Assignment 2: Scheduling Policy Demonstration Program

311552067 劉承熙

1. Describe how you implemented the program in detail

* Using library to parse the argument

/\* 1. Parse program arguments \*/

int opt;

static struct option long\_options[] = {

{"num\_threads", 1, 0, 'n'},

{"time\_wait", 1, 0, 't'},

{"policies", 1, 0, 's'},

{"priorities", 1, 0, 'p'},

{"help", 0, 0, '?'},

{0 ,0, 0, 0}

};

int num\_threads = 0;

string policies = "";

string priorities = "";

while ((opt = getopt\_long(argc, argv, "n:t:s:p:h", long\_options,NULL)) != EOF) {

switch (opt) {

case 'n':

num\_threads = atoi(optarg);

break;

case 't':

time\_wait = atof(optarg);

break;

case 's':

policies = optarg;

break;

case 'p':

priorities = optarg;

break;

case 'h':

usage(argv[0]);

default:

usage(argv[0]);

return 1;

}

}

void usage(const char\* progname) {

cout << "Usage: "<< progname <<" [options]\n";

cout << "Program Options:\n";

cout << " -n <num\_threads>: number of threads to run simultaneously\n";

cout << " -t <time\_wait>: duration of \"busy\" period\n";

cout << " -s <policies>: scheduling policy for each thread, SCHED\_FIFO or SCHED\_NORMAL\n";

cout << " -p <priorities>: real-time thread priority for real-time threads\n";

}

* Define thread and thread attribute, then set affinity, schedule\_policy and schedule\_priority for any thread.  
  pthread\_attr\_init — for initial thread attribute  
  pthread\_attr\_setaffinity\_np — set cpuset (all thread using same core) to thread attribute  
  pthread\_attr\_setschedpolicy — set schedule\_policy (SCHED\_FIFO or SCHED\_OTHER) to thread attribute   
  pthread\_attr\_setschedparam — set sched\_priority to thread attribute pthread\_attr\_setinheritsched — set PTHREAD\_EXPLICIT\_SCHED to thread attribute, ensure thread will create by our setting

pthread\_t threads[num\_threads];

int thread\_id[num\_threads];

pthread\_attr\_t thread\_attr[num\_threads];

/\* 3. Set CPU affinity \*/

cpu\_set\_t cpuset;

CPU\_ZERO(&cpuset);

CPU\_SET(3, &cpuset); // Set affinity to CPU 0

for (int i = 0; i < num\_threads; i++) {

/\* 4. Set the attributes to each thread \*/

pthread\_attr\_init(&thread\_attr[i]);

thread\_id[i] = i;

size\_t pos = policies.find(",");

string policy\_str = policies.substr(0, pos);

int sched\_policy = (policy\_str == "FIFO") ? SCHED\_FIFO : SCHED\_OTHER;

policies.erase(0, pos + 1);

pos = priorities.find(",");

string priority\_str = priorities.substr(0, pos);

int sched\_priority = stoi(priority\_str);

priorities.erase(0, pos + 1);

if (pthread\_attr\_setaffinity\_np(&thread\_attr[i], sizeof(cpu\_set\_t), &cpuset) != 0) {

perror("Error setting CPU affinity");

pthread\_exit(NULL);

}

if (pthread\_attr\_setschedpolicy(&thread\_attr[i], sched\_policy) != 0) {

perror("Error setting scheduling policy");

pthread\_exit(NULL);

}

if(sched\_priority != -1){

struct sched\_param param;

param.sched\_priority = sched\_priority;

if (pthread\_attr\_setschedparam(&thread\_attr[i], &param) != 0) {

perror("Error setting scheduling parameters");

pthread\_exit(NULL);

}

}

pthread\_attr\_setinheritsched(&thread\_attr[i],PTHREAD\_EXPLICIT\_SCHED);

}

* pthread\_barrier\_init — set barrier for waitting threas, then start all threads at once  
  Create thread with the previously configured thread attributes

/\* 2. Create <num\_threads> worker threads \*/

/\* 5. Start all threads at once \*/

pthread\_barrier\_init(&barrier, NULL, num\_threads);

for (int i = 0; i < num\_threads; i++) {

if (pthread\_create(&threads[i], &thread\_attr[i], thread\_func, &thread\_id[i]) != 0) {

perror("Error creating thread");

exit(EXIT\_FAILURE);

}

}

* pthread\_barrier\_wait — set barrier for making start all threads at once

void \*thread\_func(void \*arg)

{

/\* 1. Wait until all threads are ready \*/

pthread\_barrier\_wait(&barrier);

/\* 2. Do the task \*/

for (int i = 0; i < 3; i++) {

printf("Thread %d is running\n", \*(int\*)(arg));

/\* Busy for <time\_wait> seconds \*/

busy\_wait(time\_wait);

}

/\* 3. Exit the function \*/

pthread\_exit(NULL);

}

* Waitting all thread finishing execution  
  pthread\_attr\_destroy — for delete all thread attribute pthread\_barrier\_destroy — for delete barrier

/\* 6. Wait for all threads to finish \*/

for (int i = 0; i < num\_threads; i++) {

if (pthread\_join(threads[i], NULL) != 0) {

perror("Error joining thread");

exit(EXIT\_FAILURE);

}

}

for (int i = 0; i < num\_threads; i++) {

pthread\_attr\_destroy(&thread\_attr[i]);

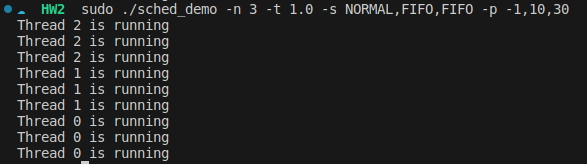
}

pthread\_barrier\_destroy(&barrier);

1. Describe the results of ./sched\_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30 and what causes that

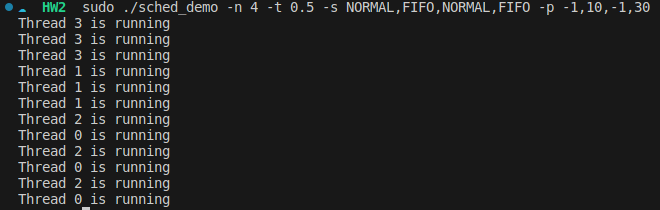
Threat 2 and Thread 1 are scheduled using the SCHED\_FIFO (Real-Time scheduling policies). Therefore, they are both prioritized and processed based on their assigned priorities. Since Thread 2 has a higher priority number, it takes precedence in utilizing the core, and only after Thread 2 completes its execution, Thread 1 is scheduled

Thread 0 is scheduled using SCHED\_NORMAL (Fair scheduling policies). Since there are no other SCHED\_NORMAL threads, it completes its execution directly



1. Describe the results of ./sched\_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30, and what causes that

Threat 3 and Thread 1 are scheduled using the SCHED\_FIFO (Real-Time scheduling policies). Therefore, they are both prioritized and processed based on their assigned priorities. Since Thread 3 has a higher priority number, it takes precedence in utilizing the core, and only after Thread 3 completes its execution, Thread 1 is scheduled

Thread 0 and Thread 2 use the SCHED\_NORMAL (Fair scheduling policies), so they will use the CPU after SCHED\_FIFO threads. They will take turns utilizing the CPU

1. Describe how did you implement n-second-busy-waiting

Using the <ctime> library's clock() function, calculate whether the required seconds have elapsed within a while loop.

void busy\_wait(double seconds) {

double start\_time = (double)clock() / CLOCKS\_PER\_SEC;

double current\_time;

do {

current\_time = (double)clock() / CLOCKS\_PER\_SEC;

} while ((current\_time - start\_time) < seconds);

}