Operating systems – Assignment 2
Scheduling

Lennart Jern
CS: ens16ljn

Teacher
Ahmed Aley

December 12, 2016

1 Introduction

2 Implementation

3 Testing

Threads	Normal	Batch	Idle	FIFO	Round Robin
1	0.718	0.712	0.719	0.717	0.709
2	0.377	0.364	0.370	0.369	0.369
3	0.271	0.252	0.250	0.249	0.248
4	0.271	0.204	0.205	0.186	0.186
5	0.238	0.223	0.228	0.290	0.245
6	0.249	0.215	0.222	0.246	0.246
7	0.240	0.197	0.197	0.211	0.212
8	0.221	0.205	0.215	0.186	0.186
9	0.207	0.201	0.207	0.244	0.245
10	0.211	0.204	0.203	0.222	0.222

Threads	Normal	Batch	Idle	FIFO	Round Robin
1	0.761	0.746	0.751	0.746	0.751
2	0.398	0.398	0.405	0.381	0.378
3	0.340	0.299	0.281	0.252	0.250
4	0.310	0.241	0.297	0.188	0.188
5	0.280	0.265	0.273	0.292	0.248
6	0.279	0.283	0.282	0.249	0.248
7	0.290	0.251	0.269	0.212	0.212
8	0.231	0.243	0.261	0.187	0.187
9	0.307	0.261	0.266	0.246	0.247
10	0.263	0.245	0.270	0.222	0.222

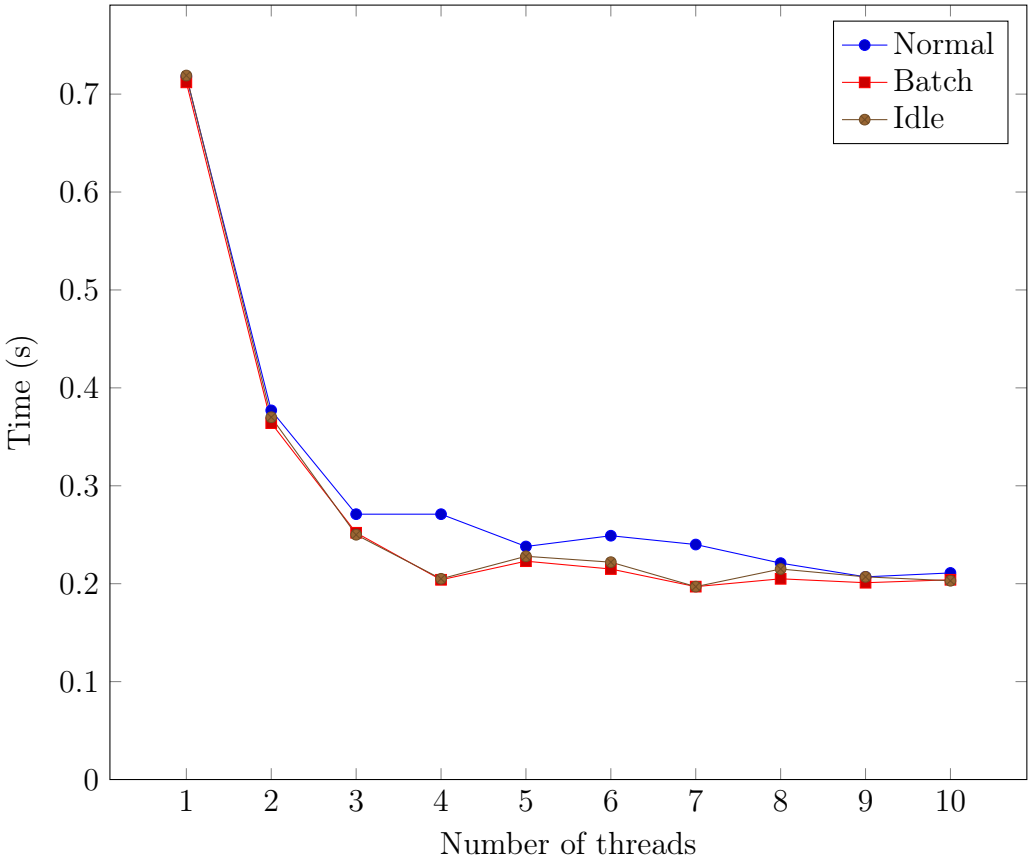

Figure 1: Median time

Threads	Normal	Batch	Idle	FIFO	Round Robin
1	0.700	0.699	0.701	0.698	0.693
2	0.362	0.358	0.361	0.358	0.366
3	0.254	0.245	0.246	0.245	0.246
4	0.208	0.191	0.192	0.186	0.186
5	0.227	0.220	0.222	0.289	0.243
6	0.209	0.210	0.210	0.243	0.243
7	0.206	0.191	0.190	0.210	0.211
8	0.206	0.201	0.205	0.186	0.186
9	0.200	0.194	0.200	0.243	0.244
10	0.205	0.194	0.193	0.220	0.221

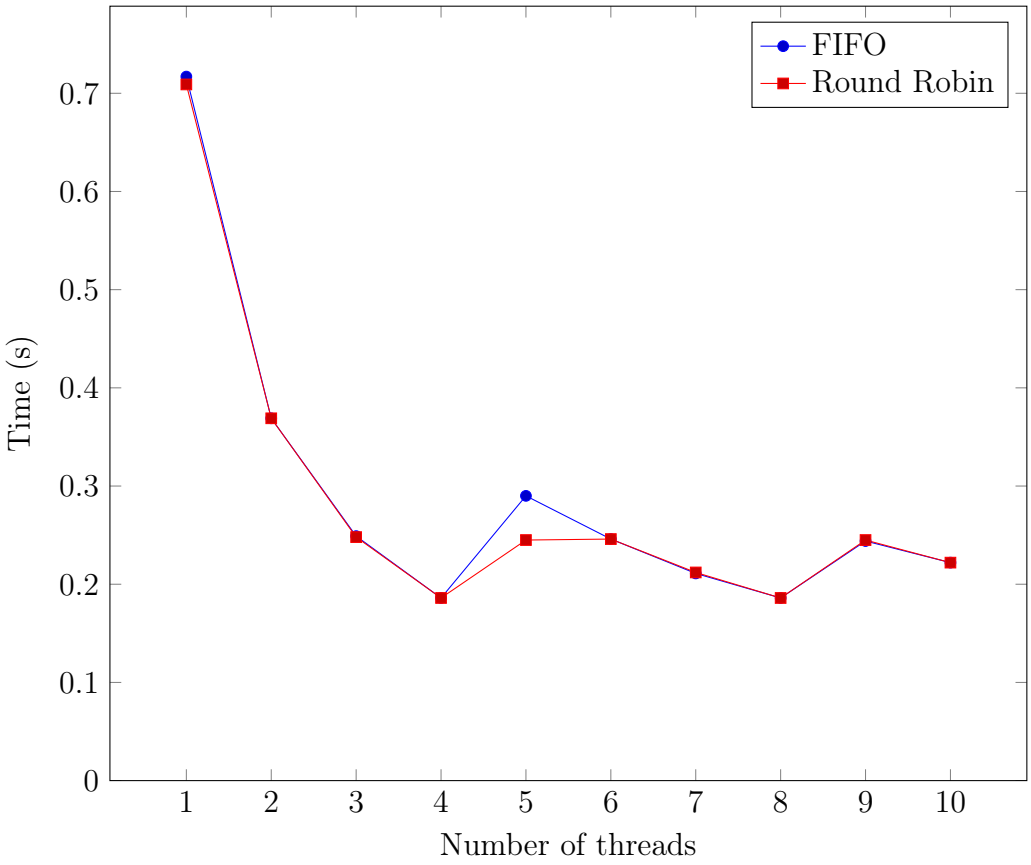

Figure 2: Median time for real time schedulers.

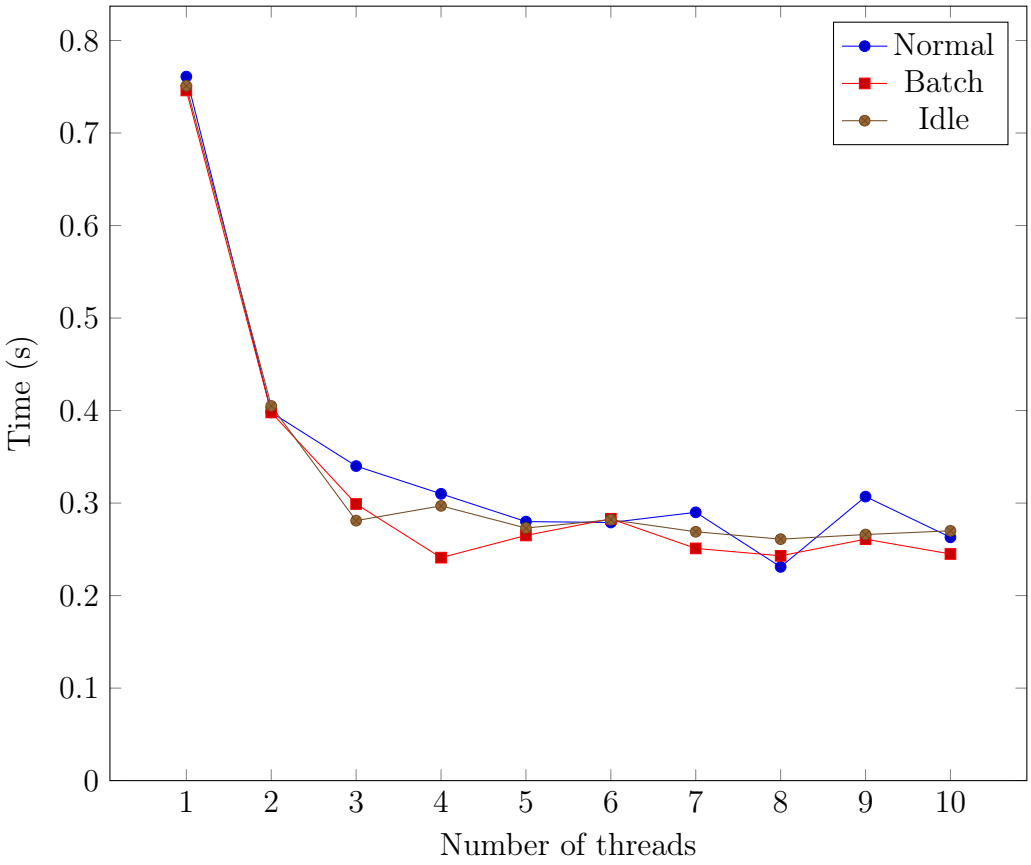

Figure 3: Maximum time

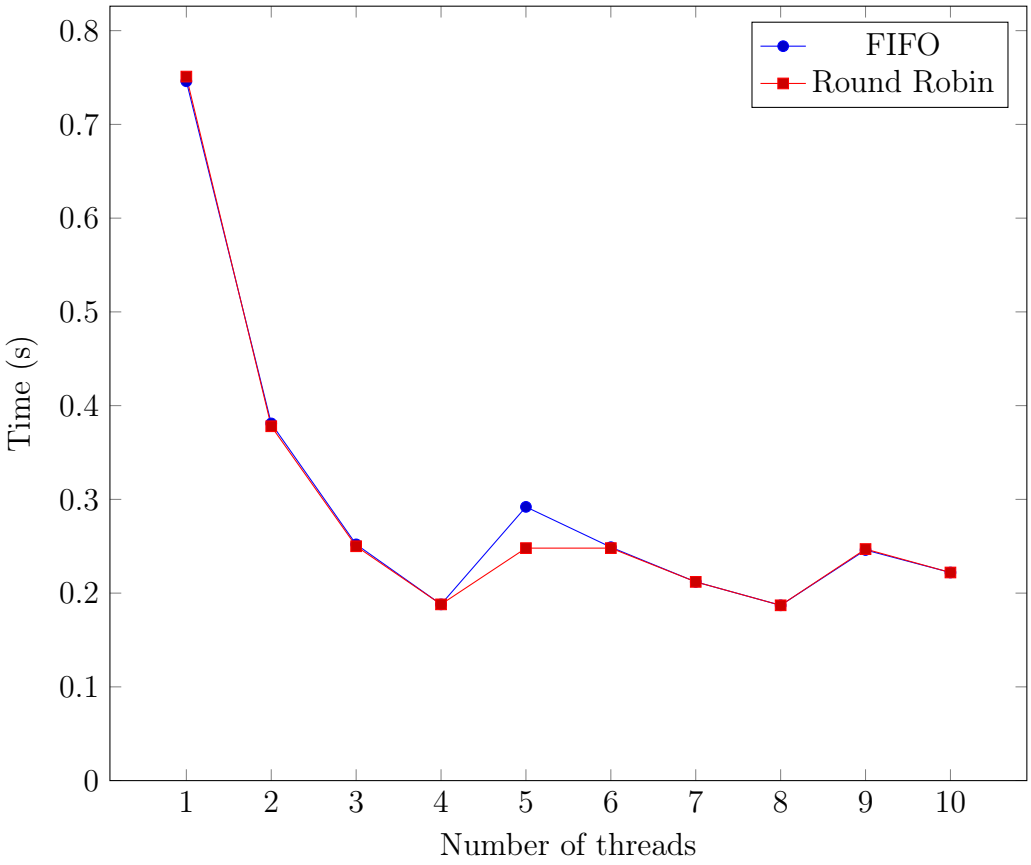

Figure 4: Maximum time for real time schedulers.

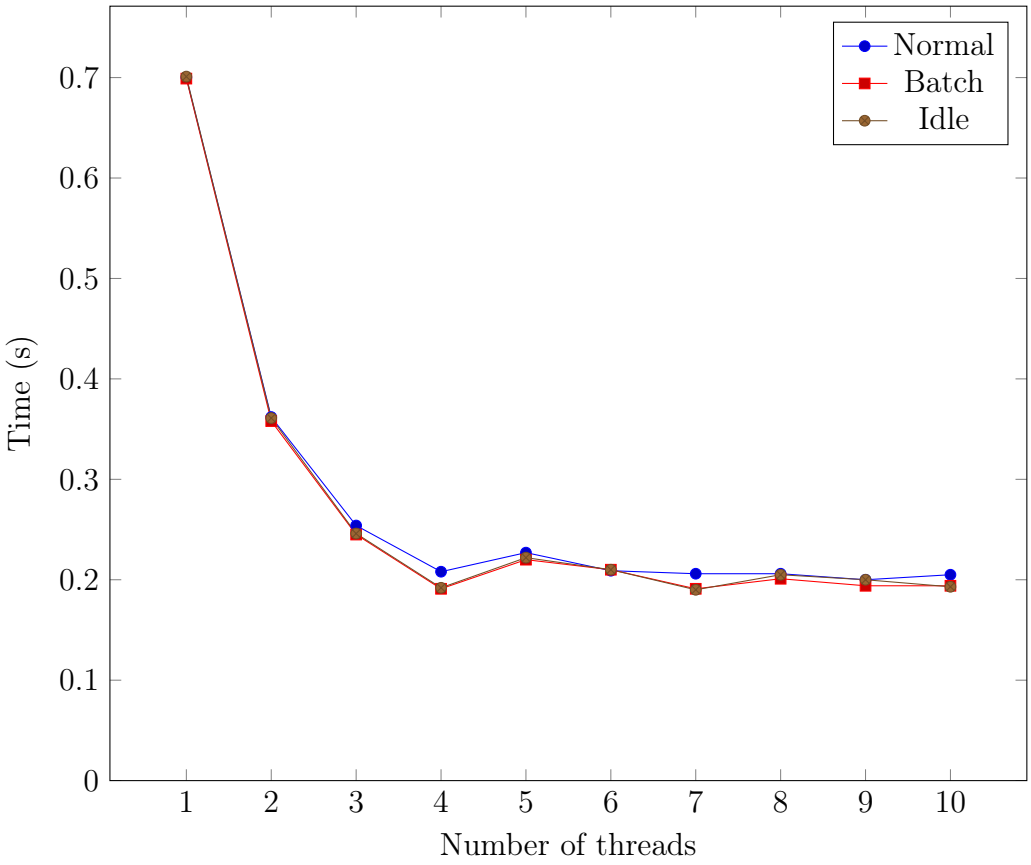

Figure 5: Minimum time

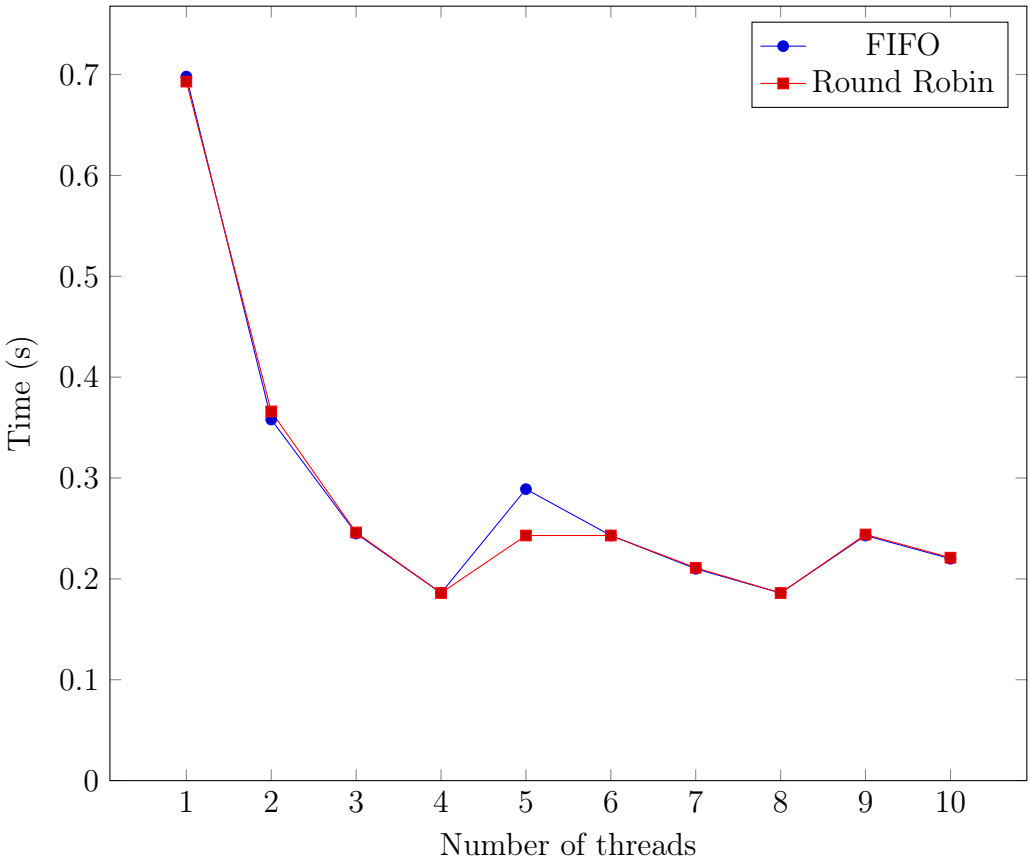

Figure 6: Minimum time for real time schedulers.

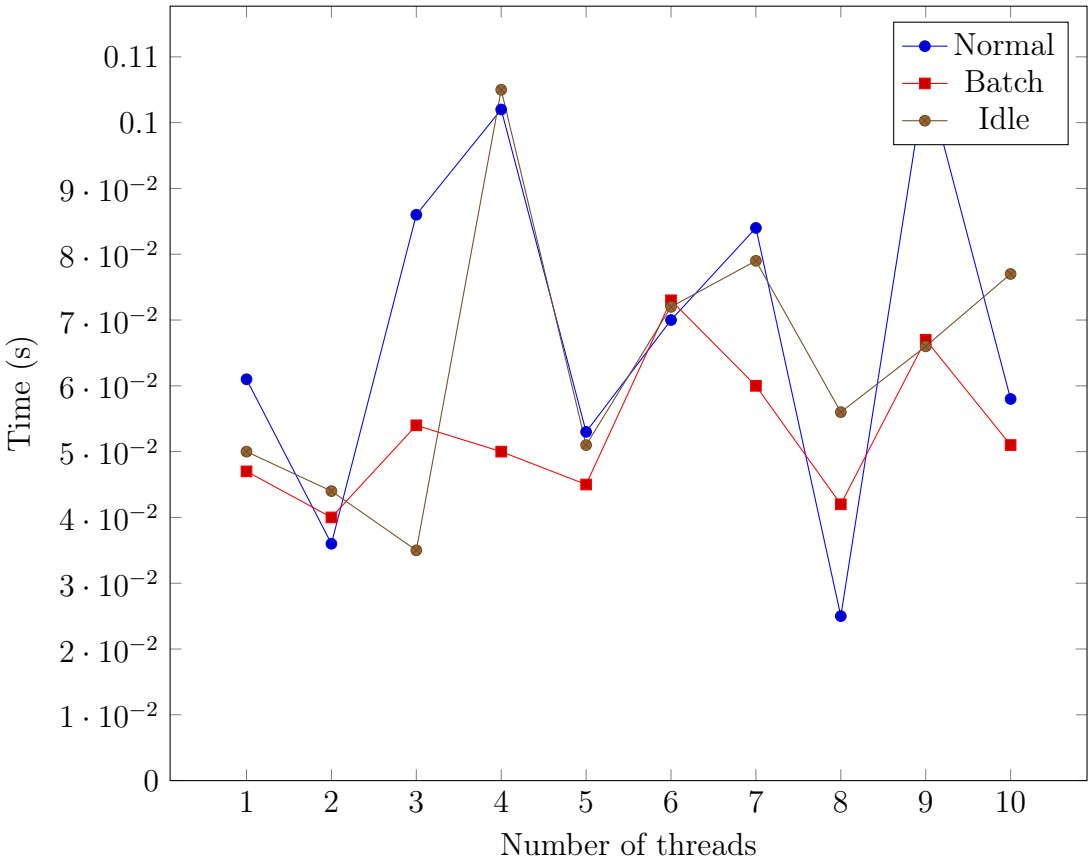

Figure 7: Spread

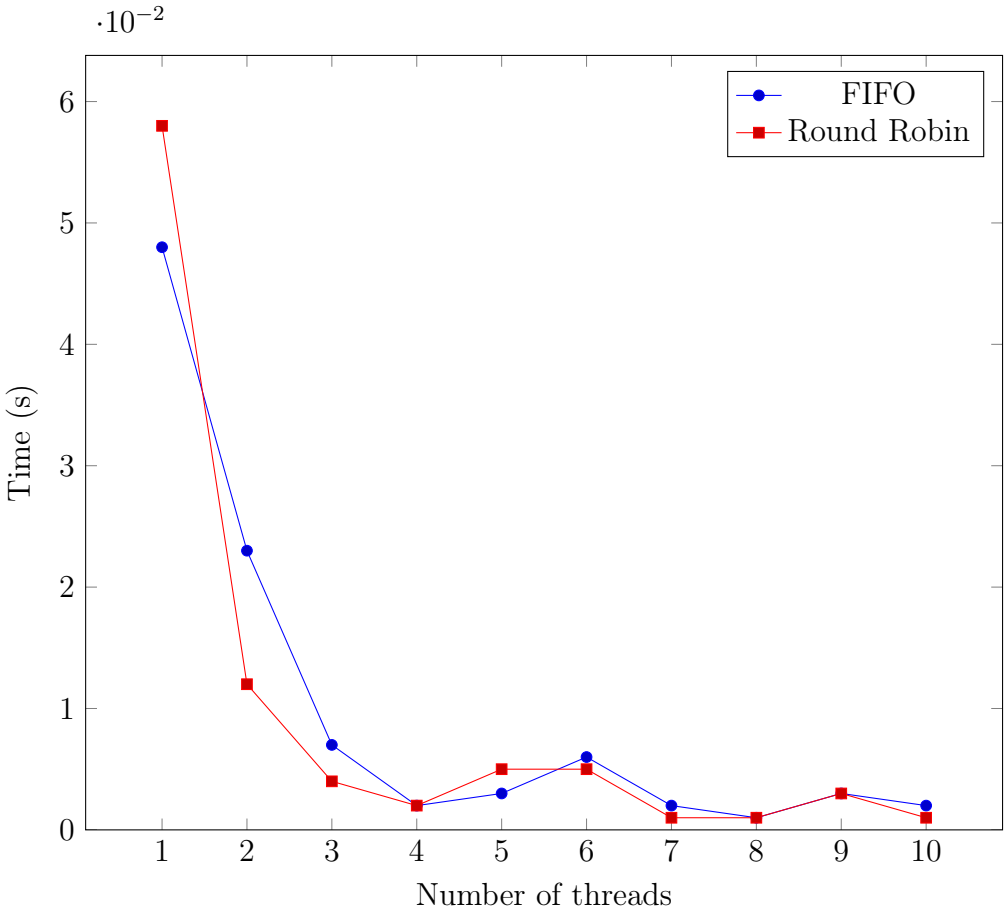

Figure 8: Spread for real time schedulers.

4 Final thoughts and lessons learned

A Code listings

```c
/**
 * work.c
 *
 * Just a silly "do something that takes time" program.
 * It tries to calculate the sum of Grandi's series (1-1+1-1+1-1...).
 * As long as the length it is summing over is even the sum should always be 0.
 *
 * Author: Lennart Jern (ens16ljn)
 *
 * Call sequence: work [-p <policy>] [-j <number of jobs>]
 * The policy is given by a single char according to this:
 * n - Normal
 * b - Batch
 * i - Idle
 * f - FIFO
 * r - RR
 * d - Deadline
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <pthread.h>        // threading
#include <sys/types.h>      // pid
#include <unistd.h>         // pid, getopt
#include <linux/sched.h>    // schduling policies

// Length of sequence to sum
//#define LENGTH 2147483400
#define LENGTH 214748340

typedef struct work_load {
    int nworkers;
    long data_length;
    char scheduler;
} WorkLoad;

typedef struct work_packet {
    long index;
    long length;
    long result;
} Packet;

WorkLoad *get_work_load();
void *work(void *data);
int get_grandi(int index);
long calculate_sum(long index, long length);
void run_workers(WorkLoad *wl);
void print_schduler();
void set_scheduler(WorkLoad *wl);
void set_settings(WorkLoad *wl, int argc, char *argv[]);

int num_policies = 6;
char c_policies[] = {'n', 'b', 'i', 'f', 'r', 'd'};
```

```c
char *str_policies[] = {"Normal", "Batch", "Idle", "FIFO", "RR", "Deadline"};
int policies[] = {SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE, SCHED_FIFO, SCHED_RR,
     SCHED_DEADLINE};

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

    WorkLoad *wl = get_work_load();

    set_settings(wl, argc, argv);

    // Set scheduler
    set_scheduler(wl);

    // Print scheduler to make sure it is set coorectly
    print_schduler();

    printf("Starting␣workers...\n");
    run_workers(wl);

    free(wl);
    printf("Done\n");
}

/**
 * get_work_load - initialize the work load and return a pointer to it
 * @return  pointer to allocated memory
 */
WorkLoad *get_work_load() {
    WorkLoad *wl;
    // Allocate memory for work load
    wl = malloc(sizeof(WorkLoad));
    // Initialize work load
    wl->nworkers = 1;
    // wl->data_length = 1073741824; // 2^30
    wl->data_length = LENGTH;
    return wl;
}

/**
* work - a silly attempt to calculate the limit of Grandi's series
* @param packet   the part of the work load to work on
* @return         nothing
*/
void *work(void *packet) {
    Packet *pkt = (Packet *)packet;

    long sum = 0;

    sum = sum + calculate_sum(pkt->index, pkt->length);
}

/**
 * get_grandi - calculate the i:th number of Grandi's series
 * @param  index    index of the number you want to know
 * @return          1 if index is even, -1 otherwise
 */
int get_grandi(int index) {
    if (index % 2 == 0) {
        return 1;
    } else {
        return -1;
    }
```

```
}

/**
 * calculate_sum - sum Grandi's series from index over a given length
 * @param  index    index to start from
 * @param  length   how many numbers to sum over
 * @return          the sum
 */
long calculate_sum(long index, long length) {
    long sum = 0;

    for (int i = index; i < index+length; i++) {
        sum = sum + get_grandi(i);
    }
    return sum;
}

/**
* run_workers - start work threads and wait for them to finish
*/
void run_workers(WorkLoad *wl) {
    int num = wl->nworkers;
    long total_sum = 0;

    Packet *pkt[num];
    for (int i = 0; i < num; i++) {
        pkt[i] = malloc(sizeof(Packet));
        if (!pkt[i]) {
            perror("malloc");
        }
        pkt[i]->result = 0;
    }

    // create threads
    pthread_t threads[num];
    long len = wl->data_length;
    long p_len = len / num;
    int i;
    for (i = 0; i < num-1; i++) {
        pkt[i]->index = i * len / num;
        pkt[i]->length = p_len;
        if (pthread_create(&threads[i], NULL, work, (void *)pkt[i]) != 0) {
            perror("Could not create thread");
        }
    }
    printf("Working...\n");
    pkt[i]->index = i * len / num;
    pkt[i]->length = p_len;
    work((void *)pkt[i]);

    total_sum += pkt[i]->result;
    free(pkt[i]);

    // Join the threads
    for (int i = 0; i < num-1; i++) {
        pthread_join(threads[i], NULL);
        total_sum += pkt[i]->result;
        free(pkt[i]);
    }

    printf("Sum is %d\n", total_sum);
}
```

```
178
179 /**
180 * print_schduler - print the current scheduler
181 * @param  pid  the pid of the process
182 */
183 void print_schduler() {
184     pid_t pid = getpid();
185     int schedlr = sched_getscheduler(pid);
186
187     char *schedlr_name;
188     switch (schedlr) {
189         case SCHED_NORMAL:
190         schedlr_name = "Normal/Other";
191         break;
192         case SCHED_BATCH:
193         schedlr_name = "Batch";
194         break;
195         case SCHED_IDLE:
196         schedlr_name = "Idle";
197         break;
198         case SCHED_FIFO:
199         schedlr_name = "FIFO";
200         break;
201         case SCHED_RR:
202         schedlr_name = "RR";
203         break;
204         case SCHED_DEADLINE:
205         schedlr_name = "Deadline";
206         break;
207         default:
208         schedlr_name = "Unknown";
209     }
210     printf("Scheduler:␣%s\n", schedlr_name);
211 }
212
213 /**
214  * set_scheduler - update scheduler to reflect the given WorkLoad
215  * @param wl    the work load
216  */
217 void set_scheduler(WorkLoad *wl) {
218     struct sched_param param;
219     pid_t pid = getpid();
220     int policy = SCHED_NORMAL;
221
222     for (int i = 0; i < num_policies; i++) {
223         if (wl->scheduler == c_policies[i]) {
224             policy = policies[i];
225             break;
226         }
227     }
228
229     // Set the priority
230     param.sched_priority = sched_get_priority_max(policy);
231
232     if (sched_setscheduler(pid, policy, &param) != 0) {
233         perror("Set␣scheduler");
234     }
235 }
236
237 /**
238  * set_settings - parse arguments and set the settings for the work load
239  * @param wl    the work load to update
```

```
 * @param argc  argument cound
 * @param argv  array of arguments
 */
void set_settings(WorkLoad *wl, int argc, char *argv[]) {
    // Two possible options: j(obs) and p(olicy)
    char *optstr = "j:p:";
    int opt;
    char policy = 'n';
    int num_threads = 1;
    int policy_ok = 0;
    int threads_ok = 0;

    // Parse flags
    while ((opt = getopt(argc, argv, optstr)) != -1) {
        char *end;
        switch (opt) {
            case 'p':
            policy = *optarg;
            break;
            case 'j':
            errno = 0;
            num_threads = strtol(optarg, &end, 10);
            if (errno != 0) {
                perror("strtol");
            }
            break;
            default:
            printf("Option␣%c␣not␣supported\n", opt);
        }
    }

    // Check the parsed options
    for (int i = 0; i < num_policies; i++) {
        if (policy == c_policies[i]) {
            policy_ok = 1;
            break;
        }
    }

    if (num_threads <= 100 && num_threads > 0) {
        threads_ok = 1;
    }

    // Set values if they are safe, or set defaults
    if (policy_ok) {
        wl->scheduler = policy;
    } else {
        wl->scheduler = 'n';
    }

    if (threads_ok) {
        wl->nworkers = num_threads;
    } else {
        wl->nworkers = 1;
    }
}
```

```
#!/bin/bash

# A timer script to measure the differences between schedulers/policies
#
```

```
# Author: Lennart Jern (ens16ljn@cs.umu.se)

for THREADS in $(seq 1 10)
do
    DATA="Normal,Batch,Idle,FIFO,Round␣Robin"
    echo "Running␣with␣$THREADS␣threads"
    # Time the commands 10 times
    for i in $(seq 1 10)
    do
        LINE=""
        # For the polices n(ormal) b(atch) and i(dle)
        for POLICY in n b i f r
        do
            # Running with 4 threads/jobs
            FLAGS="-p$POLICY␣-j$THREADS"
            COMMAND="./work␣$FLAGS␣>␣/dev/null"
            # Run the command and store the time
            t="$(sh␣-c␣"TIMEFORMAT='%5R'; time $COMMAND"␣2>&1)"
            # Build the line
            if [ "$POLICY" = "n" ]; then
                LINE="$t"
            else
                LINE="$LINE,$t"
            fi
        done
        DATA=$DATA$'\n'$LINE
        # A little progress report
        echo "Run␣$i␣done."
    done

    # Write data to a file
    echo "$DATA" > "data$THREADS.csv"

done
```

B Raw data