

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <pthread.h>
5  #include <sched.h>
6  #include <sys/syscall.h>
7  #define _GNU_SOURCE
8
9
10 #include <sys/syscall.h>
11
12 #ifndef SYS_gettid
13 #error "SYS_gettid unavailable on system"
14 #endif
15
16 #define gettid() ((pid_t)syscall(SYS_gettid))
17
18 // Define the mutex as global variable
19 pthread_mutex_t lock;
20 pthread_mutexattr_t attributes;
21
22 // Define a long loop to slow down the thread
23 const long int LOOP_LARGE_NUMBER = 500000000;
24 const long int LOOP_10PERCENT_NUMBER = LOOP_LARGE_NUMBER / 10;
25
26 // Define the scheduling policy
27 int policy = SCHED_FIFO;
28
29 int get_real_time_priority() {
30     char str[50];
31     sprintf(str, "/proc/%d/stat", gettid());
32     FILE *fp = fopen(str, "r");
33     for(int i = 0; i < 17; i++) {
34         fscanf(fp, "%s", str);
35     }
36     int rt_priority = 0;
37     fscanf(fp, "%d", &rt_priority);
38     fclose(fp);
39     return -rt_priority-1;
40 }
41
42 void *function1() {
43     printf("Thread 1 start\n");
44     fflush(stdout);
45
46     printf("Thread 1 requests the lock\n");
47     fflush(stdout);
48
49     pthread_mutex_lock(&lock);
50
51     printf("Thread 1 had the lock\n");
52     fflush(stdout);
53
54     int counter = 0;
55     for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
56     {
57         if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
58             counter += 1;
59             printf("Thread 1 running, priority %d, process %d0%%...\n", get_real_time_prio
```


Documents pthread.c - /home/pi/...

Document Project Build Tools Help

with_preemption.c x rms_with_preemption.c x pthread.c x

```
for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
{
    if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
        counter += 1;
        printf("Thread 1 running, priority %d, process %d0%...\n", get_real_time_priority(), counter);
        fflush(stdout);
    }
}

printf("Thread 1 released the lock\n");
fflush(stdout);

pthread_mutex_unlock(&lock);

printf("Thread 1 complete\n");
fflush(stdout);
}

void *function2() {
    printf("Thread 2 start\n");
    fflush(stdout);

    printf("Thread 2 requests the lock\n");
    fflush(stdout);

    pthread_mutex_lock(&lock);

    printf("Thread 2 had the lock\n");
    fflush(stdout);

    int counter = 0;
    for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
    {
        if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
            counter += 1;
            printf("Thread 2 running, priority %d, process %d0%...\n", get_real_time_priority(), counter);
            fflush(stdout);
        }
    }

    printf("Thread 2 released the lock\n");
    fflush(stdout);

    pthread_mutex_unlock(&lock);

    printf("Thread 2 complete\n");
    fflush(stdout);
}

void *function3() {
    printf("Thread 3 start\n");
    fflush(stdout);

    printf("Thread 3 requests the lock\n");
    fflush(stdout);
}
```

ln: 0 INS T/S mode: CRLF encoding: UTF-8 filetype: C scope: unknown


```
pthread_mutex_unlock(&lock);

printf("Thread 2 complete\n");
fflush(stdout);
}

void *function3() {
    printf("Thread 3 start\n");
    fflush(stdout);

    printf("Thread 3 requests the lock\n");
    fflush(stdout);

    pthread_mutex_lock(&lock);

    printf("Thread 3 had the lock\n");
    fflush(stdout);

    int counter = 0;
    for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
    {
        if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
            counter += 1;
            printf("Thread 3 running, priority %d, process %d0%...\n", get_real_time_priority(), counter);
            fflush(stdout);
        }
    }

    printf("Thread 3 released the lock\n");
    fflush(stdout);

    pthread_mutex_unlock(&lock);

    printf("Thread 3 complete\n");
    fflush(stdout);
}

int main() {
    //Create mutex and initialize it.

    pthread_mutexattr_setprotocol(&attributes, PTHREAD_PRIO_INHERIT);
    pthread_mutex_init(&lock, &attributes);

    //Check the priority range
    int maxpriority = sched_get_priority_max(policy); // get max priority
    int minpriority = sched_get_priority_min(policy); // get min priority
    printf("Priority range: [%d, %d]\n", minpriority, maxpriority);

    //Make sure the priority of main thread is the highest
    struct sched_param param_main = { 0 };
    int priority_mainthread = 99; // Highest priority
    param_main.sched_priority = priority_mainthread;
    pthread_setschedparam(pthread_self(), policy, &param_main);
    printf("Priority of main thread: %d\n", get_real_time_priority());
}
```


edf_with_preemption.c x rms_with_preemption.c x pthread.c x

```
140
141 int main() {
142     //Create mutex and initialize it.
143
144     pthread_mutexattr_setprotocol(&attributes, PTHREAD_PRIO_INHERIT);
145     pthread_mutex_init(&lock, &attributes);
146
147     //Check the priority range
148     int maxpriority = sched_get_priority_max(policy); // get max priority
149     int minpriority = sched_get_priority_min(policy); // get min priority
150     printf("Priority range: [%d, %d]\n", minpriority, maxpriority);
151
152     //Make sure the priority of main thread is the highest
153     struct sched_param param_main = { 0 };
154     int priority_mainthread = 99; // Highest priority
155     param_main.sched_priority = priority_mainthread;
156     pthread_setschedparam(pthread_self(), policy, &param_main);
157     printf("Priority of main thread: %d\n", get_real_time_priority());
158
159
160
161     int priority3 = 1;
162     pthread_t thread3 = { 0 };
163     struct sched_param param3 = { 0 };
164     param3.sched_priority = priority3;
165     pthread_attr_t attr3;
166     pthread_attr_init(&attr3);
167     pthread_attr_setinheritsched(&attr3, PTHREAD_EXPLICIT_SCHED);
168     pthread_attr_setschedpolicy(&attr3, policy);
169     pthread_attr_setschedparam(&attr3, &param3);
170
171     printf("Creating thread3...\n");
172     fflush(stdout);
173     pthread_create(&thread3, &attr3, function3, NULL);
174     sleep(1);
175
176
177
178     //Initiate thread 1
179     int priority1 = 10; //define your own priority
180     pthread_t thread1 = { 0 };
181     struct sched_param param1 = { 0 };
182     param1.sched_priority = priority1;
183     pthread_attr_t attr1;
184     pthread_attr_init(&attr1);
185     pthread_attr_setinheritsched(&attr1, PTHREAD_EXPLICIT_SCHED);
186     pthread_attr_setschedpolicy(&attr1, policy);
187     pthread_attr_setschedparam(&attr1, &param1);
188
189     //Create thread 1
190     printf("Creating thread1...\n");
191     fflush(stdout);
192     pthread_create(&thread1, &attr1, function1, NULL);
193     sleep(1);
194
195
196
197
198
199     int priority2 = 5; //define your own priority
```

col: 0 sel: 0 INS T/S mode: CRLF encoding: UTF-8 filetype: C scope: unknown


```

165 pthread_attr_t attr3;
166 pthread_attr_init(&attr3);
167 pthread_attr_setinheritsched(&attr3, PTHREAD_EXPLICIT_SCHED);
168 pthread_attr_setschedpolicy(&attr3, policy);
169 pthread_attr_setschedparam(&attr3, &param3);
170
171 printf("Creating thread3...\n");
172 fflush(stdout);
173 pthread_create(&thread3, &attr3, function3, NULL);
174 sleep(1);
175
176
177
178 //Initiate thread 1
179 int priority1 = 10; //define your own priority
180 pthread_t thread1 = { 0 };
181 struct sched_param param1 = { 0 };
182 param1.sched_priority = priority1;
183 pthread_attr_t attr1;
184 pthread_attr_init(&attr1);
185 pthread_attr_setinheritsched(&attr1, PTHREAD_EXPLICIT_SCHED);
186 pthread_attr_setschedpolicy(&attr1, policy);
187 pthread_attr_setschedparam(&attr1, &param1);
188
189 //Create thread 1
190 printf("Creating thread1...\n");
191 fflush(stdout);
192 pthread_create(&thread1, &attr1, function1, NULL);
193 sleep(1);
194
195
196
197
198
199 int priority2 = 5; //define your own priority
200 pthread_t thread2 = { 0 };
201 struct sched_param param2 = { 0 };
202 param2.sched_priority = priority2;
203 pthread_attr_t attr2;
204 pthread_attr_init(&attr2);
205 pthread_attr_setinheritsched(&attr2, PTHREAD_EXPLICIT_SCHED);
206 pthread_attr_setschedpolicy(&attr2, policy);
207 pthread_attr_setschedparam(&attr2, &param2);
208
209 //Create thread 1
210 printf("Creating thread2...\n");
211 fflush(stdout);
212 pthread_create(&thread2, &attr2, function2, NULL);
213 sleep(1);
214
215
216 pthread_join(thread3, NULL);
217 pthread_join(thread1, NULL);
218 pthread_join(thread2, NULL);
219
220
221 return 0;
222 }
223

```

col: 0 sel: 0 INS T/S mode: CRLF encoding: UTF-8 filetype: C scope: unknown