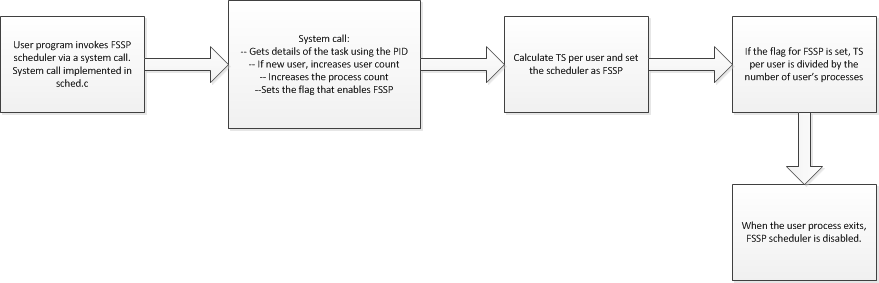
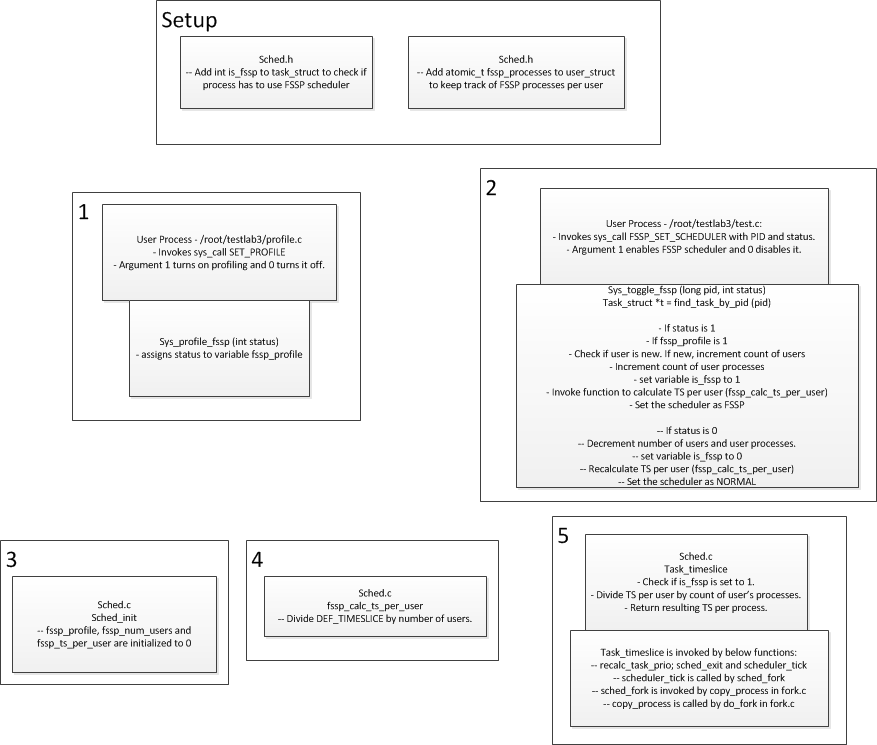
**Implementation:**

* We call our scheduler - FSSP - short for Fair Share Scheduling Policy.
* The majority of our changes were done on the file sched.c in functions - task\_timeslice() and sched\_init(). The system calls were implemented inside sched.c as we were getting compilation errors when we had it as a separate file.
* We added additional fields to the data structures task\_struct (include/linux/sched.h) - called “int is\_fssp” which decides if the process has to be scheduled using FSSP or not. A field was also added to user\_struct (include/linux/sched.h) - called atomic\_t fssp\_processes - which keeps track of the processes per user.
* The below flow chart shows our implementation details. The process is initiated when the user program invokes the system call that enables the scheduler.
* The system call checks if the user is new or not based on the count of their processes. If it is 0, the user count is incremented. The process count for that user is also incremented (atomic\_inc(&t->user->fssp\_processes)).
* The TS per user is calculated by dividing the DEF\_TIMESLICE by the count of users.
* The flag that enables the process to be scheduled using the TS calculated by FSSP is set.
* The process is now scheduled for its new TS.
* When the user process exits, it disables FSSP. The system call decreases the process count and the user count (if no more processes for that user) and recalculates the TS. The scheduler is set to the default one.



**Architecture Diagram:**

The diagram below shows in much more details what was explained in the previous section.



**Results:**

The test scenario used was with users with IDs of 504, 505, 506 and 507, corresponding to users A, B, C and D.

As can be observed, the CPU time for each user came out to be uniform. The following is the output from our post-processing script, written in Python. The output is an ordered pair with user ID and proportion of total CPU time assigned to their processes. As can be seen, each user received roughly 25% of the CPU time, which should be the case given that there are 4 users.

('506', 0.25252525252525254)

('507', 0.24915824915824916)

('504', 0.2474747474747475)

('505', 0.25084175084175087)

Also, the total times per process is equal when given a user. This can be seen by the following script output. Each segment of output is for a given UID. There, each PID and its corresponding total time executing on the CPU is written as well. As can be seen by observing the data, this value is constant among processes from the same user.

('UID: ', '506')

('PID: ', '1234')

('TS:', 400)

('PID: ', '1235')

('TS:', 400)

('PID: ', '1236')

('TS:', 400)

('UID: ', '507')

('PID: ', '1271')

('TS:', 392)

('PID: ', '1272')

('TS:', 392)

('PID: ', '1273')

('TS:', 400)

('UID: ', '504')

('PID: ', '1160')

('TS:', 392)

('PID: ', '1161')

('TS:', 392)

('PID: ', '1162')

('TS:', 392)

('UID: ', '505')

('PID: ', '1199')

('TS:', 392)

('PID: ', '1198')

('TS:', 400)

('PID: ', '1197')

('TS:', 400)

**UID Vs TimeSlice:**

|  |  |
| --- | --- |
| UID | TS |
| 504 | 1200 |
| 505 | 1184 |
| 506 | 1176 |
| 507 | 1192 |

**PID Vs TimeSlice:**

|  |  |  |
| --- | --- | --- |
| UID | PID | TS |
| 504 | 1160 | 392 |
| 1161 | 392 |
| 1162 | 392 |
|  |  |  |
| 505 | 1197 | 400 |
| 1198 | 400 |
| 1199 | 392 |
|  |  |  |
| 506 | 1234 | 400 |
| 1235 | 400 |
| 1236 | 400 |
|  |  |  |
| 507 | 1271 | 392 |
| 1272 | 392 |
| 1273 | 400 |

Graph below shows the TimeSlice split uniformly between users.

Graph below shows the TimeSlice split evenly between processes per user.

**Actual changes:**

* include/linux/sched.h
  + Define new scheduling policy (~line 129)
  + Add fssp\_processes counter to user\_struct
  + Add is\_fssp boolean to task\_struct
* kernel/sched.c
  + Remove sys\_sched\_setscheduler syscall (disable existing system call)
  + Remove sys\_sched\_setparam
  + *When enabling fssp policy for a process*
    - *If procesess under fssp for this user = 0, increase num\_users*
    - *Number of processes to be incremented by 1*
    - *Invoke fssp\_calc\_ts\_per\_user()*
  + *When disabling fssp policy for a process*
    - *Number of processes to be decremented 1*
    - *If procesess under fssp for this user = 0, decrease num\_users*
    - *Invoke fssp\_calc\_ts\_per\_user()*
* kernel/user.c
  + In alloc\_uid()
    - set fssp\_processes to 0
* fssp\_calc\_ts\_per\_user()
  + Divide DEF\_TIMESLICE / fssp\_num\_users.
* Toggle system call
  + sys\_toggle\_fssp() implemented in kernel/sched.c
  + arch/i386/kernel/entry.S - system call table entry added (285)
  + include/asm-i386/unistd.h - system call number macro added
* Profile system call
  + sys\_profile\_fssp() in kernel/sched.c
  + arch/i386/kernel/entry.S - system call table entry added (286)
  + include/asm-i386/unistd.h - system call number macro added
* Configuration
  + Using 1mb\_config
* In task\_timeslice() in sched.c:
  + Calculate user’s fssp processes number / timeslice per use