرایند قرار میگیرد CBT آن کمتر از CBT فرایند ورودی قرار گیرد . برای این کار باید متود SortedInsert را بر روی readyList اجرا کرد (خط 78 Fig-1) که فرایند و همچنین مقدار CBT را برای آن میفرستیم. برای گرفتن مقدار CBT یک فرایند باید کد زیر را به فایل thread.h اضافه کرد تا مقدار آن را به ما برگرداند .

```
05
06      int getCBT(){return machineState[InitialArgState];}
07
```

Fig-5

زمانی که ما CBT را به عنوان sortedKey برای تابع SortedInsert میفرستیم تا وقتی که فرایندی پیدا نکند که CBT ان بیشتر از فرایند فرستاده شده نباشد در لیست قرار نمیدهد .

برای انتخاب فرایند نیز کافیست مانند الگوریتم قبلی اولین فرایند از سر لیست را انتخاب کرد و اجرا کرد . زیرا فرایند ها به صورت مرتب شده در صف قرار دارند .

چون الگوریتم انحصاری مبیاشد برای تابع shouldISwitch نیز کافیست فقط مقدار doSwitch رو برابر با False قرار داد .

خروجی این الگوریتم برای همان تست کیس بالا برابر با زیر است :

```
Shortest job first scheduling
##########################################
 Sadegh Ranjbar has created this files
##########################################
Starting at Ticks: total 0
Queuing threads.
Queuing thread P1 at Time 0, priority 1
P1, Starting Burst of 7 Ticks: total 0
Queuing thread P2 at Time 1, priority 2
P1, Still 6 to go Ticks: total 1
Queuing thread P3 at Time 2, priority 3
Queuing thread P4 at Time 2, priority 4
P1, Still 5 to go Ticks: total 2
P1, Still 4 to go Ticks: total 3
P1, Still 3 to go Ticks: total 4
P1, Still 2 to go Ticks: total 5
P1, Still 1 to go Ticks: total 6
P1, Still 0 to go Ticks: total 7
P1, Done with burst Ticks: total 7
```

```
P4, Starting Burst of 1 Ticks: total 7
Queuing thread P5 at Time 8, priority 5
P4, Still 0 to go Ticks: total 8
P4, Done with burst Ticks: total 8
P3, Starting Burst of 3 Ticks: total 8
P3, Still 2 to go Ticks: total 9
P3, Still 1 to go Ticks: total 10
P3, Still 0 to go Ticks: total 11
P3, Done with burst Ticks: total 11
P5, Starting Burst of 4 Ticks: total 11
P5, Still 3 to go Ticks: total 12
P5, Still 2 to go Ticks: total 13
P5, Still 1 to go Ticks: total 14
P5, Still 0 to go Ticks: total 15
P5, Done with burst Ticks: total 15
P2, Starting Burst of 6 Ticks: total 15
P2, Still 5 to go Ticks: total 16
P2, Still 4 to go Ticks: total 17
P2, Still 3 to go Ticks: total 18
P2, Still 2 to go Ticks: total 19
P2, Still 1 to go Ticks: total 20
P2, Still 0 to go Ticks: total 21
P2, Done with burst Ticks: total 21
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
```

```
Machine halting!


Ticks: total 21, idle 0, system 21, 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


Cleaning up...
```

در این الگوریتم زمانی که p1 پردازنده را در اختیار میگیرد باقی فرایند ها وارد صف میشوند که پس از اتمام فرایند p1 فرایند p4 پردازنده را در اختیار میگیرد چون نسبت به بقیه CBT کمتری دارد .

الگوریتم اولویتی غیر انحصاری :

این یک الگوریتم غیر انحصاری است که بر اساس اولویت تصمیم میگیرد . که بیشترین اولویت باید ابتدا اجرا شود . پس زمانی که یک فرایند وارد میشود باید جایی در صف قرار گیرد که فرایند های بعد از آن همگی اولویت کمتری داشته باشند. از همان تابع sortedInsert استفاده کرده که به عنوان sortedKey یک متغییر MAX_PRIORITY در فایل thread.h تعریف شده که در Fig-6 امده است . مقدار اولویت فرایند را از مقدار MAX_PRIORITY کم کرده که هرچه مقدار کوچکتر شد در ابتدای صف قرار بگیرد . که در Fig-1 خط 67 مشخص میباشد .

```
97      static const int MAX_PRIORITY = 20;
98      static const int NORM_PRIORITY = 10;
99      static const int MIN_PRIORITY = 0;
100
```

Fig-6

در تابع `findNextToRun` نیز مانند الگوریتم های قبلی کافیست فرایندی که در اول صف قرار دارد اجرا شود .

برای تابع `shoudISwitch` چون یک الگوریتم غیر انحصاری است باید تصمیم گرفت زمانی که یک فرایند جدید میرسد باید فرایند در حال اجرا را متوقف کند یا خیر. زمانی که یک فرایند وارد میشود اگر اولویت فرایند ورودی بیشتر از فرایند در حال اجرا باشد . پردازنده از فرایند در حال اجرا گرفته میشود و به فرایند جدید اختصاص داده میشود . این تصمیم گیری در خط 139 تا خط 143 انجام میشود اگر اولویت الگوریتم ورودی بیشتر بود مقدار `doSwitch` برابر با `True` شده . در غیر اینصورت مقدار `False` میباشد .

خروجی این الگوریتم برای همان تست کیس به صورت زیر است :

```
Nonpreemptive Priority scheduling
##########################################
 Sadegh Ranjbar has created this files
```

```
##########################################
Starting at Ticks: total 0
Queuing threads.
Queuing thread P1 at Time 0, priority 7
Queuing thread P2 at Time 0, priority 5
Queuing thread P3 at Time 0, priority 2
Queuing thread P4 at Time 0, priority 8
P4, Starting Burst of 7 Ticks: total 0
P4, Still 6 to go Ticks: total 1
P4, Still 5 to go Ticks: total 2
P4, Still 4 to go Ticks: total 3
P4, Still 3 to go Ticks: total 4
Queuing thread P5 at Time 5, priority 9
P5, Starting Burst of 12 Ticks: total 5
P5, Still 11 to go Ticks: total 6
P5, Still 10 to go Ticks: total 7
P5, Still 9 to go Ticks: total 8
P5, Still 8 to go Ticks: total 9
P5, Still 7 to go Ticks: total 10
P5, Still 6 to go Ticks: total 11
P5, Still 5 to go Ticks: total 12
P5, Still 4 to go Ticks: total 13
P5, Still 3 to go Ticks: total 14
P5, Still 2 to go Ticks: total 15
P5, Still 1 to go Ticks: total 16
```

```
P5, Still 0 to go Ticks: total 17
P5, Done with burst Ticks: total 17
P4, Still 2 to go Ticks: total 17
P4, Still 1 to go Ticks: total 18
P4, Still 0 to go Ticks: total 19
P4, Done with burst Ticks: total 19
P1, Starting Burst of 5 Ticks: total 19
P1, Still 4 to go Ticks: total 20
P1, Still 3 to go Ticks: total 21
P1, Still 2 to go Ticks: total 22
P1, Still 1 to go Ticks: total 23
P1, Still 0 to go Ticks: total 24
P1, Done with burst Ticks: total 24
P2, Starting Burst of 19 Ticks: total 24
P2, Still 18 to go Ticks: total 25
P2, Still 17 to go Ticks: total 26
P2, Still 16 to go Ticks: total 27
P2, Still 15 to go Ticks: total 28
P2, Still 14 to go Ticks: total 29
P2, Still 13 to go Ticks: total 30
P2, Still 12 to go Ticks: total 31
P2, Still 11 to go Ticks: total 32
P2, Still 10 to go Ticks: total 33
P2, Still 9 to go Ticks: total 34
P2, Still 8 to go Ticks: total 35
```

```
P2, Still 7 to go Ticks: total 36
P2, Still 6 to go Ticks: total 37
P2, Still 5 to go Ticks: total 38
P2, Still 4 to go Ticks: total 39
P2, Still 3 to go Ticks: total 40
P2, Still 2 to go Ticks: total 41
P2, Still 1 to go Ticks: total 42
P2, Still 0 to go Ticks: total 43
P2, Done with burst Ticks: total 43
P3, Starting Burst of 13 Ticks: total 43
P3, Still 12 to go Ticks: total 44
P3, Still 11 to go Ticks: total 45
P3, Still 10 to go Ticks: total 46
P3, Still 9 to go Ticks: total 47
P3, Still 8 to go Ticks: total 48
P3, Still 7 to go Ticks: total 49
P3, Still 6 to go Ticks: total 50
P3, Still 5 to go Ticks: total 51
P3, Still 4 to go Ticks: total 52
P3, Still 3 to go Ticks: total 53
P3, Still 2 to go Ticks: total 54
P3, Still 1 to go Ticks: total 55
P3, Still 0 to go Ticks: total 56
P3, Done with burst Ticks: total 56
No threads ready or runnable, and no pending
interrupts.
```

```
Assuming the program completed.
Machine halting!

Ticks: total 56, idle 0, system 56, 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

Cleaning up...
```

در این فرایند در ابتدا p4 اجرا شده زیرا تماما در لحظه 0 وارد شده و اولویت p4 بیشتر از همه است . اما در ثانیه 5 فرایند p5 وارد شده که اولویت آن از p4 بیشتر میباشد . چون الگوریتم غیر انحصاری میباشد پردازنده از p4 گرفته شده و به p5 اختصاص داده میشود . پس از اتمام p5 ادامه p4 نیز اجرا میشود .

نکته : برای تغییر هرکدام از تست کیس ها کافیست فایل مربوط به آن را در پوشه thread تغییر داد . برای تغییر الگوریتم FCSF فایل test.0.cc الگوریتم SJF فایل test.3.cc و الگوریتم اولویتی غیر انحصاری فایل test.1.cc را تغییر دهید . پس از تغییر هر قسمت از کد میبایست دستور make clean;make اجرا شود.