کارهای انجام شده در این پروژه را می‌توان در دو قسمت خلاصه کرد:

الف) افزودن متغیر dsn_rt_prio به سازه task_struct و تغییر زمانبندی کرنل بر اساس این متغیر (**مسئول : کامیار میرزازاد**)

اولین قدم در پروژه ، افزودن متغیر dsn_rt_prio به سازه task_struct ، که برای مشخص کردن ریسه ها و پردازه ها در سیستم عامل لینوکس به کار می‌رود، بود. این سازه در هدر sched.h تعریف شده است و ما متغیر خودمان را پس از تعریف متغیر normal_prio در خط ۱۰۶۰ این فایل اضافه کردیم. با توجه به فرض موجود در صورت پروژه مبنی بر مقدار اولیه این متغیر ، لازم است این متغیر را در هنگام ایجاد نسخه جدید از سازه task_struct مقداردهی اولیه کنیم. در سیستم عامل لینوکس این کار در هدر init_task.h انجام می‌شود. در این جا نیز همانند sched.h مقداردهی مربوط به dsn_rt_prio را پس از مورد مربوط به normal_prio در خط ۱۷۲ اضافه کردیم.

برای تغییر زمانبندی کرنل به صورت عمل می‌کنیم : درهنگامی که scheduler می‌خواهد پردازه realtime بعدی را انتخاب کند، درصورتی که نحوه‌ی زمانبندی این پردازه به صورت FIFO باشد و در بالاترین سطح اولویت باشد ، صف مربوط به پردازه‌ها با اولویت بیشینه را بررسی کنیم و (درصورت وجود) پردازنده با نحوه‌ی زمانبندی FIFO و متغیر dsn_rt_prio بالاتر را انتخاب می‌کنیم. روند توصیف شده در بالا به صورت زیر متناظر با پیاده‌سازی در کرنل لینوکس می‌شود :

```
1. __sched(void) [in core.c] calls
   pick_next_task(struct rq *rq) [in core.c]

2. pick_next_task(struct rq) calls
   pick_next_task_rt(struct rq*, struct task_struct*)
   [ in rt.c ]

3. pick_next_task_rt(struct rq*, struct task_struct*)
   [ in rt.c ] calls
   _pick_next_task_rt(struct rq*) [ in rt.c ]

4. _pick_next_task_rt(struct rq*) [ in rt.c ] calls
   pick_next_rt_entity(struct rq*, struct rt_rq*)
```

5. **pick_next_rt_entity** returns 1st highest priority task with these 3 function calls :

```
idx = sched_find_first_bit(array->bitmap);

queue = array->queue + idx;

next = list_entry(queue->next, struct
sched_rt_entity, run_list);
```

لازم است در اینجا نحوه‌ی زمانبندی next و همچنین مقدار idx را بررسی کنیم. قطعه کد آمده در زیر این شرط را بررسی می‌کند و درصورت برقراری آن را (درصورت وجود) با پردازه دیگری با dsn_rt_prio بالاتر جایگزین می‌کند :

```
p = rt_task_of(next);

if( (idx == 0) && (p->policy == SCHED_FIFO) )

{

      ptr = queue->next->next;

      max_dsn_rt_prio = p->dsn_rt_prio;

      while( ptr != queue->next ) /* Keep searching for tasks    with
      FIFO policy till we traverse whole list */

      {

            next_tmp = list_entry( ptr , struct sched_rt_entity,
            run_list);

            p = rt_task_of(next_tmp);

            if( (p->dsn_rt_prio > max_dsn_rt_prio) && (p->policy
            ==    SCHED_FIFO) )

            {

                  next = next_tmp;

                  max_dsn_rt_prio = p->dsn_rt_prio;

            }

            ptr = ptr->next; /* Advance pointer */

      }

}
```

ب) افزودن فراخوانی‌های سیستمی به کرنل و تهیه برنامه تست در سطح کاربر **(مسئول : محمدرضا بیاتپور)**

برای کار با زمانبندی درنظرگرفته شده برای پردازه‌ها با نحوه‌ی زمانبندی FIFO ، در صورت پروژه دو فراخوانی سیستمی

- set_dsn_rt_prio
- get_dsn_rt_prio

درنظر گرفته شده بود که دو این فراخوانی با اعمال تغییرات زیر در کد کرنل پیاده‌سازی شدند :

- adding syscall prototypes to **syscalls.h** (before #endif):
  - asmlinkage int sys_get_dsn_rt_prio (pid_t arg1)
  - asmlinkage void sys_set_dsn_rt_prio (pid_t arg1 , int arg2)

- adding syscall name and number to **syscall_32.tbl** (after all other syscalls) :
  - 351 i386 get_dsn_rt_prio sys_get_dsn_rt_prio
  - 352 i386 set_dsn_rt_prio sys_set_dsn_rt_prio

- adding actual implementation of syscalls to **kernel/sched**
  - dsn_rt_prio.c :

```
#include <linux/pid.h>
#include <linux/sched.h>

asmlinkage int sys_get_dsn_rt_prio ( pid_t arg1 ){
     struct task_struct *p;
     p = pid_task(find_vpid(arg1),PIDTYPPE_PID);
     return(p->dsn_rt_prio);
}

asmlinkage void sys_set_dsn_rt_prio ( pid_t arg1 ,
     int arg2 ){
     struct task_struct *p;
     p = pid_task(find_vpid(arg1),PIDTYPE_PID);
     p->dsn_rt_prio = arg2;
}
```

- modifying makefile of directory where .c file is placed (kernel/sched/Makefile)

- adding **dsn_rt_prio.o** to end of line **obj-y +=** …. (line #14) : this way syscall will properly located and linked

جهت اطمینان از عملکرد زمانبندی و فراخوانی‌های سیستمی جدید، برنامه‌ای به زبان c++ تهیه شد. این برنامه دو پردازه ایجاد کرده و نحوه‌ی زمانبندی آنها را به FIFO و اولویت آنها را به مقدار بیشینه تغییر می‌دهد به استثنای متغیر dsn_rt_prio که مقدار آن برای یکی بزرگتر از دیگری است. هردوی این پردازه یک برنامه به نام filewriter.o را اجرا می‌کنند که در داخل فایل مشخص شده اعداد ۱ تا ۱۰۰۰۰۰۰ را می نویسد. با توجه به اینکه در سیستم عامل لینوکس پردازه‌ها با زمانبندی FIFO فقط در یکی از سه حالت زیر اجرایشان متوقف می‌شود ، این برنامه انتخاب شد چراکه برای پردازه‌های درنظرگرفته شده در پروژه، پردازه با اولوبت بالاتر وجود ندارد و تنها راه مقایسه عملکرد آنها این است هردو مجبور به درخواست I/O کنیم. برنامه نوشته شده به c++ براساس اینکه کدام پردازه نخست به اتمام می‌رسد، عملکرد زمانبندی جدید را بررسی می‌کند.

[man sched_setscheduler] A SCHED_FIFO thread runs until either

1. it is blocked by an I/O request,
2. it is preempted by a higher priority thread, or
3. it calls sched_yield()

پیوست :

در پیوست برنامه c++ نوشته شده برای تست زمانبندی جدید همچنین patch مربوط به تغییرات کرنل نسبت به نسخه‌ی اصلی آمده است.