BAHIR DAR UNIVERSITY

Bahir Dar Institute of Technology (BiT)

Faculty of Computing

Department of Software Engineering

Operating System and System Programming

Individual Assignment (¡¡)

System Call

sched_setscheduler()

Documented by:-

Name: Smachew Gedefaw

Id Number: BDU1308736

Department: Software Engineering

Submitted to: Instructor Wondmu Baye

Submit date: 15/11/2014 E.C

Dear my instructor, I am really sorry for my lateness, and I want to tell you that the real reason beyond my lateness is not being lazy or ignoring the given work. I am late for the only reason which is because I have no my own computer to run the implementation, so I was obliged to wait until my friends finished their work. And I want to tell you that I was trying really my best to handle it by my own in the digital library, but most of those computers of the digital library have not any type of Operating system other than windows. And the online terminals have greater problems of supporting c libraries (libc), so with these reasons I would like to say very sorry for my lateness.

Introduction

In this short representation of concepts some points are addressed briefly and clearly as much as possible. What will present is restricted under some topics.

The concept of system call is touched through a specific type of process operation called process scheduling. Process scheduling is the main concern of this work, and it will briefly explain, exemplify and implement this concept in clear way.

1. What is sched_setscheduler(S)?

Scheduling Policies

The scheduler is the kernel part that decides which runnable process will be executed by the CPU next. Each process has an associated scheduling policy and a *static* scheduling priority, ***sched_priority***; these are the settings that are modified by ***sched_setscheduler()***. The scheduler makes it decisions based on knowledge of the scheduling policy and static priority of all processes on the system.

The Linux scheduler offers three different scheduling policies, one for *normal processes* and two for *real-time applications*. A static priority value ***sched_priority*** is assigned to each process and this value can be changed only via system calls.

- For processes scheduled under one of the normal scheduling policies (**SCHED_OTHER, SCHED_IDLE**, **SCHED_BATCH**), ***sched_priority*** is not used in scheduling decisions (it must be specified as 0).
- For processes scheduled under one of the real-time policies (**SCHED_FIFO**, **SCHED_RR**) have a ***sched_priority*** value in the range 1 (low) to 99 (high). (As the numbers imply, real-time processes always have higher priority than normal processes.)

Note! POSIX.1-2001 only requires an implementation to support a minimum 32 distinct priority levels for the real-time policies, and some systems supply just this minimum. Portable programs should use **sched_get_priority_min**(2) and **sched_get_priority_max**(2) to find the range of priorities supported for a particular policy.

Conceptually, the scheduler maintains a list of runnable processes for each possible *sched_priority* value. In order to determine which process runs next, the scheduler looks for the nonempty list with the highest static priority and selects the process at the head of this list.

A process's scheduling policy determines where it will be inserted into the list of processes with equal static priority and how it will move inside this list.

All scheduling is preemptive: if a process with a higher static priority becomes ready to run, the currently running process will be preempted and returned to the wait list for its static priority level. The scheduling policy only determines the ordering within the list of runnable processes with equal static priority.

sched_setscheduler()

What is sched_setscheduler?

sched_setscheduler is a function for system call which is defined in header file called ***sched.h***. The **sched_setscheduler**() system call sets both the scheduling policy and parameters for the thread whose ID is specified in declared variable ***pid***. (See more on: how does it work?) it is system calls that modify and retrieve scheduling policies and priorities.

Synopsis

```
[Option Start] #include <sched.h>
               int sched_setscheduler(pid_t pid, int policy, const struct
 sched_param *param);
               int sched_getscheduler(pid_t pid);
               ...
               int sched_priority;
               ...
               };  [Option End]
```

2. Why this system call is called?

What does it work?

The sched_setscheduler does set scheduling policy and parameters (REALTIME). This function sets both the absolute priority and the scheduling policy for a task. It assigns the absolute priority value given by param and the scheduling policy policy to the task with ID pid, or the calling task if pid is zero. If policy is negative, sched_setscheduler keeps the existing scheduling policy.

3. How does it work?

Procedural description

As previously expressed, The **sched_setscheduler**() system call sets both the scheduling policy and parameters for the thread whose ID is specified in *pid*. If *pid* equals zero, the scheduling policy and parameters of the calling thread will be set(the attributes of the calling process are changed).

The scheduling parameters are specified in the *param* argument, which is a pointer to a structure of the following form:-

```
struct sched_param {
     ...
     int sched_priority;
     ...
 };
```

In the current implementation, the structure contains only one field, *sched_priority*. The interpretation of *param* depends on the selected policy. Currently, Linux supports the following "normal" (i.e., non-real- time) scheduling policies as values that may be specified in*policy*:-

SCHED_OTHER
The standard round-robin time-sharing policy;

SCHED_BATCH
For "batch" style execution of processes;

SCHED_IDLE
For running *very* low priority background jobs.

For each of the above policies, *param->sched_priority* must be 0. Various "real-time" policies are also supported, for special time-critical applications that need precise control over the way in which runnable threads are selected for execution. For the rules governing when a process may use these policies[1]. The real-time policies that may be specified in *policy* are:

SCHED_FIFO
A first-in, first-out policy; and

SCHED_RR
A round-robin policy.

For each of the above policies, *param->sched_priority* specifies a scheduling priority for the thread. This is a number in the range returned by calling sched_get_priority_min(2) and sched_get_priority_max(2) with the specified *policy*. On Linux, these system calls return, respectively, 1 and 99. Since Linux 2.6.32, the **SCHED_RESET_ON_FORK** flag can be ORed in *policy* when calling **sched_setscheduler**(). As a result of including this flag, children created by fork(2) do not inherit privileged scheduling policies. **sched_getscheduler**()returns the current scheduling policy of the thread identified by *pid*. If *pid* equals zero, the policy of the calling thread will be retrieved.

Return value

On successful [No Error]:-

- ✓ **sched_setscheduler**() returns zero.
- ✓ **sched_getscheduler**() returns the policy for the thread (a nonnegative integer).
- ✓ Upon successful completion, the function shall return the former scheduling policy of the specified process.

On unsuccessful [On error]:-

- ✓ both calls return -1, and
- ✓ *errno* is set to indicate the error.
- ✓ If the sched_setscheduler() function fails to complete successfully, the policy and scheduling parameters shall remain unchanged, and the function shall return a value of -1 and set errno to indicate the error.

```
#include <sched.h>
int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param);
```
Returns **0** on success, or **–1** on error.

[1] For more see :- ***sched(7) — Linux manual page (https://man7.org/linux/man-pages/man7/sched.7.html)***

Errors

The sched_setscheduler() function shall fail if:-

EINVAL

The value of the policy parameter is invalid, or one or more of the parameters contained in param is outside the valid range for the specified scheduling policy.

- ✓ Invalid arguments: pid is negative or param is NULL.
- ✓ *(sched_setscheduler())*policy is not one of the recognized policies.
- ✓ **(sched_setscheduler())** param does not make sense for the specified policy

EPERM

The requesting process does not have permission to set either or both of the scheduling parameters or the scheduling policy of the specified process.

- ✓ The calling thread does not have appropriate privileges.

ESRCH

No process can be found corresponding to that specified by ***pid***.

- ✓ The thread whose ID is pid could not be found.

4. List of parameters

1. Param (*param)

The param argument is a pointer to a structure of the following form:-

```
struct sched_param {
int sched_priority;      // Scheduling priority
};
```

SUSv3[2] defines the ***param*** argument as a structure to ***allow an implementation*** to include additional implementation-specific fields, which may be useful if an implementation provides additional scheduling policies.

However, like most UNIX implementations, Linux provides just the ***sched_priority*** field, which specifies the scheduling priority. For the **SCHED_RR** and **SCHED_FIFO** policies, this must be a value in The range indicated by sched_get_priority_min() and sched_get_priority_max(); for other policies, the priority must be **0**.

2. Policy

The ***policy*** argument determines the ***scheduling policy*** for the process. It is specified as one of the policies shown in below:-

[2] **Single UNIX Specification** is a standard for computer operating systems, compliance with which is required to qualify for using the "UNIX" trademark. For more:- https://en.wikipedia.org/wiki/Single_UNIX_Specification

Policy	Description
SCHED_FIFO	**Real-time first-in first-out**
SCHED_RR	**Real-time round-robin**
SCHED_OTHER SCHED_BATCH SCHED_IDLE	**Standard round-robin time-sharing Similar to SCHED_OTHER, but intended for batch execution (since Linux 2.6.16) Similar to SCHED_OTHER, but with priority even lower than nice value +19 (since Linux 2.6.23)**

Table 1: Linux real-time and non-real-time scheduling policies

3. pid

A successful sched_setscheduler() call moves the process specified by ***pid*** to the back of the queue for its priority level. SUSv3 specifies that the return value of a successful sched_setscheduler() call should be the previous scheduling policy. However, Linux deviates from the standard in that a successful call returns 0. A portable application should test for success by checking that the return status is not –1.

The scheduling policy and priority are inherited by a child created via fork(), and they are preserved across an exec(). The sched_setparam() system call provides a subset of the functionality of sched_setscheduler(). It modifies the scheduling priority of a process while leaving the policy unchanged.

```
#include <sched.h>
int sched_setparam(pid_t pid, const struct sched_param *param);
```

Returns **0** on success, or **–1** on error

Generally:-

pid – PID of process

policy – policy flag

param – pointer to sched_param structure

The *pid* and param arguments are the same as for sched_setscheduler(). A successful sched_setparam() call moves the process specified by ***pid*** to the back of the queue for its priority level. The program in in the implementation uses sched_setscheduler() to set the policy and priority of the processes specified by its command-line arguments. The first argument is a letter specifying a scheduling policy, the second is an integer priority, and the remaining arguments are the process IDs of the processes whose scheduling attributes are to be changed.

5. List of Flags and their purpose

This system call has three parameters as previously explained and many policies for the implementation of the process scheduling. The sched_setscheduler() function will cooperate with sched_getscheduler() function in order to facilitate the code based implementation. In short, in this of system call, the are no more bit based components called ***flags.***

And therefore, there are no flags are available to present in this work, only 3 parameters and around 5 different policies under 2 categories, are present and hold the whole job of the process scheduling through system call.

6. Implementation for sched_setscheduler

What happens During Implementation of sched_setscheduler() system call?

Implementations may require that the requesting process have permission to set its own scheduling parameters or those of another process. Additionally, implementation-defined restrictions may apply as to the appropriate privileges required to set a process' own scheduling policy, or another process' scheduling policy, to a particular value.

During implementation of the scheduling codes with sched_setscheduler(), the following points are noticeable:-

* The sched_setscheduler() function shall :-
  - Set the scheduling policy (either normal real-time policies) and
  - Scheduling parameters of the process specified by ***pid*** to policy and the parameters specified in the ***sched_param*** structure pointed to by ***param***, respectively.
* The value of the ***sched_priority*** member in the ***sched_param*** structure shall be any integer within the inclusive priority range for the scheduling policy specified by policy.
* If the value of ***pid*** is negative, the behavior of the sched_setscheduler() function is unspecified.
* The possible values for the policy parameter are defined in the <sched.h> header.
* If a process specified by pid exists, and if the calling process has permission, the scheduling policy and scheduling parameters shall be set for the process whose process ID is equal to pid.
* If pid is zero, the scheduling policy and scheduling parameters shall be set for the calling process.
* The conditions under which one process has the appropriate privilege to change the scheduling parameters of another process are implementation-defined.
* This function is not atomic with respect to other threads in the process. Threads may continue to execute while this function call is in the process of changing the scheduling policy and associated scheduling parameters for the underlying kernel-scheduled entities used by the process contention scope threads.

Implementation Examples:-

Obviously, most operating systems use C programming language for their implementation part, Linux's scheduling implementations can be written with C language, also possible with other programming languages.

Implementation 1: Modifying process scheduling policies and priorities

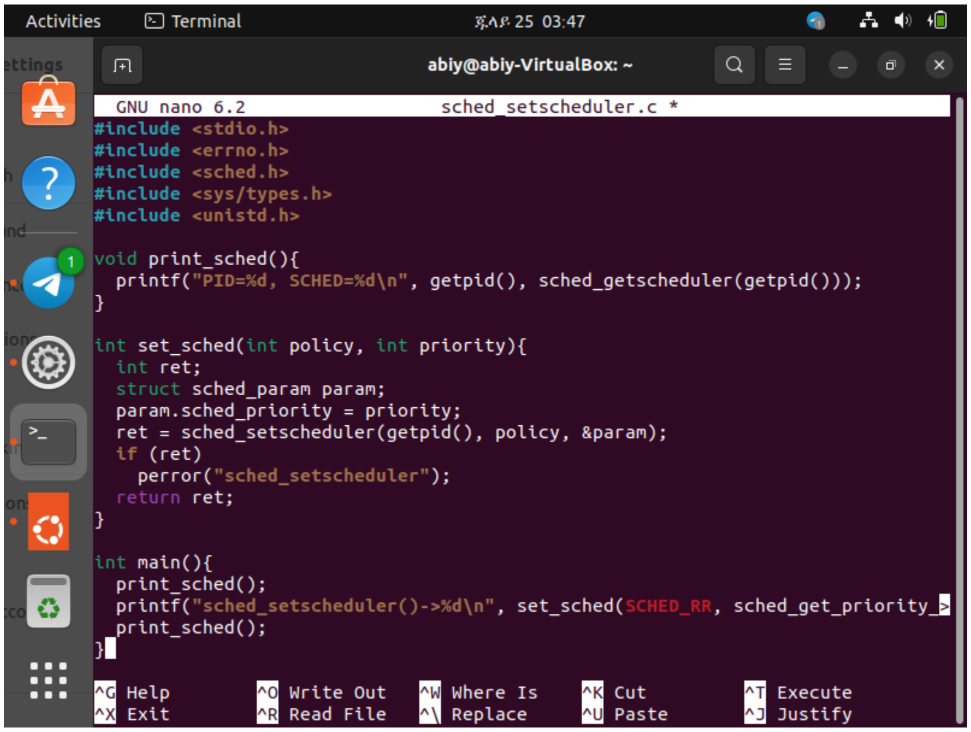
```
#include <sched.h>

#include "tlpi_hdr.h"

int main(int argc, char *argv[])

{

   int j, pol;

   struct sched_param sp;
```

```
 if (argc < 3 || strchr("rfo", argv[1][0]) == NULL)
 usageErr("%s policy priority [pid...]\n"
   " policy is 'r' (RR), 'f' (FIFO), "
#ifdef SCHED_BATCH            /* Linux-specific */
   "'b' (BATCH), "
#endif
#ifdef SCHED_IDLE             /* Linux-specific */
   "'i' (IDLE), "
#endif
   "or 'o' (OTHER)\n",
argv[0]);
pol = (argv[1][0] == 'r') ? SCHED_RR :
   (argv[1][0] == 'f') ? SCHED_FIFO :
#ifdef SCHED_BATCH
   (argv[1][0] == 'b') ? SCHED_BATCH :
#endif
#ifdef SCHED_IDLE
   (argv[1][0] == 'i') ? SCHED_IDLE :
#endif
   SCHED_OTHER;
   sp.sched_priority = getInt(argv[2], 0, "priority");
   for (j = 3; j < argc; j++)
    if (sched_setscheduler(getLong(argv[j], 0, "pid"), pol, &sp) == -1)
     errExit("sched_setscheduler");
     exit(EXIT_SUCCESS);
}
```

Implementation 2:-

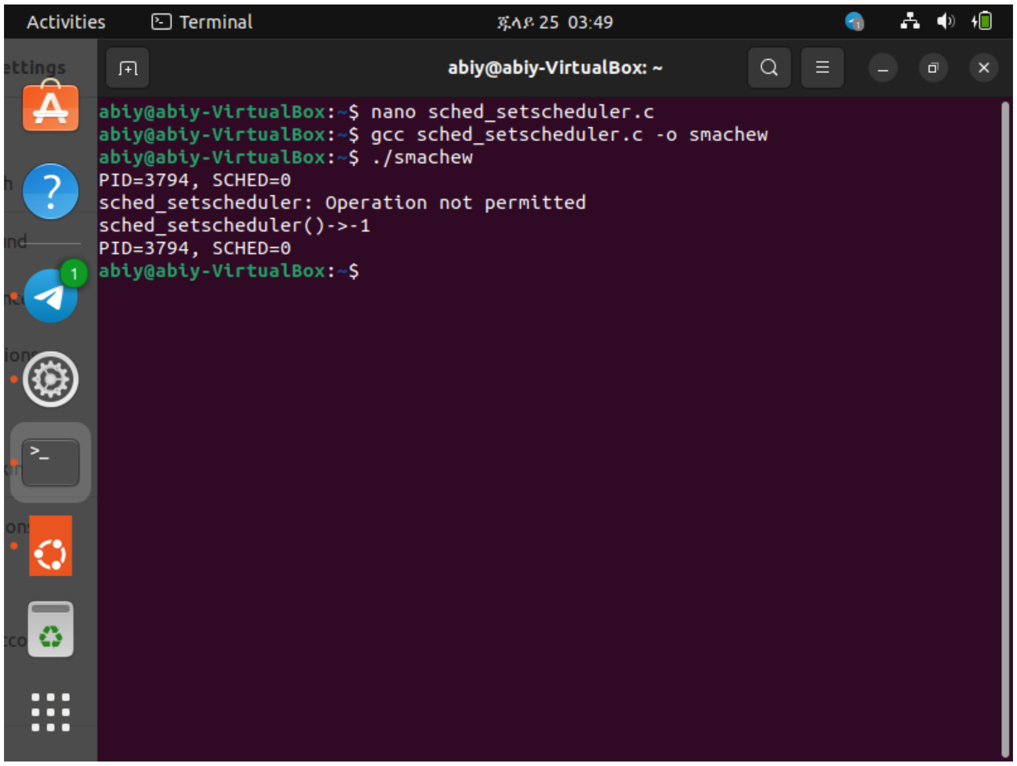
```
#include <stdio.h>
```

```c
#include <errno.h>

#include <sched.h>

#include <sys/types.h>

#include <unistd.h>


void print_sched(){

  printf("PID=%d, SCHED=%d\n", getpid(), sched_getscheduler(getpid()));

}


int set_sched(int policy, int priority){

  int ret;

  struct sched_param param;

  param.sched_priority = priority;

  ret = sched_setscheduler(getpid(), policy, &param);

  if (ret)

    perror("sched_setscheduler");

  return ret;

}


int main(){

  print_sched();

  printf("sched_setscheduler()->%d\n", set_sched(SCHED_RR,
sched_get_priority_max(SCHED_RR)));

  print_sched();

}
```



```
 GNU nano 6.2                    sched_setscheduler.c *
#include <stdio.h>
#include <errno.h>
#include <sched.h>
#include <sys/types.h>
#include <unistd.h>

void print_sched(){
  printf("PID=%d, SCHED=%d\n", getpid(), sched_getscheduler(getpid()));
}

int set_sched(int policy, int priority){
  int ret;
  struct sched_param param;
  param.sched_priority = priority;
  ret = sched_setscheduler(getpid(), policy, &param);
  if (ret)
    perror("sched_setscheduler");
  return ret;
}

int main(){
  print_sched();
  printf("sched_setscheduler()->%d\n", set_sched(SCHED_RR, sched_get_priority_>
  print_sched();
}

^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify
```

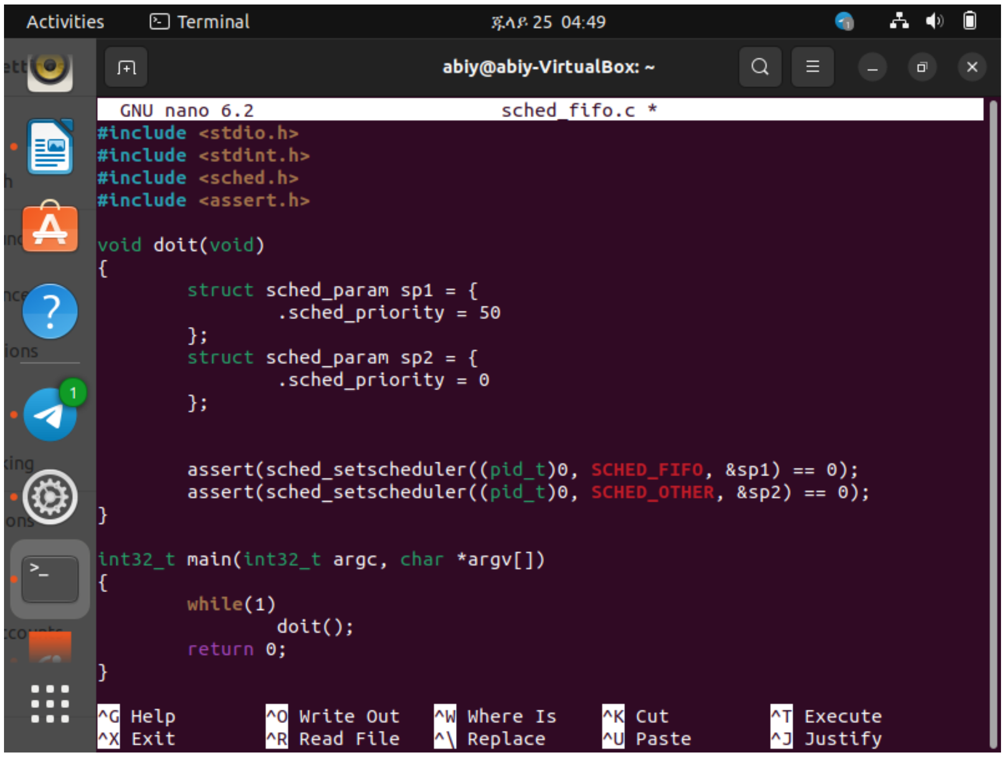
OUTPUT:



```
abiy@abiy-VirtualBox:~$ nano sched_setscheduler.c
abiy@abiy-VirtualBox:~$ gcc sched_setscheduler.c -o smachew
abiy@abiy-VirtualBox:~$ ./smachew
PID=3794, SCHED=0
sched_setscheduler: Operation not permitted
sched_setscheduler()->-1
PID=3794, SCHED=0
abiy@abiy-VirtualBox:~$
```

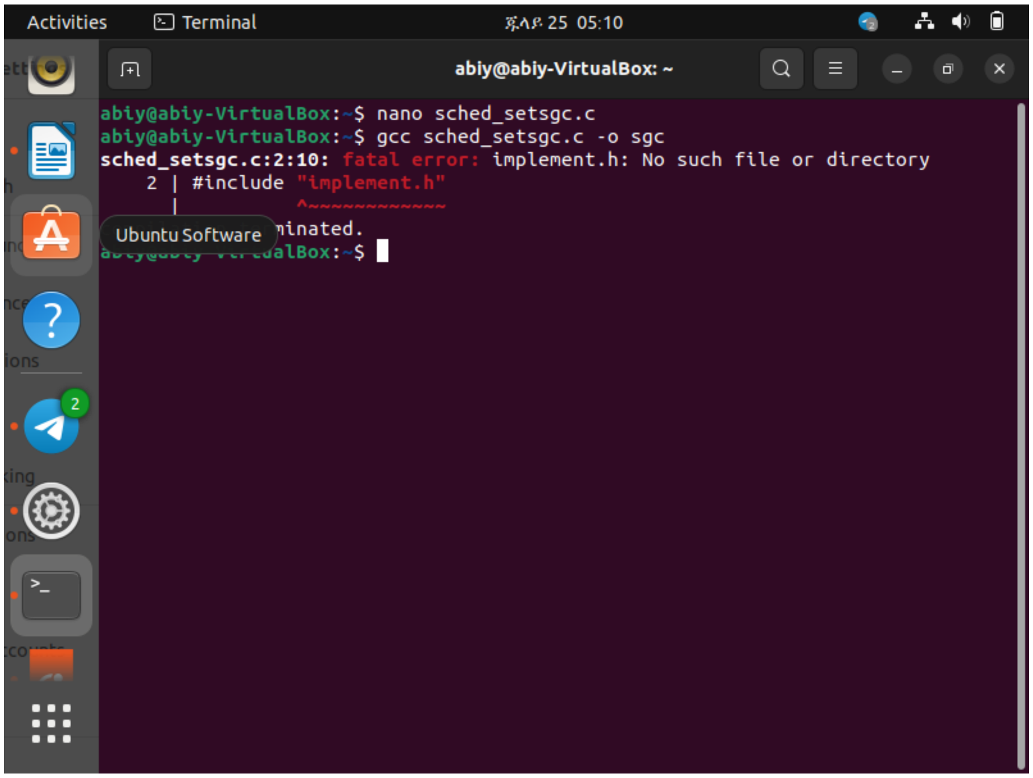
Implementation 3:-

```
#include <stdio.h>

#include <stdint.h>

#include <sched.h>

#include <assert.h>

void doit(void)

{

	struct sched_param sp1 = {

		.sched_priority = 50

	};

	struct sched_param sp2 = {

		.sched_priority = 0

	};

	assert(sched_setscheduler((pid_t)0, SCHED_FIFO, &sp1) == 0);

	assert(sched_setscheduler((pid_t)0, SCHED_OTHER, &sp2) == 0);

}

int32_t main(int32_t argc, char *argv[])

{

	while(1)

		doit();

	return 0;

}
```



```
  GNU nano 6.2                          sched_fifo.c *
#include <stdio.h>
#include <stdint.h>
#include <sched.h>
#include <assert.h>

void doit(void)
{
        struct sched_param sp1 = {
                .sched_priority = 50
        };
        struct sched_param sp2 = {
                .sched_priority = 0
        };


        assert(sched_setscheduler((pid_t)0, SCHED_FIFO, &sp1) == 0);
        assert(sched_setscheduler((pid_t)0, SCHED_OTHER, &sp2) == 0);
}

int32_t main(int32_t argc, char *argv[])
{
        while(1)
                doit();
        return 0;
}
^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify
```

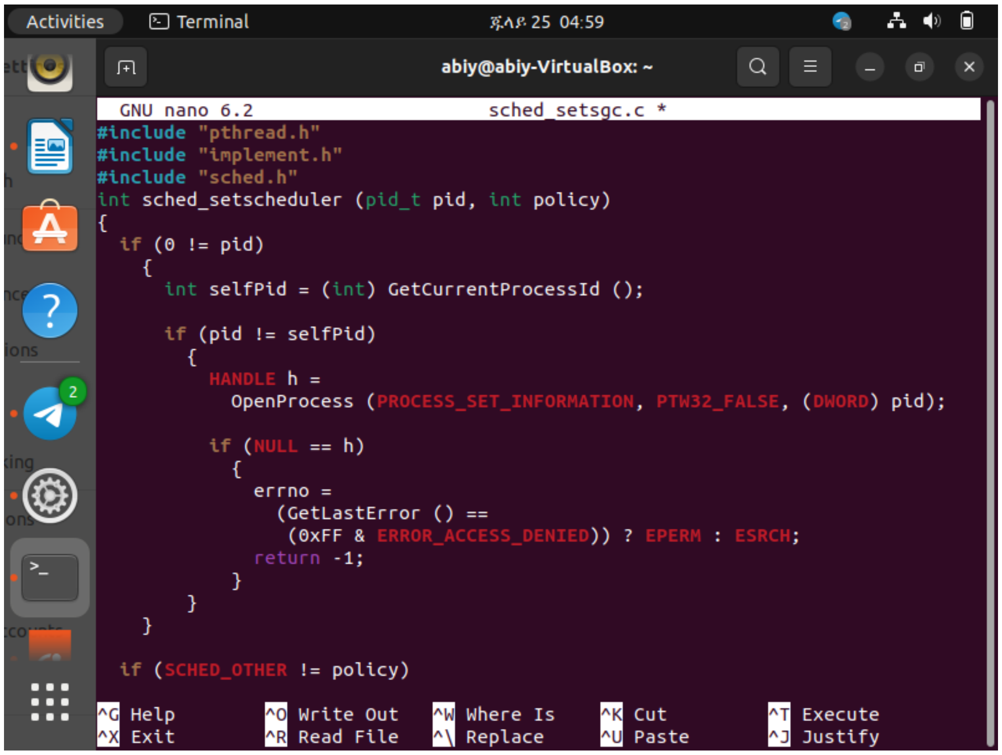
OUTPUT:



```
abiy@abiy-VirtualBox:~$ nano sched_setsgc.c
abiy@abiy-VirtualBox:~$ gcc sched_setsgc.c -o sgc
sched_setsgc.c:2:10: fatal error: implement.h: No such file or directory
    2 | #include "implement.h"
      |          ^~~~~~~~~~~~~
                     minated.
abiy@abiy-VirtualBox:~$ 
```

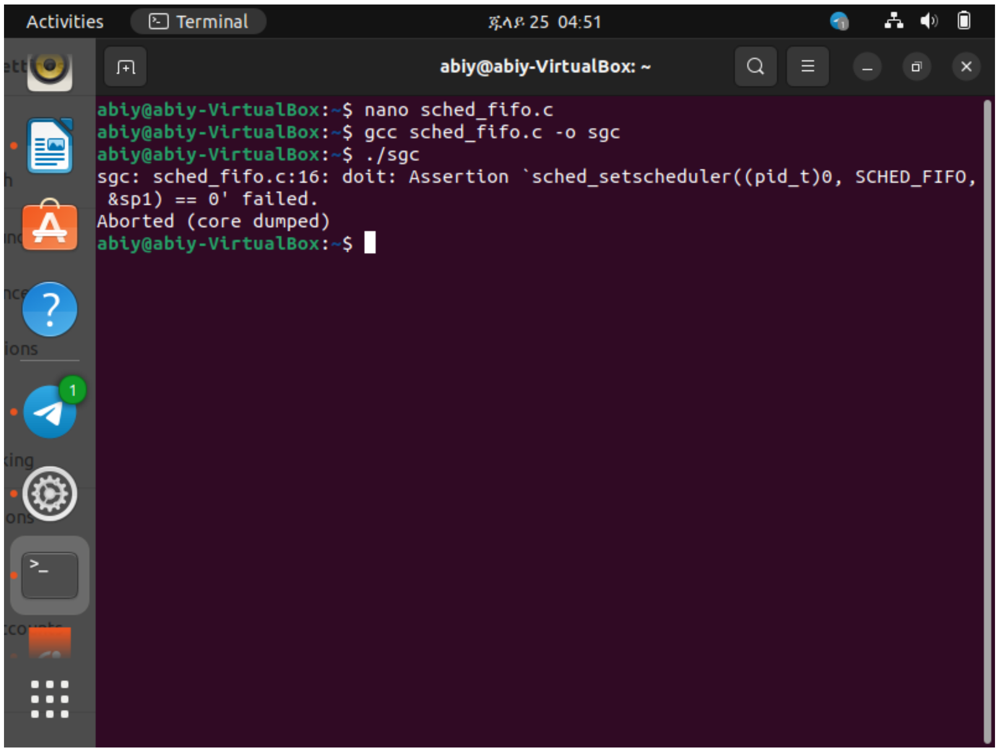
Implementation 4:-

```
#include "pthread.h"

#include "implement.h"

#include "sched.h"


int

sched_setscheduler (pid_t pid, int policy)

{

  /*

   * Win32 only has one policy which we call SCHED_OTHER.

   * However, we try to provide other valid side-effects

   * such as EPERM and ESRCH errors. Choosing to check

   * for a valid policy last allows us to get the most value out

   * of this function.

   */

  if (0 != pid)

    {

      int selfPid = (int) GetCurrentProcessId ();


      if (pid != selfPid)

      {

        HANDLE h =

   OpenProcess (PROCESS_SET_INFORMATION, PTW32_FALSE, (DWORD) pid);

        if (NULL == h)

          {

            errno =

            (GetLastError () ==

             (0xFF & ERROR_ACCESS_DENIED)) ? EPERM : ESRCH;
          return -1;
        }
    }
  }

if (SCHED_OTHER != policy)
  {
    errno = ENOSYS;
    return -1;
  }

/*
 * Don't set anything because there is nothing to set.
 * Just return the current (the only possible) value.
 */
return SCHED_OTHER;
}
```



```
  GNU nano 6.2                          sched_setsgc.c *
#include "pthread.h"
#include "implement.h"
#include "sched.h"
int sched_setscheduler (pid_t pid, int policy)
{
  if (0 != pid)
    {
      int selfPid = (int) GetCurrentProcessId ();

      if (pid != selfPid)
        {
          HANDLE h =
            OpenProcess (PROCESS_SET_INFORMATION, PTW32_FALSE, (DWORD) pid);

          if (NULL == h)
            {
              errno =
                (GetLastError () ==
                 (0xFF & ERROR_ACCESS_DENIED)) ? EPERM : ESRCH;
              return -1;
            }
        }
    }

  if (SCHED_OTHER != policy)
^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify
```

OUTPUT:



```
abiy@abiy-VirtualBox:~$ nano sched_fifo.c
abiy@abiy-VirtualBox:~$ gcc sched_fifo.c -o sgc
abiy@abiy-VirtualBox:~$ ./sgc
sgc: sched_fifo.c:16: doit: Assertion `sched_setscheduler((pid_t)0, SCHED_FIFO,
 &sp1) == 0' failed.
Aborted (core dumped)
abiy@abiy-VirtualBox:~$ 
```

7. Key words

Synopses:-

It is the blueprint of the project work or a brief **summary** that gives audiences an idea of what a composition is about.

Flag :-

It is one or more data bits used to store binary values as specific program structure indicators. A flag is a component of a programming language's data structure. A computer interprets a flag value in relative terms or based on the data structure presented during processing, and uses the flag to mark a specific data structure. Thus, the flag value directly impacts the processing outcome.

Sched_setscheduler:-

Set the scheduling policy and parameters of a specified thread.

References

- Basic scheduling functions, https://www.gnu.org/software/libc/manual/html_node/Basic-Scheduling-Functions.html
- sched_setscheduler(2) , Linux manual page, https://man7.org/linux/man-pages/man2/sched_setscheduler.2.html
- Sched_setscheduler, die.net, https://linux.die.net/man/2/sched_setscheduler
- The Linux Programming interface, Michael KerrisK, , A Linux and UNIX® System Programming Handbook.
- sched_setscheduler() - Unix, Linux System Call, javapoint tutorials, https://www.google.com/url?sa=t&source=web&rct=j&url=https://www.tutorialspoint.com/unix_system_calls/sched_setscheduler.htm