THE UNIVERSITY of EDINBURGH
informatics

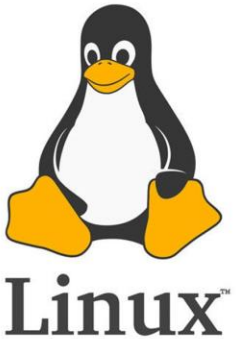
Coursework 1

Operating Systems
Spring 25

Overview

Introduction

- In this coursework you shall explore the data structures and mechanisms that the Linux kernel uses to manage processes, threads, and scheduling decisions.



Important Dates

- Coursework will be released on 11 Feb
- Coursework due on 25 Feb 2025 12:00

Scoring

- This coursework contributes to 17% of your overall grade.
- There are 3 tasks, scored independently.

Component	Weight
Task 1	20
Task 2	30
Task 3	50
Maximum Score	100

Instructions

Base Kernel

- We provide you a base kernel.
- Download it from here.
- You must work on this kernel.
- This kernel contains incomplete function definitions which you must complete.
- The following CONFIG options must be set
  - CONFIG_SMP
  - CONFIG_SCHED_STATS
  - CONFIG_SCHED_DEBUG

Submission

- You must restrict all your work to a single file: **kernel/sched/core.c**
- Submit your core.c in a zip archive named

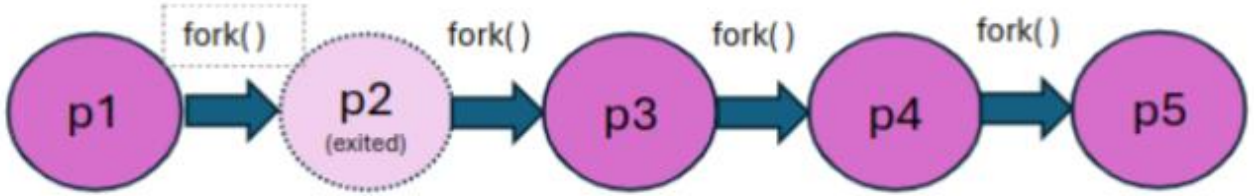
  CW1_<your student UUN>.zip

- Please DO NOT MODIFY ANY OTHER FILE.

Specifications

Task 1 – What's My Ancestry?

- Implement a New Syscall (ID: 463)
- Syscall(463, pid, n)
  - Pid: Target Process
  - N: Max Height
- Should Return the pid of the nth ancestor
- Find Incomplete definition (ancestor_pid) in core.c



$$\texttt{sys\_ancestor\_pid}(p5,\ 0) = p5.$$
$$\texttt{sys\_ancestor\_pid}(p5,\ 1) = p4.$$
$$\texttt{sys\_ancestor\_pid}(p5,\ 2) = p3.$$
$$\texttt{sys\_ancestor\_pid}(p4,\ 2) = \texttt{-ESRCH}.$$
$$\texttt{sys\_ancestor\_pid}(p4,\ 3) = \texttt{-ESRCH}.$$

Task 1 – Testing

NOTE: if pid argument is 0,
Use PID of the caller process

```c
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <sys/wait.h>

#define SYSCALL1 463  // Use the assigned syscall number

int main() {
  pid_t parent = getpid(), gparent = getppid();
  pid_t argpid = 0;

  pid_t pid = fork();  // Fork a new process

  if (pid == -1) {
    perror("fork");
    return 1;
  }

  if (pid == 0) {
    // Call the syscall from the child process
    for (unsigned int n = 0; n < 10; ++n)
        printf("CHILD: %d->%ld\n", n, syscall(SYSCALL1, argpid, n));
    return 0;
  } else {
    // Parent process
    wait(NULL);  // Wait for the child to finish
    printf("PARENT: chld:%d me(par):%d gp:%d\n", pid, parent, gparent);
  }

  return 0;
}
```

CHILD: 0->1100
CHILD: 1->1099
CHILD: 2->977
CHILD: 3->822
CHILD: 4->1
CHILD: 5->0
CHILD: 6->-1
CHILD: 7->-1
CHILD: 8->-1
CHILD: 9->-1
PARENT: chld:1100 me(par):1099 gp:977

Task 1 – Where to start?

- Find the incomplete syscall inside core.c
- COMPLETE IT!

- Check out:
  - struct task_struct in *include/linux/sched.h*
  - Find pointer to parent task inside task_struct

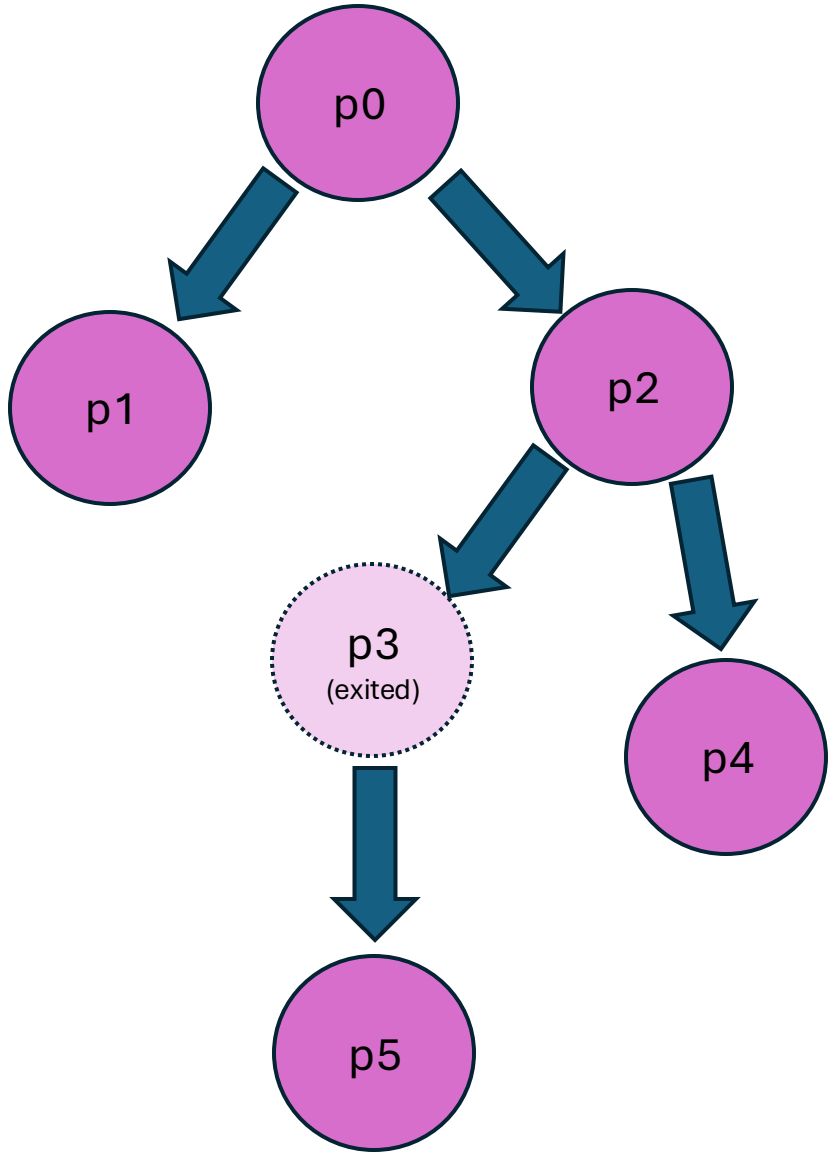
```
8135    /**
8136     * CW1
8137     * sys_ancestor_pid - get PID of the nth ancestor of process
8138     * @pid: pid of the process
8139     * @n: nth ancestor
8140     *
8141     * Return: PID on success. Error otherwise.
8142     */
8143    SYSCALL_DEFINE2(ancestor_pid, pid_t, pid, unsigned int, n)
8144    {
8145      return 0;
8146    }
8147
```

Task 2 – TrickleDown Niceness

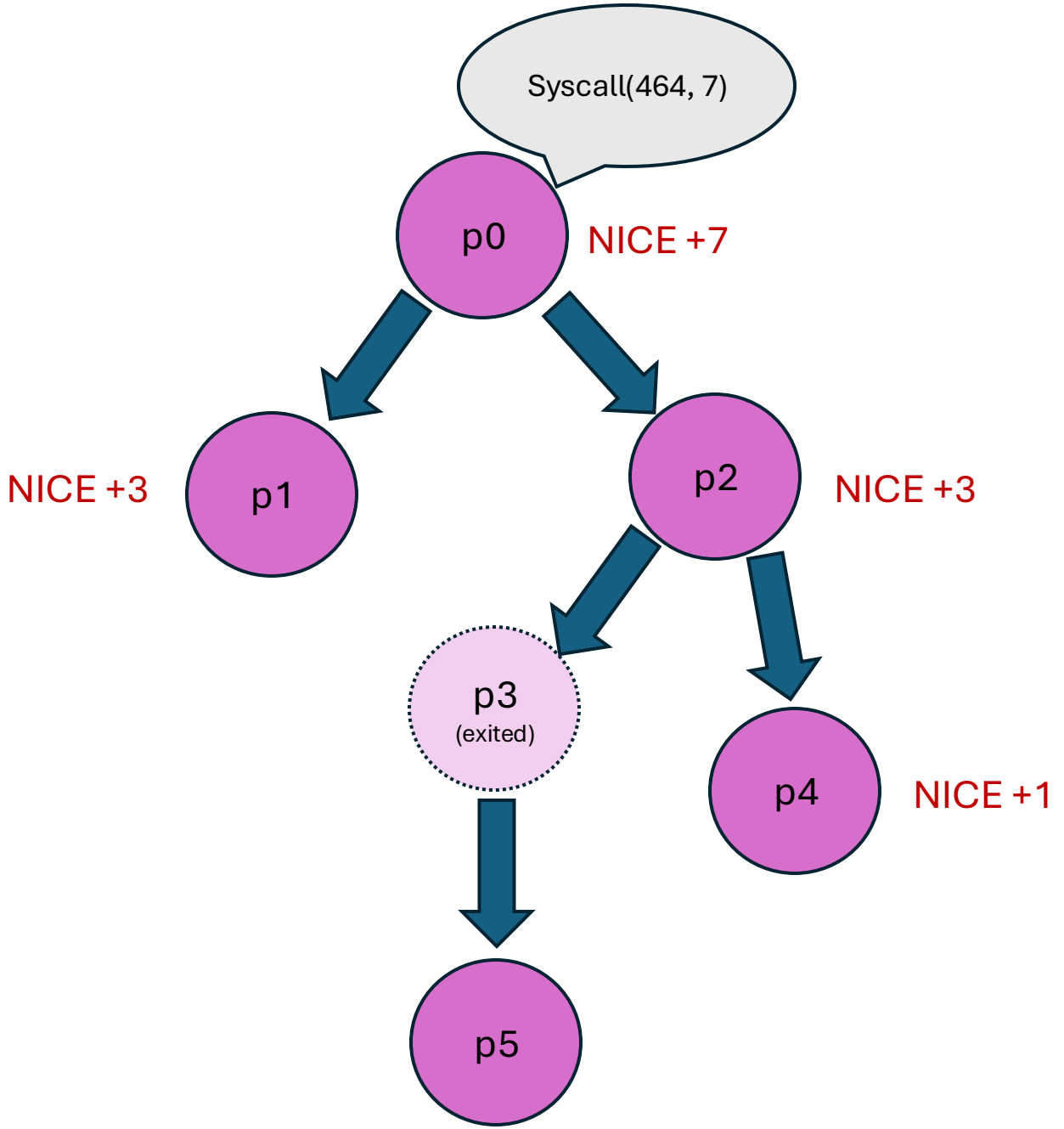
- Implement a syscall ID 464
- Syscall(464, n)
  - Increment caller's nice value by n
  - Increment its children's niceness by n/2
  - Increment its grand-children by n/4
  - Increment its great-grand-children by n/8
  - So on

Task 2 Example

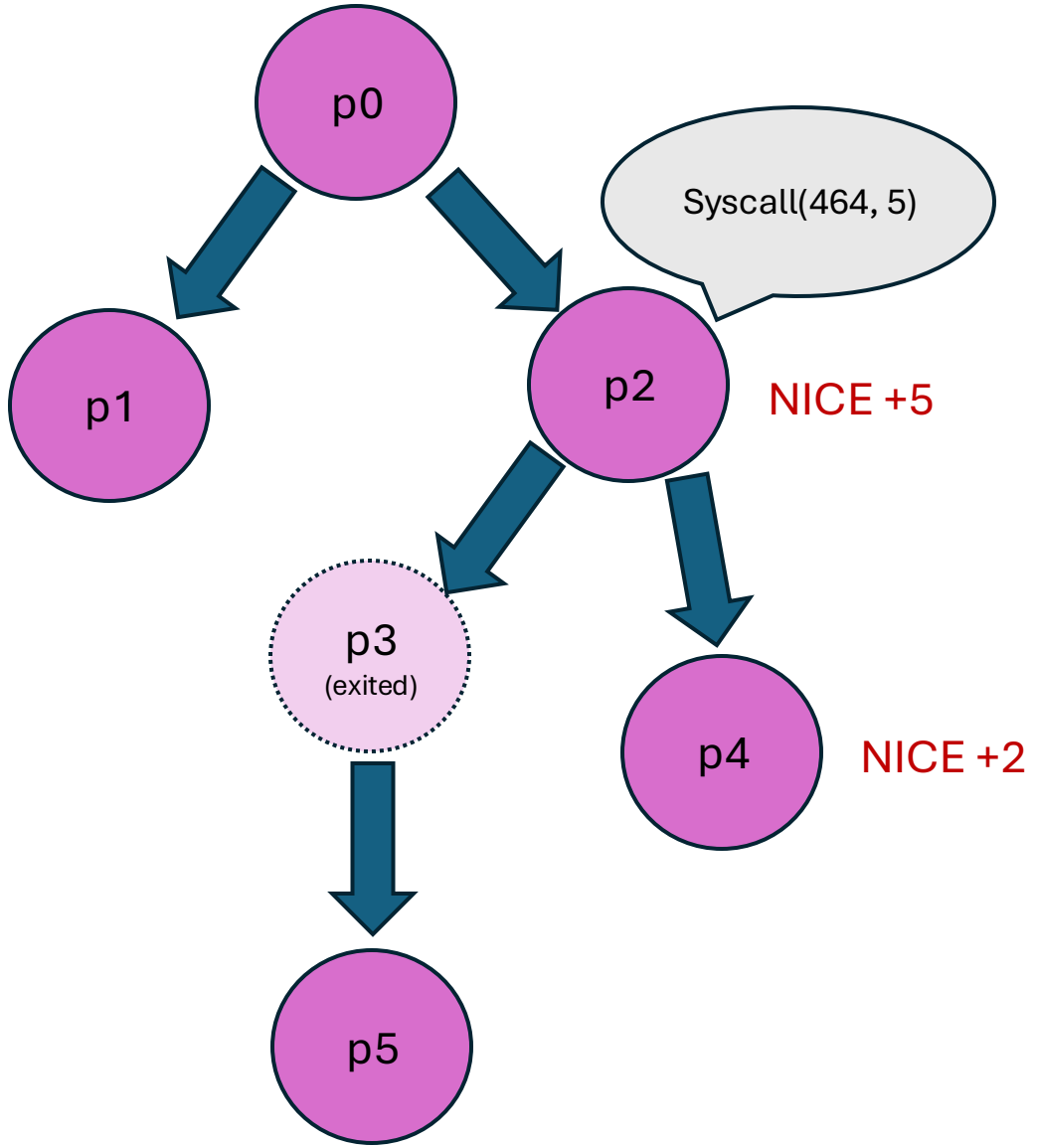
Initially all processes are at NICE=0



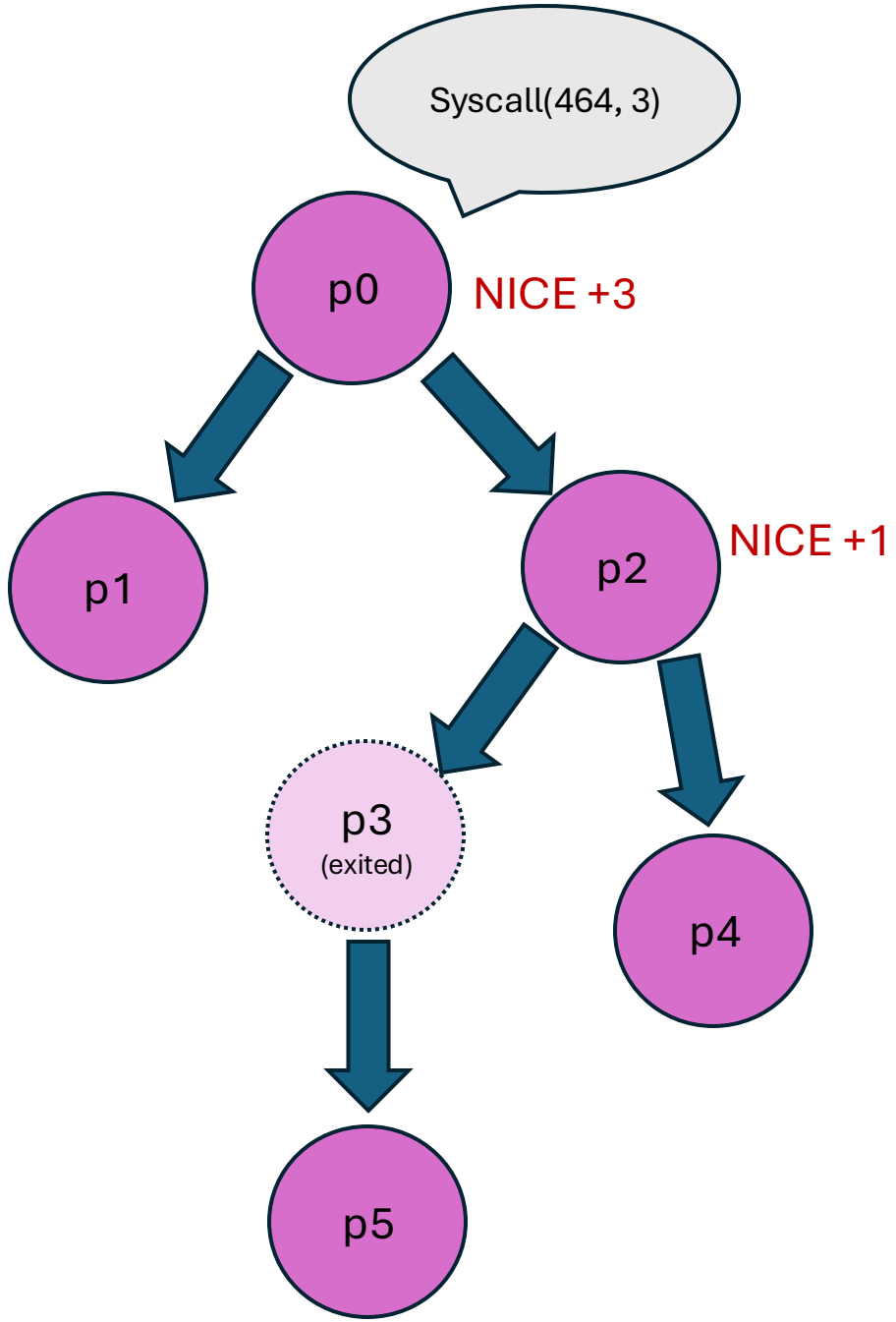
Task 2 Example



Task 2 Example

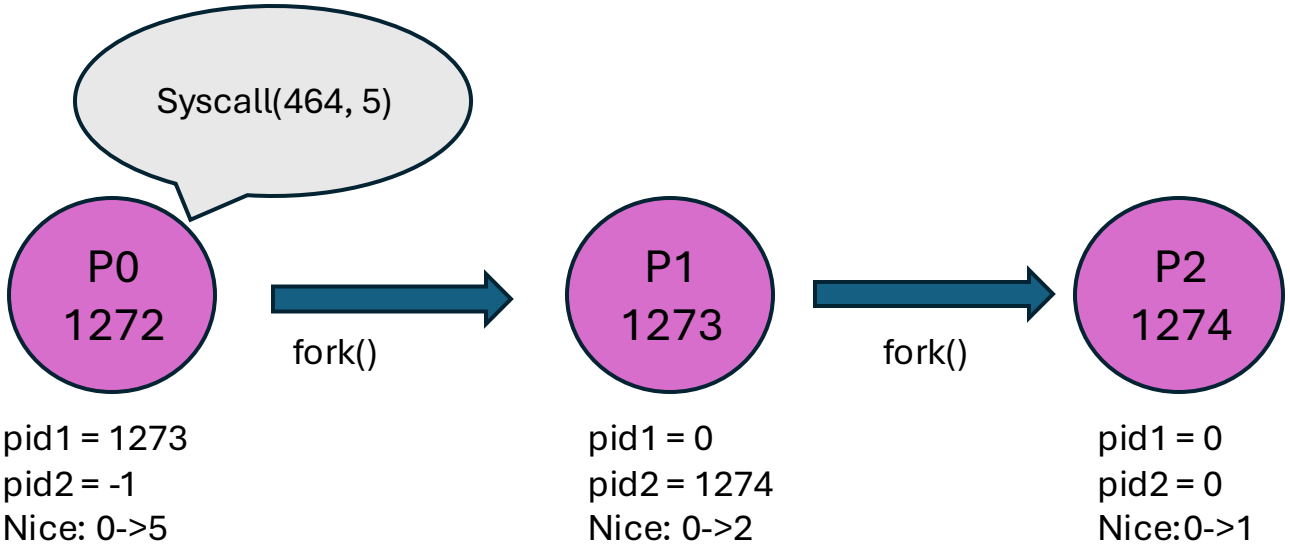


Task 2 Example



Task 2 - Testing

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/resource.h>

#define SYSCALL2 464

int main()
{
        pid_t pid1 = -1, pid2 = -1;

        pid1 = fork();
        if (pid1 == 0) {
                pid2 = fork();
        }

        printf("pid1:%d, pid2:%d, ni:%d\n",
                pid1, pid2, getpriority(PRIO_PROCESS, 0));

        sleep(1);
        if (pid2 == -1)
                syscall(SYSCALL2, 5);
        sleep(1);

        printf("pid1:%d, pid2:%d, ni:%d\n",
                pid1, pid2, getpriority(PRIO_PROCESS, 0));

        return 0;
}
```



OUTPUT

Before syscall

```
pid1:1273, pid2:-1, ni:0
pid1:0, pid2:1274, ni:0
pid1:0, pid2:0, ni:0
```

After syscall

```
pid1:1273, pid2:-1, ni:5
pid1:0, pid2:1274, ni:2
pid1:0, pid2:0, ni:1
```

Task 2 – Where to start?

- Find the incomplete syscall inside core.c
- COMPLETE IT!

- Check out:
  - struct task_struct in *include/linux/sched.h*
  - Find pointer to children tasks inside task_struct

```
7358    /**
7359     * CW1
7360     * sys_propagate_nice - trickle-down nice-increment to descendants
7361     * @increment: nice-increment for calling process
7362     *
7363     * Return: 0 on success. Error otherwise.
7364     */
7365    SYSCALL_DEFINE1(propagate_nice, int, increment)
7366    {
7367      return 0;
7368    }
```

Task 3 – Scheduling History

- Print scheduler stats inside /proc/<PID>/schedstat
  - By default it prints 3 numbers
  - In your base kernel, it will have a fourth string (enclosed by '[]')
  - Implement this functionality: This fourth string should print the list of CPUs this process was scheduled on in this epoch (epoch=10 seconds of runtime)
  - Time spent waiting on a runqueue should not be included in epoch! Only RUNTIME is counted.

Task 3 - Where to Start?

- Struct task_struct has 2 new members
  - used_cpus is a cpumask_t
    - When a process is scheduled on a CPU, set the CPU in used_cpus
  - Epoch_ticks is the number of scheduler ticks this process endured in this epoch.
    - Use this to track the end of an epoch (used_cpus must be reset every epoch).
- Core.c contains a macro TICKS_PER_EPOCH
  - Equals Number of scheduler ticks in 10 secs (epoch)

```
788          * Tracking a recently used CPU at
789          * used CPU that may be idle.
790          */
791         int             recent_used_cpu;
792         int             wake_cpu;
793
794         /* CW1 */
795         cpumask_t used_cpus;
796         unsigned int epoch_ticks;
797     #endif
798         int             on_rq;
799
800         int             prio;
```

```
117     EXPORT_TRACEPOINT_SYMBOL_GPL(sched_compute_energy_
118
119     DEFINE_PER_CPU_SHARED_ALIGNED(struct rq, runqueues
120
121     /* CW1 - an epoch equals 10 seconds */
122     #define TICKS_PER_EPOCH (10 * HZ)
123
124     #ifdef CONFIG_SCHED_DEBUG
125     /*
126      * Debugging: various feature bits
127      *
```

Best of Luck!