به نام خدا

گزارش فاز سوم پروژه ی درس سیستم های عامل

نام استاد: جناب آقای دکتر جوادی

نام، نام خانوادگی و شماره ی دانشجویی دانشجو: فرشید نوشی- ۹۸۳۱۰۶۸

گزارش

برای انجام این پروژه ایجاد تغییرات لازم در ساختار پردازه های سیستم عامل اعم از قرار دادن متغیر های لازم برای بدست اوردن زمان ایجاد یک پردازه زمان اتمام آن، اولویت آن پردازه و ... لازم بود. برای این کار در فایل proc.h تغییرات را در ساختار پردازه ها اعمال کردیم. بخش زیادی از منطق پیاده سازی قسمت های مختلف برنامه به مانند dmlp, mlp, rr with priority, priority schedulling همگی تابع هایی در فایل proc.c دارند که دو ورودی میگیرد یکی مربوط به context cpu و دیگری پوینتری از جدول پردازه ها و هر کدام مسئولیتشان انتخاب یک پردازه آماده برحسب منطقشان و ورودی هایشان از جدول پردازه ها هست.

برای سوییچ کردن بین پردازه ها از متد context switch استفاده کردیم که در همین فایل هست و خودمان آن را پیاده سازی کرده ایم.

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)

  int stackTop;               // The top of the stack for this process
  int threads;                 // The number of child threads this process has
  int priority;                // The priority of this process
  int full_time_runner;        // How much this process ran the last time
  int quantum_value;           // QUANTUM
  int creation_time;              // when this process was created
  int termination_time;           // when this process was terminated
  int sleeping_time;              // total Time in SLEEPING state
  int runnable_time;              // total Time in RUNNABLE state
  int running_time;               // total Time in RUNNING state

};
```

در متد updateProcTimes زمان های دیگری که در پردازه ها تعریف کرده ایم را بروز رسانی نموده ایم تا مقادیرشان قابل اعتماد همواره باقی بمانند.

```c
int getTurnAroundTime()
{
  return myproc()->sleeping_time + myproc()->runnable_time + myproc()->running_time;
}

int getBurstTime()
{
  return myproc()->running_time;
}

// update time in each process in every states
void updateProcTimes()
{
  struct proc *p;
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
  {
    switch (p->state)
    {
    case RUNNING:
      p->running_time++;
      break;
    case RUNNABLE:
      p->runnable_time++;
      break;
    case SLEEPING:
      p->sleeping_time++;
      break;
    default:
      break;
    }
  }
}
```

برای بدست آوردن تایم های خواسته شده در سوال نیز سیستم کال های مشخصی را در سیستم عامل تعریف کردیم که پیاده سازی برخی از آن ها در تصویر بالا واضح است.

```c
void priority_scheduler(struct cpu *cpu, struct proc *proc)
{
  int best = 10;
  struct proc *p_best = 0;
  acquire(&ptable.lock);
  for (proc = ptable.proc; proc < &ptable.proc[NPROC]; proc++)
  {
    if (proc->state == RUNNABLE && proc->priority < best)
    {
      best = proc->priority;
    }
  }
  if (best != 10)
  {
    for (; proc < &ptable.proc[NPROC]; proc++)
    {
      if (proc->state != RUNNABLE)
        continue;

      if (proc->priority == best)
      {
        p_best = proc;
      }
    }
    context_switch(cpu, p_best);
  }
  release(&ptable.lock);
}
```

تابع بالا به طور نمونه به این صورت است که میان همه ی پردازه ها میچرخد و بهترین پردازه را از نظر اولویت انتخاب میکند و به آن سوییچ میکنیم. اگر هم پردازه ای نبود کاری نمیکنیم. در ادامه نمونه خروجی های برنامه آورده شده اند.

هم چنین تست های نوشته شده برای سوال تست های خود صورت پروژه میباشند. با توجه به خروجی های برنامه برای کوانتوم کم و زیاد به این نتیجه رسیدیم که اگر خیلی زیاد باشد الگوریتم ما شبیه به FCFS میشود و اگر خیلی کم شود بخاطر context switch فراوان throughput کمتری خواهیم داشت و overhead بالا میرود.

```
Pid: 41, Burst Time: 139, Waiting Time: 5148, Turnaround Time: 5287
Pid: 42, Burst Time: 112, Waiting Time: 4748, Turnaround Time: 4860
Pid: 43, Burst Time: 127, Waiting Time: 5014, Turnaround Time: 5141
Pid: 44, Burst Time: 136, Waiting Time: 4861, Turnaround Time: 4997
Pid: 45, Burst Time: 128, Waiting Time: 5036, Turnaround Time: 5164
Pid: 46, Burst Time: 137, Waiting Time: 4911, Turnaround Time: 5048
Pid: 47, Burst Time: 113, Waiting Time: 5055, Turnaround Time: 5168
Pid: 48, Burst Time: 130, Waiting Time: 4957, Turnaround Time: 5087
Pid: 49, Burst Time: 153, Waiting Time: 4984, Turnaround Time: 5137
Pid: 50, Burst Time: 154, Waiting Time: 4991, Turnaround Time: 5145
Pid: 51, Burst Time: 152, Waiting Time: 4973, Turnaround Time: 5125
Pid: 52, Burst Time: 147, Waiting Time: 4930, Turnaround Time: 5077
Pid: 53, Burst Time: 136, Waiting Time: 4990, Turnaround Time: 5126
Pid: 54, Burst Time: 135, Waiting Time: 4978, Turnaround Time: 5113
Pid: 55, Burst Time: 137, Waiting Time: 4952, Turnaround Time: 5089
Pid: 56, Burst Time: 157, Waiting Time: 4969, Turnaround Time: 5126
Pid: 57, Burst Time: 139, Waiting Time: 4975, Turnaround Time: 5114
Pid: 58, Burst Time: 116, Waiting Time: 4888, Turnaround Time: 5004
Pid: 59, Burst Time: 140, Waiting Time: 4950, Turnaround Time: 5090



Avg: bt: 134, tt: 5056, wt: 4922
$ _
```

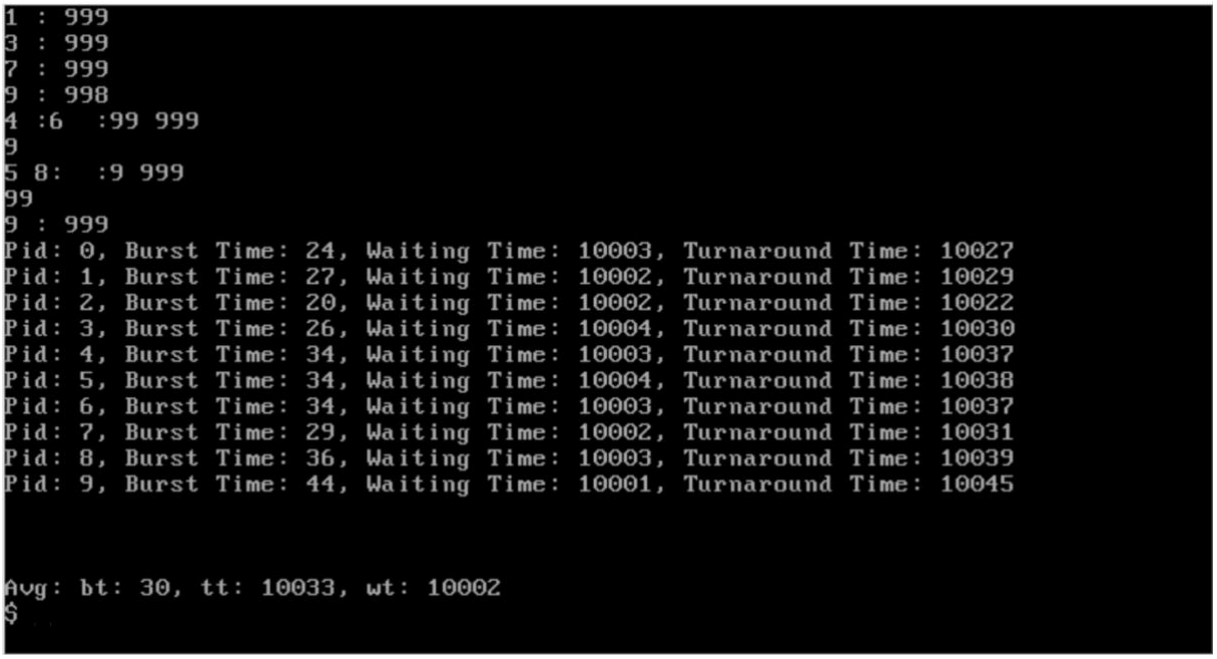
نمونه خروجی برای multi layered priority

```
Pid: 41, Burst Time: 65, Waiting Time: 2307, Turnaround Time: 2372
Pid: 42, Burst Time: 58, Waiting Time: 2311, Turnaround Time: 2369
Pid: 43, Burst Time: 66, Waiting Time: 2349, Turnaround Time: 2415
Pid: 44, Burst Time: 76, Waiting Time: 2308, Turnaround Time: 2384
Pid: 45, Burst Time: 57, Waiting Time: 2365, Turnaround Time: 2422
Pid: 46, Burst Time: 76, Waiting Time: 2288, Turnaround Time: 2364
Pid: 47, Burst Time: 55, Waiting Time: 2368, Turnaround Time: 2423
Pid: 48, Burst Time: 62, Waiting Time: 2329, Turnaround Time: 2391
Pid: 49, Burst Time: 71, Waiting Time: 2368, Turnaround Time: 2439
Pid: 50, Burst Time: 62, Waiting Time: 2218, Turnaround Time: 2280
Pid: 51, Burst Time: 75, Waiting Time: 2301, Turnaround Time: 2376
Pid: 52, Burst Time: 61, Waiting Time: 2345, Turnaround Time: 2406
Pid: 53, Burst Time: 61, Waiting Time: 2165, Turnaround Time: 2226
Pid: 54, Burst Time: 61, Waiting Time: 2363, Turnaround Time: 2424
Pid: 55, Burst Time: 52, Waiting Time: 2347, Turnaround Time: 2399
Pid: 56, Burst Time: 77, Waiting Time: 2343, Turnaround Time: 2420
Pid: 57, Burst Time: 65, Waiting Time: 2248, Turnaround Time: 2313
Pid: 58, Burst Time: 69, Waiting Time: 2341, Turnaround Time: 2410
Pid: 59, Burst Time: 73, Waiting Time: 2288, Turnaround Time: 2361



Avg: bt: 63, tt: 2325, wt: 2262
$
```

نمونه خروجی برای dynamic multi layered priority

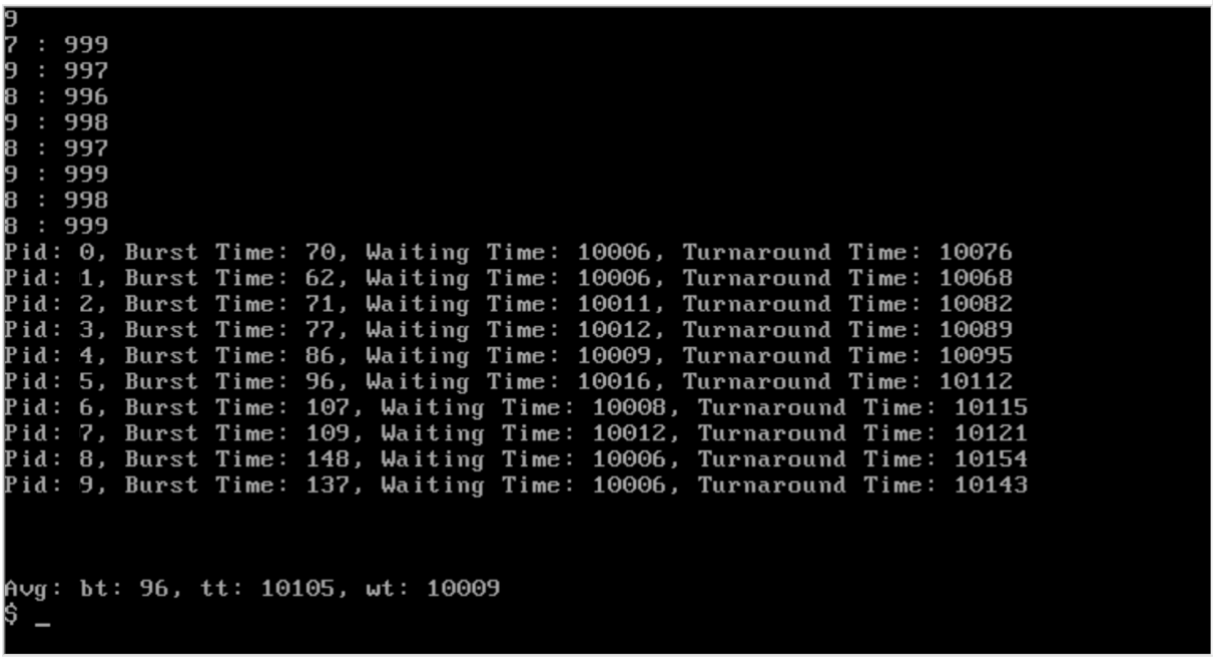
```
Avg times per classes
Priority class: 1 ----> Avg Turnaround: 3368, Avg Waiting: 3166, Avg getBurstTim
e: 201
Priority class: 2 ----> Avg Turnaround: 3362, Avg Waiting: 3155, Avg getBurstTim
e: 206
Priority class: 3 ----> Avg Turnaround: 3386, Avg Waiting: 3186, Avg getBurstTim
e: 200
Priority class: 4 ----> Avg Turnaround: 3353, Avg Waiting: 3162, Avg getBurstTim
e: 191
Priority class: 5 ----> Avg Turnaround: 3276, Avg Waiting: 3112, Avg getBurstTim
e: 164
Priority class: 6 ----> Avg Turnaround: 3207, Avg Waiting: 3057, Avg getBurstTim
e: 150



****Total Avg times****



Avg: bt: 185, tt: 3325, wt: 3139
$ _
```

نمونه خروجی برای priority scheduling



نمونه ی خروجی برای round robin scheduling

در نمونه خروجی های بالا مقدار Quantum در فایل param.h برابر با ۴۰۰ فرض شده بود در نمونه ی زیر این مقدار برابر با 4 گرفته شده است که تعداد context switch ها در آن بسیار زیاد از حد بوده است.



نمونه ی خروجی برای round robin scheduling با کوانتوم زمانی ۴